Bayesian Artificial Intelligence

Kevin B. Korb
Ann E. Nicholson

CHAPMAN & HALL/CRC

A CRC Press Company
Boca Raton London New York Washington, D.C.

© 2004 by Chapman & Hall/CRC Press LLC

Chapman & Hall/CRC

Series in Computer Science and Data Analysis

The interface between the computer and statistical sciences is increasing, as each discipline seeks to harness the power and resources of the other. This series aims to foster the integration between the computer sciences and statistical, numerical and probabilistic methods by publishing a broad range of reference works, textbooks and handbooks.

SERIES EDITORS
John Lafferty, Carnegie Mellon University
David Madigan, Rutgers University
Fionn Murtagh, Queen's University Belfast
Padhraic Smyth, University of California Irvine

Proposals for the series should be sent directly to one of the series editors above, or submitted to:

Chapman & Hall/CRC Press UK
23-25 Blades Court
London SW15 2NU
UK

© 2004 by Chapman & Hall/CRC Press LLC

Library of Congress Cataloging-in-Publication Data

Korb, Kevin B.
Bayesian artificial intelligence / Kevin B. Korb, Ann E. Nicholson.
p. cm. — (Chapman & Hall/CRC computer science and data analysis)
Includes bibliographical references and index.
ISBN 1-58488-387-1 (alk. paper)
1. Bayesian statistical decision theory—Data processing. 2. Machine learning. 3. Neural networks (Computer science) I. Nicholson, Ann E. II. Title. III. Series.

QA279.5.K67 2003
519.5′42—dc21 2003055428

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

The consent of CRC Press LLC does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press LLC for such copying.

Direct all inquiries to CRC Press LLC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation, without intent to infringe.

Visit the CRC Press Web site at www.crcpress.com

© 2004 by Chapman & Hall/CRC

No claim to original U.S. Government works
International Standard Book Number 1-58488-387-1
Library of Congress Card Number 2003055428
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

© 2004 by Chapman & Hall/CRC Press LLC

To Judea Pearl and Chris Wallace

© 2004 by Chapman & Hall/CRC Press LLC

Preface

Bayesian Artificial Intelligence, in our understanding, is the incorporation of Bayesian inferential methods in the development of a software architecture for an Artificial Intelligence (AI). We believe that important ingredients of such an architecture will be Bayesian networks and the Bayesian learning of Bayesian networks (Bayesian causal discovery) from observation and experiment. In this book we present the elements of Bayesian network technology, automated causal discovery, learning probabilities from data, and examples and ideas about how to employ these technologies in developing probabilistic expert systems, which we call Knowledge Engineering with Bayesian Networks.

This is a very practical project, because data mining with Bayesian networks (applied causal discovery) and the deployment of Bayesian networks in industry and government are two of the most promising areas in applied AI today. But it is also a very theoretical project, because the achievement of a Bayesian AI would be a major theoretical achievement.

With our title there are a number of subjects we could naturally include, but have not. Thus, another necessary aspect of an effective Bayesian AI will be the learning of concepts, and hierarchies of concepts. Bayesian methods for concept formation exist (e.g., Chris Wallace's Snob [290]), but we do not treat them here. We could also have discussed Bayesian methods of classification, polynomial curve fitting, time series modeling, etc. We have chosen to hew close to the theme of using and discovering Bayesian networks both because this is our own main research area and because, important as the other Bayesian learning methods are, we believe the Bayesian network technology is central to the overall project.

Our text differs from others available on Bayesian networks in a number of ways. We aim at a practical and accessible introduction to the main concepts in the technology, while paying attention to foundational issues. Most texts in this area require somewhat more mathematical sophistication than ours; we presuppose only a basic understanding of algebra and calculus. Also, we give roughly equal weight to the causal discovery of networks and to the Bayesian inference procedures using a network once found. Most texts either ignore causal discovery or treat it lightly. Richard Neapolitan's recent book, *Learning Bayesian Networks* [200], is an exception, but it is more technically demanding than ours. Another distinguishing feature of our text is that we advocate a causal interpretation of Bayesian networks, and we discuss the use of Bayesian networks for causal modeling. We also illustrate various applications of the technology at length, drawing upon our own applied research. We hope that these illustrations will be of some interest and indicate some of the possibilities

© 2004 by Chapman & Hall/CRC Press LLC

for the technology. Our text is aimed at advanced undergraduates in computer science who have some background in artificial intelligence and at those who wish to engage in applied or pure research in Bayesian network technology.

The book Web site is

http://www.csse.monash.edu.au/bai

and contains a variety of aids for study, including example Bayesian networks and data sets. Instructors can email us for sample solutions to many of the problems in the text.

There are many whom we wish to acknowledge. For assistance reviewing portions of the text we thank: David Albrecht, Helen Armstrong, Tali Boneh, Darren Boulton, Mark Burgman, Steven Gardner, Lucas Hope, Finn Jensen, Emily Korb, Richard Neapolitan, Kaye Stacey, Vicki Steinle, Charles Twardy, Chris Wallace and the CRC Press reviewer. Uffe Kjærulff (Hugin), Brent Boerlage (Netica) and Marek Druzdzel (GeNIe) helped us with their software packages, while Kevin Murphy assisted with the software package summary in Appendix B. Our research partners in various projects include: Nathalie Jitnah, Scott Thomson, Jason Carlton, Darren Boulton (Bayesian poker); Ryan McGowan, Daniel Willis, Ian Brown (ambulation monitoring); Kaye Stacey, Tali Boneh, Liz Sonenberg, Vicki Steinle, Tim Wilkin (intelligent tutoring); Tali Boneh, Liz Sonenberg (Matilda); Lucas Hope (VE); Ingrid Zukerman, Ricky McConachy (NAG); Chris Wallace, Julian Neil, Lucas Hope, Helen Armstrong, Charles Twardy, Rodney O'Donnell, Honghua Dai (causal discovery); Russell Kennett, Chris Ryan (seabreeze prediction). Various colleagues have been influential in our intellectual development leading us to this endeavor; we wish in particular to acknowledge: David Albrecht, Mike Brady, Tom Dean, Colin Howson, Finn Jensen, Leslie Kaelbling, Uffe Kjærulff, Jak Kirman, Noretta Koertge, Richard Neapolitan, Stuart Russell, Wesley Salmon, Neil Thomason, Ingrid Zukerman. We thank Alan Dorin for creating our cover image. Our dedication reflects our indebtedness to two of the great teachers, Judea Pearl and Chris Wallace. Finally, on a personal level Ann would like to thank her parents, Paul, and Robbie, and Kevin would like to thank Emily and Su.

© 2004 by Chapman & Hall/CRC Press LLC

About the Authors

Kevin B. Korb, Ph.D., earned his doctorate in the philosophy of science at Indiana University (1992) working on the philosophical foundations for the automation of Bayesian reasoning. Since then he has lectured at Monash University in Computer Science, combining his interests in philosophy of science and artificial intelligence in work on understanding and automating inductive inference, the use of MML in learning causal theories, artificial evolution of cognitive and social behavior and modeling Bayesian and human reasoning in the automation of argumentation.

Ann E. Nicholson, D.Phil., did her undergraduate computer science studies at the University of Melbourne and her doctorate in the robotics research group at Oxford University (1992), working on dynamic Bayesian networks for discrete monitoring. She then spent two years at Brown University as a post-doctoral research fellow before taking up a lecturing position at Monash University in Computer Science. Her general research focus is AI methods for reasoning under uncertainty, while her current research includes knowledge engineering with Bayesian networks, applications of Bayesian networks and user modeling.

© 2004 by Chapman & Hall/CRC Press LLC

Contents

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

List of Figures

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

© 2004 by Chapman & Hall/CRC Press LLC

List of Tables

© 2004 by Chapman & Hall/CRC Press LLC

Notation

A few remarks about notation before we begin. The notation special to Bayesian networks (or to our treatment of them) will be introduced as we proceed; first introductions of notation (and acronyms) will be recorded, with page numbers, in Appendix A. General mathematical notation is listed on the next page.

Here we describe the simplest aspects of the notation we adopt. First, variables (nodes in the network) will be named, with the names being capitalized and usually italicized (e.g., *Y*, *Alarm*, *Cancer*). Sets of variables will be set in boldface (e.g., $\mathbf{X}$, $\mathbf{Y}$). The values that variables take will not be capitalized, but will be italicized; thus, to assert that the alarm is on, we might write *Alarm* = *on*. Values abbreviated to single letters, such as *True (T)* and *False (F)*, however, will be capitalized. Where no confusion is likely to arise, variables and values may be abbreviated.

When first introducing new terminology we shall employ boldface to point it out; thus, for example, the first appearance (after this) of "Bayesianism" will be in boldface.

When the identity of the evidence or observation nodes is of interest in an example figure, they will be shown shaded in the network.

© 2004 by Chapman & Hall/CRC Press LLC

Basic Notation

$A \subset B$	A is a proper subset of B
$A \subseteq B$	A is a proper subset of B or equal to B
$A \backslash B$	set A with all elements of B removed
$\emptyset$	the empty set
$A \cup B$	the union of A and B
$A \cap B$	the intersection of A and B
$x \in A$	x is a member of set A
$\|A\|$	the number of objects in set A
$x \in [y, z]$	$y \leq x \leq z$
$\{A_i\}$	a set of sets indexed by i
$\bigcup_i A_i$	the union of all $A_i \in \{A_i\}$
$A \vee B$	A or B
$A \wedge B$	A and B
$A \equiv B$	A is equivalent to B
$\neg A$	not A
$\forall x$	for all x
$N!$	N factorial
X_i	A variable
$X_i = x_i$	The variable X_i takes value x_i
$\overline{X}$	$\sum_{i=1}^{n} \frac{x_i}{n}$
$P(X)$	(prior, marginal) probability of X
$P(X = x)$	probability that X takes value x
$P(X\|E)$	probability of X given evidence E
$Bel(X)$	posterior distribution over X
$Parents(X)$	set of parent nodes of X
$A - B$	an undirected link
$A \rightarrow B$	a directed link
$i \leftarrow i + 1$	assignment
$E[X]$	the expected value of X
$E[f(X)]$	the expected value of $f(X)$
$EU(A\|E)$	the expected utility of action A, given evidence E
$U(O\|A)$	the utility of outcome O, given action A
$\int_a^b f(x)dx$	the integral of $f(x)$ from a to b
$\sum_i f(x_i)$	the sum of $f(x_i)$ indexed by i
$\prod_i f(x_i)$	the product of $f(x_i)$ indexed by i
$C_j^n = \binom{n}{j}$	The number of ways of taking a subset of j objects from a set of size n

© 2004 by Chapman & Hall/CRC Press LLC

Part I

PROBABILISTIC REASONING

© 2004 by Chapman & Hall/CRC Press LLC

1

Bayesian Reasoning

1.1 Reasoning under uncertainty

Artificial intelligence (AI), should it ever exist, will be an intelligence developed by humans, implemented as an artifact. The level of intelligence demanded by Alan Turing's famous test [275] — the ability to fool ordinary (unfoolish) humans about whether the other end of a dialogue is being carried on by a human or by a computer — is some indication of what AI researchers are aiming for. Such an AI would surely transform our technology and economy. We would be able to automate a great deal of human drudgery and paperwork. Since computers are universal, programs can be effortlessly copied from one system to another (to the consternation of those worried about intellectual property rights!), and the labor savings of AI support for bureaucratic applications of rules, medical diagnosis, research assistance, manufacturing control, etc. promises to be enormous. If a serious AI is ever developed.

There is little doubt that an AI will need to be able to reason logically. An inability to discover, for example, that a system's conclusions have reached inconsistency is more likely to be debilitating than the discovery of an inconsistency itself. For a long time there has also been widespread recognition that practical AI systems shall have to cope with **uncertainty** — that is, they shall have to deal with incomplete evidence leading to beliefs that fall short of knowledge, with fallible conclusions and the need to recover from error, called **non-monotonic reasoning**. Nevertheless, the AI community has been slow to recognize that any serious, general-purpose AI will need to be able to reason probabilistically, what we call here **Bayesian reasoning**.

There are at least three distinct forms of uncertainty which an intelligent system operating in anything like our world shall need to cope with:

1. **Ignorance.** The limits of our knowledge lead us to be uncertain about many things. Does our poker opponent have a flush or is she bluffing?
2. **Physical randomness or indeterminism.** Even if we know everything that we might care to investigate about a coin and how we impart spin to it when we toss it, there will remain an inescapable degree of uncertainty about whether it will land heads or tails when we toss it. A die-hard determinist might claim otherwise, that some unimagined amount of detailed investigation might someday reveal which way the coin will fall; but such a view is for the forseeable future a mere act of scientistic faith. We are all practical indeterminists.

© 2004 by Chapman & Hall/CRC Press LLC

3. **Vagueness.** Many of the predicates we employ appear to be vague. It is often unclear whether to classify a dog as a spaniel or not, a human as brave or not, a thought as knowledge or opinion.

Bayesianism is the philosophy that asserts that in order to understand human opinion as it ought to be, constrained by ignorance and uncertainty, the probability calculus is the single most important tool for representing appropriate strengths of belief. In this text we shall present Bayesian computational tools for reasoning with and about strengths of belief as probabilities; we shall also present a Bayesian view of physical randomness. In particular we shall consider a probabilistic account of causality and its implications for an intelligent agent's reasoning about its physical environment. We will not address the third source of uncertainty above, vagueness, which is fundamentally a problem about semantics and one which has no good analysis so far as we are aware.

1.2 Uncertainty in AI

The successes of formal logic have been considerable over the past century and have been received by many as an indication that logic should be the primary vehicle for **knowledge representation** and reasoning within AI. **Logicism** in AI, as this has been called, dominated AI research in the 1960s and 1970s, only losing its grip in the 1980s when artificial neural networks came of age. Nevertheless, even during the heyday of logicism, any number of practical problems were encountered where logic would not suffice, because uncertain reasoning was a key feature of the problem. In the 1960s, medical diagnosis problems became one of the first attempted application areas of AI programming. But there is no symptom or prognosis in medicine which is strictly logically implied by the existence of any particular disease or syndrome; so the researchers involved quickly developed a set of "probabilistic" relations. Because probability calculations are hard — in fact, NP hard in the number of variables [52] (i.e., computationally intractable; see §1.11) — they resorted to implementing what has subsequently been called "naive Bayes" (or, "Idiot Bayes"), that is, probabilistic updating rules which assume that symptoms are independent of each other given diseases.

The independence constraints required for these systems were so extreme that the systems were received with no wide interest. On the other hand, a very popular set of expert systems in the 1970s and 1980s were based upon Buchanan and Shortliffe's MYCIN, or the uncertainty representation within MYCIN which they called **certainty factors** [34]. Certainty factors (CFs) were obtained by first eliciting from experts a "degree of increased belief" which some evidence e should imply for a hypothesis h, $MB(h, e) \in [0, 1]$, and also a corresponding "degree of increased

© 2004 by Chapman & Hall/CRC Press LLC

disbelief," $MD(h,e) \in [0,1]$. These were then combined:

$$CF(h,e) = MB(h,e) - MD(h,e) \in [-1,1]$$

This division of changes in "certainty" into changes in belief and disbelief reflects the curious notion that belief and disbelief are not necessarily related to one another (cf. [34, section 11.4]). A popular AI text, for example, sympathetically reports that "it is often the case that an expert might have confidence 0.7 (say) that some relationship is true and have no feeling about it being not true" [175, p.329]. The same point can be put more simply: experts are often inconsistent. Our goal in Bayesian modeling is, at least largely, to find the most accurate representation of a real system about which we may be receiving inconsistent expert advice, rather than finding ways of modeling the inconsistency itself.

Regardless of how we may react to this interpretation of certainty factors, no operational semantics for CFs were provided by Buchanan and Shortliffe. This meant that no real guidance could be given to experts whose opinions were being solicited. Most likely, they simply assumed that they were being asked for conditional probabilities of h given e and of $\neg h$ given e. And, indeed, there finally was a probabilistic semantics given for certainty factors: David Heckerman in 1986 [104] proved that a consistent probabilistic interpretation of certainty factors* would once again require strong independence assumptions: in particular that, when combining multiple pieces of evidence, the different pieces of evidence must always be independent of each other. Whereas this appears to be a desirable simplification of **rule-based** systems, allowing rules to be "modular," with the combined impact of diverse evidence being a compositional function of their separate impacts it is easy to demonstrate that the required independencies are frequently unavailable. The price of rule-based simplicity is irrelevance.

Bayesian networks provide a natural representation of probabilities which allow for (and take advantage of, as we shall see in Chapter 2) any independencies that may hold, while not being limited to problems satisfying strong independence requirements. The combination of substantial increases in computer power with the Bayesian network's ability to use any existing independencies to computational advantage make the approximations and restrictive assumptions of earlier uncertainty formalisms pointless. So we now turn to the main game: understanding and representing uncertainty with probabilities.

1.3 Probability calculus

The probability calculus allows us to represent the independencies which other systems require, but also allows us to represent any dependencies which we may need.

*In particular, a mapping of certainty factors into likelihood ratios.

© 2004 by Chapman & Hall/CRC Press LLC

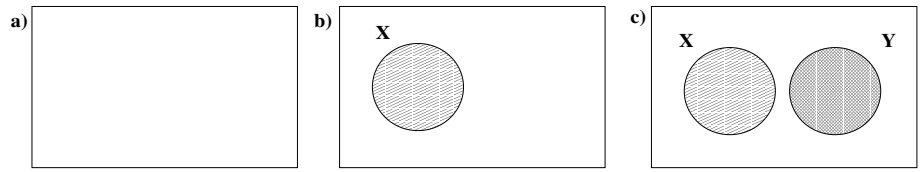


FIGURE 1.1
(a) The event space U; (b) $P(X)$; (c) $P(X \cup Y)$.

The probability calculus was specifically invented in the 17th century by Fermat and Pascal in order to deal with the problems of physical uncertainty introduced by gambling. But it did not take long before it was noticed that the concept of probability could be used in dealing also with the uncertainties introduced by ignorance, leading Bishop Butler to declare in the 18th century that "probability is the very guide to life." So now we introduce this formal language of probability, in a very simple way using Venn diagrams.

Let U be the universe of possible events; that is, if we are uncertain about which of a number of possibilities is true, we shall let U represent all of them collectively (see Figure 1.1(a)). Then the maximum probability must apply to the true event lying within U. By convention we set the maximum probability to 1, giving us Kolmogorov's first axiom for probability theory [153]:

Axiom 1.1 $P(U) = 1$

This probability mass, summing or integrating to 1, is distributed over U, perhaps evenly or perhaps unevenly. For simplicity we shall assume that it is spread evenly, so that the probability of any region is strictly proportional to its area. For any such region X its area cannot be negative, even if X is empty; hence we have the second axiom (Figure 1.1(b)):

Axiom 1.2 *For all* $X \subseteq U, P(X) \geq 0$

We need to be able to compute the probability of combined events, X and Y. This is trivial if the two events are mutually exclusive, giving us the third and last axiom (Figure 1.1(c)), known as **additivity**:

Axiom 1.3 *For all X,Y* $\subseteq U$, if $X \cap Y = \emptyset$, then $P(X \cup Y) = P(X) + P(Y)$

Any function over a field of subsets of U satisfying the above axioms will be a probability function[†].

A simple theorem extends addition to events which overlap (i.e., sets which intersect):

[†] A set-theoretic field is a set of sets containing U and $\emptyset$ and is closed under union, intersection and complementation.

© 2004 by Chapman & Hall/CRC Press LLC

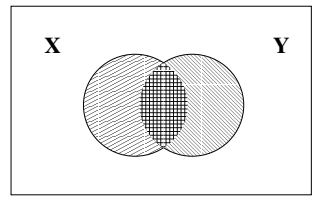


FIGURE 1.2
Conditional probability: $P(X|Y) = P(X \cap Y)/P(Y)$.

Theorem 1.1 *For all X,Y* $\subseteq U, P(X \cup Y) = P(X) + P(Y) - P(X \cap Y)$.

This can be intuitively grasped from Figure 1.2: the area of $X \cup Y$ is less than area of X plus the area of Y because when adding the area of intersection $X \cap Y$ has been counted twice; hence, we simply remove the excess to find $P(X \cup Y)$ for any two events X and Y.

The concept of **conditional probability** is crucial for the useful application of the probability calculus. It is usually introduced by definition:

Definition 1.1 Conditional probability

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)}$$

That is, given that the event Y has occurred, or will occur, the probability that X will also occur is $P(X|Y)$. Clearly, if Y is an event with zero probability, then this conditional probability is undefined. This is not an issue for probability distributions which are **positive**, since, by definition, they are non-zero over every event. A simple way to think about probabilities conditional upon Y is to imagine that the universe of events U has shrunk to Y. The conditional probability of X on Y is just the measure of what is left of X relative to what is left of Y; in Figure 1.2 this is just the ratio of the darker area (representing $X \cap Y$) to the area of Y. This way of understanding conditional probability is justified by the fact that the conditional function $P(\cdot|Y)$ is itself a probability function[‡] — that is, it provably satisfies the three axioms of probability.

Another final probability concept we need to introduce is that of **independence** (or, marginal independence). Two events X and Y are probabilistically independent (in notation, $X \perp\!\!\!\perp Y$) whenever conditioning on one leaves the probability of the other unchanged:

Definition 1.2 Independence $X \perp\!\!\!\perp Y \equiv P(X|Y) = P(X)$

This is provably symmetrical: $X \perp\!\!\!\perp Y \equiv Y \perp\!\!\!\perp X$. The simplest examples of independence come from gambling. For example, two rolls of dice are normally independent.

[‡] $P(\cdot|Y)$ is just the function equal to $P(X|Y)$ for all $X \subset U$.

© 2004 by Chapman & Hall/CRC Press LLC

Getting a one with the first roll will neither raise nor lower the probability of getting a one the second time. If two events are **dependent**, then one coming true will *alter* the probability of the other. Thus, the probability of getting a diamond flush in poker (five diamonds in five cards drawn) is *not* simply $(1/4)^5 = 1/1024$: the probability that the first card drawn being a diamond is $1/4$, but the probability of subsequent cards being diamonds is influenced by the fact that there are then fewer diamonds left in the deck.

Conditional independence generalizes this concept to X and Y being independent given some additional event Z:

Definition 1.3 Conditional independence $X \perp\!\!\!\perp Y | Z \equiv P(X|Y,Z) = P(X|Z)$

This is a true generalization because, of course, Z can be the empty set $\emptyset$, when it reduces to marginal independence. Conditional independence holds when the event Z tells us everything that Y does about X and possibly more; once you know Z, learning Y is uninformative. For example, suppose we have two diagnostic tests for cancer X, an inexpensive but less accurate one, Y, and a more expensive and more accurate one, Z. If Z is more accurate partly because it effectively incorporates all of the diagnostic information available from Y, then knowing the outcome of Z will render an additional test of Y irrelevant — Y will be "screened off" from X by Z.

1.3.1 Conditional probability theorems

We introduce without proof two theorems on conditional probability which will be of frequent use:

Theorem 1.2 Total Probability *Assume the set of events $\{A_i\}$ is a partition of U; i.e., $\bigcup_i A_i = U$ and for any distinct i and j $A_i \cap A_j = \emptyset$. Then*

$$P(U) = \sum_i P(A_i)$$

We can equally well partition the probability of any particular event B instead of the whole event space. In other words, under the above conditions (and if $\forall i A_i \neq \emptyset$),

$$P(B) = \sum_i P(B|A_i)$$

We shall refer to either formulation under the title "Total Probability."

Theorem 1.3 The Chain Rule *Given three events A, B, C,*

$$P(C|A) = P(C|B)P(B|A) + P(C|\neg B)P(\neg B|A)$$

assuming the conditional probabilities are defined. This allows us to divide the probabilistic influence of C on A across the different states of a third variable. (Here, the third variable is binary, but the theorem is easily generalized to variables of arbitrary arity.)

© 2004 by Chapman & Hall/CRC Press LLC

1.3.2 Variables

Although we have introduced probabilities over events, in most of our discussion we shall be concerned with probabilities over **random variables**. A random variable is a variable which reports the outcome of some measurement process. It can be related to events, to be sure. For example, instead of talking about which event in a partition $\{A_i\}$ turns out to be the case, we can equivalently talk about which state x_i the random variable X takes, which we write $X = x_i$. The set of states a variable X can take form its state space, written Ω_X, and its size (or arity) is $|\Omega_X|$.

The discussion thus far has been implicitly of discrete variables, those with a finite state space. However, we need also to introduce the concept of probability distributions over **continuous variables**, that is, variables which range over real numbers, like *Temperature*. For the most part in this text we shall be using probability distributions over discrete variables (events), for two reasons. First, the Bayesian network technology is primarily oriented towards handling discrete state variables, for example the inference algorithms of Chapter 3. Second, for most purposes continuous variables can be **discretized**. For example, temperatures can be divided into ranges of ± 5 degrees for many purposes; and if that is too crude, then they can be divided into ranges of ± 1 degree, etc.

Despite our ability to evade probabilities over continuous variables much of the time, we shall occasionally need to discuss them. We introduce these probabilities by first starting with a **density function** $f(X)$ defined over the continuous variable X. Intuitively, the density assigns a weight or measure to each possible value of X and can be approximated by a finely partitioned histogram reporting samples from X. Although the density is not itself a probability function, it can be used to generate one so long as $f(\cdot)$ satisfies the conditions:

$$f(x) \geq 0 \tag{1.1}$$

$$\int_{-\infty}^{+\infty} f(x)dx = 1 \tag{1.2}$$

In words: each point value is positive or zero and all values integrate to 1. In that case we can define the **cumulative probability distribution** $F(\cdot)$ by

$$F(x) = P(X \leq x) = \int_{x' \leq x} f(x')dx' \tag{1.3}$$

This function assigns probabilities to ranges from each possible value of x down to negative infinity. Note that we can analogously define probabilites over any continuous interval of values for X, so long as the interval is not degenerate (equal to a point). In effect, we obtain a probability distribution by discretizing the continuous variable — i.e., by looking at the mass of the density function over intervals.

© 2004 by Chapman & Hall/CRC Press LLC

1.4 Interpretations of probability

There have been two main contending views about how to understand probability. One asserts that probabilities are fundamentally dispositional properties of non-deterministic physical systems, the classical such systems being gambling devices, such as dice. This view is particularly associated with **frequentism**, advocated in the 19th century by John Venn [284], identifying probabilities with long-run frequencies of events. The obvious complaint that short-run frequencies clearly do not match probabilities (e.g., if we toss a coin only once, we would hardly conclude that it's probability of heads is either one or zero) does not actually get anywhere, since no claim is made identifying short-run frequencies with probabilities. A different complaint does bite, however, namely that the distinction between short-run and long-run is vague, leaving the commitments of this frequentist interpretation unclear. Richard von Mises in the early 20th century [286] fixed this problem by formalizing the frequency interpretation, identifying probabilities with frequency limits in infinite sequences satisfying certain assumptions about randomness. Some version of this frequency interpretation is commonly endorsed by statisticians.

A more satisfactory theoretical account of physical probability arises from Karl Popper's observation [220] that the frequency interpretation, precise though it was, fails to accommodate our intuition that probabilities of singular events exist and are meaningful. If, in fact, we do toss a coin once and once only, and if this toss should *not* participate in some infinitude (or even large number) of appropriately similar tosses, it would not for that reason fail to have some probability of landing heads. Popper identified physical probabilities with the **propensities** (dispositions) of physical systems ("chance setups") to produce particular outcomes, whether or not those dispositions were manifested repeatedly. An alternative that amounts to much the same thing is to identify probabilities with counterfactual frequencies generated by hypothetically infinite repetitions of an experiment [282].

Whether physical probability is relativized to infinite random sequences, infinite counterfactual sequences or chance setups, these accounts all have in common that the assertion of a probability is relativized to *some* definite physical process or the outcomes it generates.

The traditional alternative to the concept of physical probability is to think of probabilities as reporting our subjective degrees of belief. This view was expressed by Thomas Bayes [16] (Figure 1.3) and Pierre Simon de Laplace [68] two hundred years ago. This is a more general account of probability in that we have subjective belief in a huge variety of propositions, many of which are not at all clearly tied to a physical process capable even in principle of generating an infinite sequence of outcomes. For example, most of us have a pretty strong belief in the Copernican hypothesis that the earth orbits the sun, but this is based on evidence not obviously the same as the outcome of a sampling process. We are not in any position to generate solar systems repeatedly and observe the frequency with which their planets revolve around the sun, for example. Bayesians nevertheless are prepared to talk about the probability

© 2004 by Chapman & Hall/CRC Press LLC

of the truth of the Copernican thesis and can give an account of the relation between that probability and the evidence for and against it. Since these probabilities are typically subjective, not clearly tied to physical models, most frequentists (hence, most statisticians) deny their meaningfulness. It is not insignificant that this leaves their (usual) belief in Copernicanism unexplained.

The first thing to make clear about this dispute between physicalists and Bayesians is that Bayesianism can be viewed as *generalizing* physicalist accounts of probability. That is, it is perfectly compatible with the Bayesian view of probability as measuring degrees of subjective belief to adopt what David Lewis dubbed the **Principal Principle** [171]: whenever you learn that the physical probability of an outcome is r, set your subjective probability for that outcome to r. This is really just common sense: you may think that the probability of a friend shaving his head is 0.01, but if you learn that he will do so if and only if a fair coin yet to be flipped lands heads, you'll revise your opinion accordingly.

So, the Bayesian and physical interpretations of probability are compatible, with the Bayesian interpretation *extending* the application of probability beyond what is directly justifiable in physical terms. That is the view we adopt here. But what justifies this extension?

FIGURE 1.3
Reverend Thomas Bayes (1702–1761).

© 2004 by Chapman & Hall/CRC Press LLC

1.5 Bayesian philosophy

1.5.1 Bayes' theorem

The origin of Bayesian philosophy lies in an interpretation of **Bayes' Theorem:**

Theorem 1.4 Bayes' Theorem

$$P(h|e) = \frac{P(e|h)P(h)}{P(e)}$$

This is a non-controversial (and simple) theorem of the probability calculus. Under its usual Bayesian interpretation, it asserts that the probability of a hypothesis h conditioned upon some evidence e is equal to its **likelihood** $P(e|h)$ times its probability prior to any evidence $P(h)$, **normalized** by dividing by $P(e)$ (so that the conditional probabilities of all hypotheses sum to 1). So much is not controvertible.

The further claim that this is a right and proper way of adjusting our beliefs in our hypotheses given new evidence is called **conditionalization**, and it is controversial.

Definition 1.4 Conditionalization *After applying Bayes' theorem to obtain $P(h|e)$ adopt that as your posterior degree of belief in h — or, $Bel(h) = P(h|e)$.*

Conditionalization, in other words, advocates belief updating via probabilities conditional upon the available evidence. It identifies **posterior probability** (the probability function after incorporating the evidence, which we are writing $Bel(\cdot)$) with **conditional probability** (the prior probability function conditional upon the evidence, which is $P(\cdot|e)$). Put thus, conditionalization may also seem non-controvertible. But there are certainly situations where conditionalization very clearly does not work. The two most basic such situations simply violate what are frequently explicitly stated as assumptions of conditionalization: (1) There must exist joint priors over the hypothesis and evidence spaces. Without a joint prior, Bayes' theorem cannot be used, so conditionalization is a non-starter. (2) The evidence conditioned upon, e, is all and only the evidence learned. This is called the **total evidence condition**. It is a significant restriction, since in many settings it cannot be guaranteed.

The first assumption is also significant. Many take it as the single biggest objection to Bayesianism to raise the question "Where do the numbers come from?" For example, the famous anti-Bayesian Clark Glymour [94] doesn't complain about Bayesian reasoning involving gambling devices, when the outcomes are engineered to start out equiprobable, but doubts that numbers can be found for more interesting cases. To this kind of objection Bayesians react in a variety of ways. In fact, the different varieties of response pretty much identify the different schools of Bayesianism. Objectivists, such as Rudolf Carnap [39] and Ed Jaynes [122], attempt to define prior probabilities based upon the structure of language. Extreme subjectivists, such as de Finetti [67], assert that it makes no difference what source your priors have: given

© 2004 by Chapman & Hall/CRC Press LLC

that de Finetti's representation theorem shows that non-extreme priors converge in the limit (under reasonable constraints), it just doesn't matter what priors you adopt.

The practical application of Bayesian reasoning does not appear to depend upon settling this kind of philosophical problem. A great deal of useful application can be done simply by refusing to adopt a dogmatic position and accepting common-sense prior probabilities. For example, if there are ten possible suspects in a murder mystery, a fair starting point for any one of them is a 1 in 10 chance of guilt; or, again, if burglaries occur in your neighborhood of 10,000 homes about once a day, then the probability of your having been burglarized within the last 24 hours might reasonably be given a prior probability of 1/10000.

Colin Howson points out that conditionalization is a valid rule of inference if and only if $Bel(e|h) = P(e|h)$, that is, if and only if your prior and posterior probability functions share the relevant conditional probabilities (cf. [116]). This is certainly a pertinent observation, since encountering some possible evidence may well inform us more about defects in our own conditional probability structure than about the hypothesis at issue. Since Bayes' theorem has $P(h|e)$ being proportional to $P(e|h)$, if the evidence leads us to revise $P(e|h)$, we will be in no position to conditionalize.

How to generate prior probabilities or new conditional probability structure is not dictated by Bayesian principles. Bayesian principles advise how to update probabilities once such a conditional probability structure has been adopted, given appropriate priors. Expecting Bayesian principles to answer all questions about reasoning is expecting too much. Nevertheless, we shall show that Bayesian principles implemented in computer programs can deliver a great deal more than the nay-sayers have ever delivered.

Definition 1.5 Jeffrey conditionalization *Suppose your observational evidence does not correspond specifically to proposition e, but can be represented as a posterior shift in belief about e. In other words, posterior belief in e is not full but partial, having shifted from $P(e)$ to $Bel(e)$. Then, instead of Bayesian conditionalization, apply Jeffrey's update rule [123] for* **probability kinematics***: $Bel(h) = P(h|e)Bel(e) + P(h|\neg e)Bel(\neg e)$.*

Jeffrey's own example is one where your hypothesis is about the color of a cloth, the evidence proposition e describes the precise quality of your visual experience under good light, but you are afforded a view of the cloth only under candlelight, in such a way that you cannot exactly articulate what you have observed. Nevertheless, you have learned *something*, and this is reflected in a shift in belief about the quality of your visual experience. Jeffrey conditionalization is very intuitive, but again is not strictly valid. As a practical matter, the need for such partial updating is common in Bayesian modeling.

1.5.2 Betting and odds

Odds are the ratio between the cost of a bet in favor of a proposition and the reward should the bet be won. Thus, assuming a stake of \$1 (and otherwise simply rescaling

© 2004 by Chapman & Hall/CRC Press LLC

the terms of the bet), a bet at 1:19 odds costs \$1 and returns \$20 should the proposition come true (with the reward being \$20 minus the cost of the bet)[§]. The odds may be set at any ratio and may, or may not, have something to do with one's probabilities. Bookies typically set odds for and against events at a slight discrepancy with their best estimate of the probabilities, for their profit lies in the difference between the odds for and against.

While odds and probabilities may deviate, probabilities and **fair odds** $O(\cdot)$ are strictly interchangeable concepts. The fair odds in favor of h are defined simply as the ratio of the probability that h is true to the probability that it is not:

Definition 1.6 Fair odds

$$O(h) = \frac{P(h)}{1 - P(h)}$$

Given this, it is an elementary matter of algebraic manipulation to find $P(h)$ in terms of odds:

$$P(h) = \frac{O(h)}{1 + O(h)} \tag{1.4}$$

Thus, if a coin is fair, the probability of heads is 1/2, so the odds in favor of heads are 1:1 (usually described as "50:50"). Or, if the odds of getting "snake eyes" (two 1's) on the roll of two dice are 1:35, then the probability of this is:

$$\frac{1/35}{1 + 1/35} = \frac{1/35}{36/35} = 1/36$$

as will always be the case with fair dice. Or, finally, suppose that the probability an agent ascribes to the Copernican hypothesis (CH) is zero; then the odds that agent is giving to Copernicus having been wrong ($\neg CH$) are *infinite*:

$$O(\neg CH) = \frac{1}{0} = \infty$$

At these odds, incidentally, it is trivial that the agent can never reach a degree of belief in CH above zero on any finite amount of evidence, if relying upon conditionalization for updating belief.

With the concept of fair odds in hand, we can reformulate Bayes' theorem in terms of (fair) odds, which is often useful:

Theorem 1.5 Odds-Likelihood Bayes' Theorem

$$O(h|e) = \frac{P(e|h)}{P(e|\neg h)} O(h)$$

This is readily proven to be equivalent to Theorem 1.4. In English it asserts that the odds on h conditional upon the evidence e are equal to the prior odds on h times the **likelihood ratio** $P(e|h) : P(e|\neg h)$. Clearly, the fair odds in favor of h will rise if and only if the likelihood ratio is greater than one.

[§] It is common in sports betting to invert the odds, quoting the odds *against* a team winning, for example. This makes no difference; the ratio is simply reversed.

© 2004 by Chapman & Hall/CRC Press LLC

1.5.3 Expected utility

Generally, agents are able to assign utility (or, value) to the situations in which they find themselves. We know what we like, we know what we dislike, and we also know when we are experiencing neither of these. Given a general ability to order situations, and bets with definite probabilities of yielding particular situations, Frank Ramsey [231] demonstrated that we can identify particular utilities with each possible situation, yielding a **utility function**.

If we have a utility function $U(O_i|A)$ over every possible outcome of a particular action A we are contemplating, and if we have a probability for each such outcome $P(O_i|A)$, then we can compute the probability-weighted average utility for that action — otherwise known as the **expected utility** of the action:

Definition 1.7 Expected utility

$$EU(A) = \sum_i U(O_i|A) \times P(O_i|A)$$

It is commonly taken as axiomatic by Bayesians that agents ought to *maximize their expected utility*. That is, when contemplating a number of alternative actions, agents ought to decide to take that action which has the maximum expected utility. If you are contemplating eating strawberry ice cream or else eating chocolate ice cream, presumably you will choose that flavor which you prefer, other things being equal. Indeed, if you chose the flavor you liked *less*, we should be inclined to think that other things are *not* equal — for example, you are under some kind of external compulsion — or perhaps that you are not being honest about your preferences. Utilities have behavioral consequences *essentially*: any agent who consistently ignores the putative utility of an action or situation arguably does not have that utility.

Regardless of such foundational issues, we now have the conceptual tools necessary to understand what is fair about fair betting. **Fair bets** are fair because their expected utility is zero. Suppose we are contemplating taking the fair bet B on proposition h for which we assign probability $P(h)$. Then the expected utility of the bet is:

$$EU(B) = U(h|B)P(h|B) + U(\neg h|B)P(\neg h|B)$$

Typically, betting on a proposition has no effect on the probability that it is true (although this is not necessarily the case!), so $P(h|B) = P(h)$. Hence,

$$EU(B) = U(h|B)P(h) + U(\neg h|B)(1 - P(h))$$

Assuming a stake of 1 unit for simplicity, then by definition $U(h|B) = 1 - P(h)$ (i.e., this is the utility of h being true given the bet for h) while $U(\neg h|B) = -P(h)$, so,

$$EU(B) = (1 - P(h))P(h) - P(h)(1 - P(h)) = 0$$

Given that the bet has zero expected utility, the agent should be no more inclined to take the bet in favor of h than to take the opposite bet against h.

© 2004 by Chapman & Hall/CRC Press LLC

1.5.4 Dutch books

The original Dutch book argument of Ramsey [231] (see also [67]) claims to show that subjective degrees of belief, if they are to be rational, *must* obey the probability calculus. It has the form of a *reductio ad absurdum* argument:

1. A rational agent should be willing to take either side of any combination of fair bets.
2. A rational agent should never be willing to take a combination of bets which guarantees a loss.
3. Suppose a rational agent's degrees of belief violate one or more of the axioms of probability.
4. Then it is provable that some combination of fair bets will lead to a guaranteed loss.
5. Therefore, the agent is both willing and not willing to take this combination of bets.

Now, the inferences to (4) and (5) in this argument are not in dispute (see §1.11 for a simple demonstration of (4) for one case). A *reductio* argument needs to be resolved by finding a prior assumption to blame, and concluding that it is false. Ramsey, and most Bayesians to date, supposed that the most plausible way of relieving the contradiction of (5) is by refusing to suppose that a rational agent's degrees of belief may violate the axioms of probability. This result can then be generalized beyond settings of explicit betting by taking "bets with nature" as a metaphor for decision-making generally. For example, walking across the street is in some sense a bet about our chances of reaching the other side.

Some anti-Bayesians have preferred to deny (1), insisting for example that it would be uneconomic to invest in bets with zero expected value (e.g., [48]). But the ascription of the radical incoherence in (5) simply to the willingness of, say, bored aristocrats to place bets that will net them nothing clearly will not do: the effect of incoherence is entirely out of proportion with the proposed cause of effeteness.

Alan Hájek [99] has recently pointed out a more plausible objection to (2). In the scenarios presented in Dutch books there is always some combination of bets which guarantees a net loss whatever the outcomes on the individual bets. But equally there is always some combination of bets which guarantees a net gain — a "Good Book." So, one agent's half-empty glass is another's half-full glass! Rather than dismiss the Dutch-bookable agent as irrational, we might commend it for being open to a guaranteed win! So, Hájek's point seems to be that there is a fundamental symmetry in Dutch book arguments which leaves open the question whether violating probability axioms is rational or not. Certainly, when metaphorically extending betting to a "struggle" with Nature, it becomes rather implausible that She is really out to Dutch book us!

Hájek's own solution to the problem posed by his argument is to point out that whenever an agent violates the probability axioms there will be some variation of its system of beliefs which is guaranteed to win money whenever the original system is

© 2004 by Chapman & Hall/CRC Press LLC

guaranteed to win, and which is also capable of winning in some situations when the original system is not. So the variant system of belief in some sense dominates the original: it is everywhere at least as good as the original and in some places better. In order to guarantee that your system of beliefs cannot be dominated, you must be probabilistically coherent (see §1.11). This, we believe, successfully rehabilitates the Dutch book in a new form.

Rather than rehabilitate, a more obviously Bayesian response is to consider the probability of a bookie hanging around who has the smarts to pump our agent of its money and, again, of a simpleton hanging around who will sign up the agent for guaranteed winnings. In other words, for rational choice surely what matters is the relative expected utility of the choice. Suppose, for example, that we are offered a set of bets which has a guaranteed loss of $10. Should we take it? The Dutch book assumes that accepting the bet is irrational. But, if the one and only alternative available is another bet with an expected loss of $1,000, then it no longer seems so irrational. An implicit assumption of the Dutch book has always been that betting is voluntary and when all offered bets are turned down the expected utility is zero. The further implicit assumption pointed out by Hájek's argument is that there is always a shifty bookie hanging around ready to take advantage of us. No doubt that is not always the case, and instead there is only some probability of it. Yet referring the whole matter of justifying the use of Bayesian probability to expected utility smacks of circularity, since expectation is understood in terms of Bayesian probability.

Aside from invoking the rehabilitated Dutch book, there is a more pragmatic approach to justifying Bayesianism, by looking at its importance for dealing with cases of practical problem solving. We take Bayesian principles to be normative, and especially to be a proper guide, under some range of circumstances, to evaluating hypotheses in the light of evidence. The form of justification that we think is ultimately most compelling is the "method of reflective equilibrium," generally attributed to Goodman [96] and Rawls [232], but first set out by Aristotle [10]. In a nutshell, it asserts that the normative principles to accept are those which best accommodate our basic, unshakable intuitions about what is good and bad (e.g., paradigmatic judgments of correct inference in simple domains, such as gambling) and which best integrate with relevant theory and practice. We now present some cases which Bayesian principle handles readily, and better than any alternative normative theory.

1.5.5 Bayesian reasoning examples

1.5.5.1 Breast Cancer

Suppose the women attending a particular clinic show a long-term chance of 1 in 100 of having breast cancer. Suppose also that the initial screening test used at the clinic has a false positive rate of 0.2 (that is, 20% of women without cancer will test positive for cancer) and that it has a false negative rate of 0.1 (that is, 10% of women with cancer will test negative). The laws of probability dictate from this last fact that the probability of a positive test given cancer is 90%. Now suppose that you are such a woman who has just tested positive. What is the probability that you have cancer?

© 2004 by Chapman & Hall/CRC Press LLC

This problem is one of a class of probability problems which has become notorious in the cognitive psychology literature (cf. [277]). It seems that very few people confronted with such problems bother to pull out pen and paper and compute the right answer via Bayes' theorem; even fewer can get the right answer without pen and paper. It appears that for many the probability of a positive test (which is observed) given cancer (i.e., 90%) dominates things, so they figure that they have quite a high chance of having cancer. But substituting into Theorem 1.4 gives us:

$$P(Cancer|Pos) = \frac{P(Pos|Cancer)P(Cancer)}{P(Pos)}$$

Note that the probability of *Pos* given *Cancer* — which is the likelihood 0.9 — is only *one* term on the right hand side; the other crucial term is the prior probability of cancer. Cognitive psychologists studying such reasoning have dubbed the dominance of likelihoods in such scenarios "base-rate neglect," since the base rate (prior probability) is being suppressed [137]. Filling in the formula and computing the conditional probability of *Cancer* given *Pos* gives us quite a different story:

$$\begin{aligned} P(Cancer|Pos) &= \frac{P(Pos|Cancer)P(Cancer)}{P(Pos)} \\ &= \frac{P(Pos|Cancer)P(Cancer)}{P(Pos|Cancer)P(Cancer) + P(Pos|\neg Cancer)P(\neg Cancer)} \\ &= \frac{0.9 \times 0.01}{0.9 \times 0.01 + 0.2 \times 0.99} \\ &= \frac{0.009}{0.009 + 0.198} \\ &\approx 0.043 \end{aligned}$$

Now the discrepancy between 4% and 80 or 90% is no small matter, particularly if the consequence of an error involves either unnecessary surgery or (in the reverse case) leaving a cancer untreated. But decisions similar to these are constantly being made based upon "intuitive feel" — i.e., without the benefit of paper and pen, let alone Bayesian networks (which are simpler to use than paper and pen!).

1.5.5.2 People v. Collins

The legal system is replete with misapplications of probability and with incorrect claims of the irrelevance of probabilistic reasoning as well.

In 1964 an interracial couple was convicted of robbery in Los Angeles, largely on the grounds that they matched a highly improbable profile, a profile which fit witness reports [272]. In particular, the two robbers were reported to be

- A man with a mustache
- Who was black and had a beard
- And a woman with a ponytail
- Who was blonde

© 2004 by Chapman & Hall/CRC Press LLC

- The couple was interracial
- And were driving a yellow car

The prosecution suggested that these characteristics had the following probabilities of being observed at random in the LA area:

1. A man with a mustache 1/4
2. Who was black and had a beard 1/10
3. And a woman with a ponytail 1/10
4. Who was blonde 1/3
5. The couple was interracial 1/1000
6. And were driving a yellow car 1/10

The prosecution called an instructor of mathematics from a state university who apparently testified that the "product rule" applies to this case: where mutually independent events are being considered jointly, the joint probability is the product of the individual probabilities[¶]. This last claim is, in fact, correct (see Problem 2 below); what is false is the idea that the product rule is relevant to this case. If we label the individual items of evidence e_i $(i = 1, \ldots, 6)$, the joint evidence e, and the hypothesis that the couple was guilty h, then what is claimed is

$$P(e|\neg h) = \prod_i P(e_i|\neg h) = 1/12000000$$

The prosecution, having made this inference, went on to assert that the probability the couple were innocent was no more than 1/12000000. The jury convicted.

As we have already suggested, the product rule does *not* apply in this case. Why not? Well, because the individual pieces of evidence are obviously *not* independent. If, for example, we know of the occupants of a car that one is black and the other has blonde hair, what then is the probability that the occupants are an interracial couple? Clearly not 1/1000! If we know of a man that he has a mustache, is the probability of having a beard unchanged? These claims are preposterous, and it is simply shameful that a judge, prosecutor and defence attorney could not recognize how preposterous they are — let alone the mathematics "expert" who testified to them. Since e_2 implies e_1, while e_2, e_3, e_4 jointly imply e_5 (to a fair approximation), a far better estimate for $P(e|\neg h)$ is $P(e_2|\neg h)P(e_3|\neg h)P(e_4|\neg h)P(e_6|\neg h) = 1/3000$.

To be sure, if we accepted that the probability of innocence were a mere 1/3000 we might well accept the verdict. But there is a more fundamental error in the prosecution reasoning than neglecting the conditional dependencies in the evidence. If, unlike the judge, prosecution and jury, we take a peek at Bayes' theorem, we discover that the probability of guilt $P(h|e)$ is *not* equal to $1 - P(e|\neg h)$; instead

$$P(h|e) = \frac{P(e|h)P(h)}{P(e|h)P(h) + P(e|\neg h)P(\neg h)}$$

[¶] Coincidentally, this is just the kind of independence required for certainty factors to apply.

© 2004 by Chapman & Hall/CRC Press LLC

Now if the couple in question *were* guilty, what are the chances the evidence accumulated would have been observed? That's a rather hard question to answer, but feeling generous towards the prosecution, let us simplify and say 1. That is, let us accept that $P(e|h) = 1$. Plugging in our assumptions we have thus far:

$$P(h|e) = \frac{P(h)}{P(h) + P(\neg h)/3000}$$

We are missing the crucial prior probability of a random couple being guilty of the robbery. Note that we cannot here use the prior probability of, for example, an interracial couple being guilty, since the fact that they are interracial is a piece of the evidence. The most plausible approach to generating a prior of the needed type is to count the number of couples in the LA area and give them an equal prior probability. In other words, if N is the number of possible couples in the LA area, $P(h) = 1/N$. So, what is N? The population at the time was about 6.5 million people [73]. If we conservatively take half of them as being eligible to be counted (e.g., being adult humans), this gives us 1,625,000 eligible males and as many females. If we simplify by supposing that they are all in heterosexual partnerships, that will introduce a slight bias in favor of innocence; if we also simplify by ignoring the possibility of people traveling in cars with friends, this will introduce a larger bias in favor of guilt. The two together give us 1,625,000 available couples, suggesting a prior probability of guilt of 1/1625000. Plugging this in we get:

$$P(h|e) = \frac{1/1625000}{1/1625000 + (1 - 1/1625000)/3000} \approx 0.002$$

In other words, even ignoring the huge number of trips with friends rather than partners, we obtain a 99.8% chance of innocence and so a very large probability of a nasty error in judgment. The good news is that the conviction (of the man only!) was subsequently overturned, partly on the basis that the independence assumptions are false. The bad news is that the appellate court finding also suggested that probabilistic reasoning is just irrelevant to the task of establishing guilt, which is a nonsense. One right conclusion about this case is that, assuming the likelihood has been *properly* worked out, a sensible prior probability must also be taken into account. In some cases judges have specifically ruled out all consideration of prior probabilities, while allowing testimony about likelihoods! Probabilistic reasoning which simply ignores half of Bayes' theorem is dangerous indeed!

Note that we do not claim that 99.8% is the best probability of innocence that can be arrived at for the case of People v. Collins. What we *do* claim is that, for the particular facts represented as having a particular probabilistic interpretation, this is far closer to a reasonable probability than that offered by the prosecution, namely 1/12000000. We also claim that the forms of reasoning we have here illustrated are *crucial* for interpreting evidence in general: namely, whether the offered items of evidence are conditionally independent and what the prior probability of guilt may be.

© 2004 by Chapman & Hall/CRC Press LLC

1.6 The goal of Bayesian AI

The most commonly stated goal for artificial intelligence is that of producing an artifact which performs difficult intellectual tasks at or beyond a human level of performance. Of course, machine chess programs have satisfied this criterion for some time now. Although some AI researchers have claimed that therefore an AI has been produced — that denying this is an unfair shifting of the goal line — it is absurd to think that we ought to be satisfied with programs which are strictly special-purpose and which achieve their performance using techniques that deliver nothing when applied to most areas of human intellectual endeavor.

Turing's test for intelligence appears to be closer to satisfactory: fooling ordinary humans with verbal behavior not restricted to any domain would surely demonstrate some important *general* reasoning ability. Many have pointed out that the conditions for Turing's test, strictly verbal behavior without any afferent or efferent nervous activity, yield at best some kind of disembodied, ungrounded intelligence. John Searle's Chinese Room argument, for example, can be interpreted as making such a case ([246]; for this kind of interpretation of Searle see [100] and [156]). A more convincing criterion for human-like intelligence is to require of an artificial intelligence that it be capable of powering a robot-in-the-world in such a way that the robot's performance cannot be distinguished from human performance in terms of behavior (disregarding, for example, whether the skin can be so distinguished). The program that can achieve this would surely satisfy any sensible AI researcher, or critic, that an AI had been achieved.

We are not, however, actually motivated by the idea of behaviorally cloning humans. If all we wish to do is reproduce humans, we would be better advised to employ the tried and true methods we have always had available. Our motive is to understand *how* such performance can be achieved. We are interested in knowing how humans perform the many interesting and difficult cognitive tasks encompassed by AI — such as, natural language understanding and generation, planning, learning, decision making — but we are also interested in knowing how they might be performed otherwise, and in knowing how they might be performed optimally. By building artifacts which model our best understanding of how humans do these things (which can be called **descriptive artificial intelligence**) and also building artifacts which model our best understanding of what is optimal in these activities (**normative artificial intelligence**), we can further our understanding of the nature of intelligence and also produce some very useful tools for science, government and industry.

As we have indicated through example, medical, legal, scientific, political and most other varieties of human reasoning either consider the relevant probabilistic factors and accommodate them or run the risk of introducing egregious and damaging errors. The goal of a Bayesian artificial intelligence is to produce a thinking agent which does as well or better than humans in such tasks, which can adapt to stochastic and changing environments, recognize its own limited knowledge and cope sensibly with these varied sources of uncertainty.

© 2004 by Chapman & Hall/CRC Press LLC

1.7 Achieving Bayesian AI

Given that we have this goal, how can we achieve it? The first step is to develop algorithms for doing Bayesian conditionalization properly and, insofar as possible, efficiently. This step has already been achieved, and the relevant algorithms are described in Chapters 2 and 3. The next step is to incorporate methods for computing expected utilities and develop methods for maximizing utility in decision making. We describe algorithms for this in Chapter 4. We would like to test these ideas in application: we describe some Bayesian network applications in Chapter 5.

These methods for probability computation are fairly well developed and their improvement remains an active area of research in AI today. The biggest obstacles to Bayesian AI having a broad and deep impact outside of the research community are the difficulties in developing applications, difficulties with eliciting knowledge from experts, and integrating and validating the results. One issue is that there is no clear methodology for developing, testing and deploying Bayesian network technology in industry and government — there is no recognized discipline of "software engineering" for Bayesian networks. We make a preliminary effort at describing one — Knowledge Engineering with Bayesian Networks (KEBN) in Part III, including its illustration in case studies of Bayesian network development in Chapter 11.

Another important response to the difficulty of building Bayesian networks by hand is the development of methods for their automated learning — the machine learning of Bayesian networks (aka "data mining"). In Part II we introduce and develop the main methods for learning Bayesian networks with reference to the theory of causality underlying them. These techniques logically come before the knowledge engineering methodology, since that draws upon and integrates machine learning with expert elicitation.

1.8 Are Bayesian networks Bayesian?

Many AI researchers like to point out that Bayesian networks are not inherently Bayesian at all; some have even claimed that the label is a misnomer. At the 2002 Australasian Data Mining Workshop, for example, Geoff Webb made the former claim. Under questioning it turned out he had two points in mind: (1) Bayesian networks are frequently "data mined" (i.e., learned by some computer program) via non-Bayesian methods. (2) Bayesian networks at bottom represent probabilities; but probabilities can be interpreted in any number of ways, including as some form of frequency; hence, the networks are not intrinsically either Bayesian or non-Bayesian, they simply represent values needing further interpretation.

These two points are entirely correct. We shall ourselves present non-Bayesian methods for automating the learning of Bayesian networks from statistical data. We

© 2004 by Chapman & Hall/CRC Press LLC

shall also present Bayesian methods for the same, together with some evidence of their superiority. The interpretation of the probabilities represented by Bayesian networks is open so long as the philosophy of probability is considered an open question. Indeed, much of the work presented here ultimately depends upon the probabilities being understood as *physical probabilities*, and in particular as propensities or probabilities determined by propensities. Nevertheless, we happily invoke the Principal Principle: where we are convinced that the probabilities at issue reflect the true propensities in a physical system we are certainly going to use them in assessing our own degrees of belief.

The advantages of the Bayesian network representations are largely in simplifying conditionalization, planning decisions under uncertainty and explaining the outcome of stochastic processes. These purposes all come within the purview of a clearly Bayesian interpretation of what the probabilities mean, and so, we claim, the Bayesian network technology which we here introduce is aptly named: it provides the technical foundation for a truly Bayesian artificial intelligence.

1.9 Summary

How best to reason about uncertain situations has always been of concern. From the 17th century we have had available the basic formalism of probability calculus, which is far and away the most promising formalism for coping with uncertainty. Probability theory has been used widely, but not deeply, since then. That is, the elementary ideas have been applied to a great variety of problems — e.g., actuarial calculations for life insurance, coping with noise in measurement, business decision making, testing scientific theories, gambling — but the problems have typically been of highly constrained size, because of the computational infeasibility of conditionalization when dealing with large problems. Even in dealing with simplified problems, humans have had difficulty handling the probability computations. The development of Bayesian network technology automates the process and so promises to free us from such difficulties. At the same time, improvements in computer capacity, together with the ability of Bayesian networks to take computational advantage of any available independencies between variables, promise to both widen and deepen the domain of probabilistic reasoning.

1.10 Bibliographic notes

An excellent source of information about different attempts to formalize reasoning about uncertainty — including certainty factors, non-monotonic logics, Dempster-

© 2004 by Chapman & Hall/CRC Press LLC

Shafer calculus, as well as probability — is the anthology *Readings in Uncertain Reasoning* edited by Shafer and Pearl [253]. Three polemics against non-Bayesian approaches to uncertainty are those by Drew McDermott [185], Peter Cheeseman [42] and Kevin Korb [159]. For understanding Bayesian philosophy, Ramsey's original paper "Truth and Probability" is beautifully written, original and compelling [231]; for a more comprehensive and recent presentation of Bayesianism see Howson and Urbach's *Scientific Reasoning* [117] (a third edition is under preparation). For Bayesian decision analysis see Richard Jeffrey's *The Logic of Decision* [123]. DeGroot and Schervish [72] provide an accessible introduction to both the probability calculus and statistics.

Karl Popper's original presentation of the propensity interpretation of probability is [220]. This view is related to the elaboration of a probabilistic account of causality in recent decades. Wesley Salmon [243] provides an overview of probabilistic causality.

Naive Bayes models, despite their simplicity, have done surprisingly well as predictive classifiers for data mining problems; see Mitchell's *Machine Learning* [192] for a discussion and comparison with other classifiers.

1.11 Technical notes

A Dutch book

Here is a simple Dutch book. Suppose someone assigns $P(A) = -0.1$, violating probability Axiom 2. Then $O(A) = -0.1/(1 - (-0.1)) = -0.1/1.1$. The reward for a bet on A with a \$1 stake is $\$(1 - P(U)) = \$1.1$ if A comes true and $\$ - P(U) = \$0.1$ if A is false. That's everywhere positive and so is a "Good Book." The Dutch book simply requires this agent to take the fair bet *against* A, which has the payoffs $-\$1.1$ if A is true and $-\$0.1$ otherwise.

The rehabilitated Dutch book

Following Hájek, we can show that incoherence (violating the probability axioms) leads to being "dominated" by someone who is coherent — that is, the coherent bettor can take advantage of offered bets that the incoherent bettor cannot and otherwise will do as well.

Suppose Ms. Incoherent assigns $P_I(U) < 1$ (where U is the universal event that *must* occur), for example. Then Ms. Incoherent will take any bet for U at odds of $P_I(U)/(1 - P_I(U))$ or greater. But Ms. Coherent has assigned $P_C(U) = 1$, of course, and so can take any bet for U at any odds offered greater than zero. So for the odds within the range $[0, \frac{P_I(U)}{1-P_I(U)}]$ Ms. Coherent is guaranteed a profit whereas Ms. Incoherent is sitting on her hands.

© 2004 by Chapman & Hall/CRC Press LLC

NP hardness
A problem is Non-deterministic Polynomial-time (NP) if it is solvable in polynomial time on a non-deterministic Turing machine. A problem is Non-deterministic Polynomial time hard (NP hard) if every problem that is NP can be translated into this NP hard problem in polynomial time. If there is a polynomial time solution to any NP hard problem, then because of polynomial time translatability for all other NP problems, there must be a polynomial time solution to all NP problems. No one knows of a polynomial time solution to any NP hard problem; the best known solutions are exponentially explosive. Thus, "NP hard" problems are generally regarded as computationally intractable. (The classic introduction to computational complexity is [89].)

1.12 Problems

Probability Theory

Problem 1

Prove that the conditional probability function $P(\cdot|e)$, if well defined, is a probability function (i.e., satisfies the three axioms of Kolmogorov).

Problem 2

Given that two pieces of evidence e_1 and e_2 are conditionally independent given the hypothesis — i.e., $P(e_1|e_2, h) = P(e_1|h)$ — prove the "product rule": $P(e_1, e_2|h) = P(e_1|h) \times P(e_2|h)$.

Problem 3

Prove the theorems of §1.3.1, namely the Total Probability theorem and the Chain Rule.

Problem 4

There are five containers of milk on a shelf; unbeknownst to you, two of them have passed their use-by date. You grab two at random. What's the probability that neither have passed their use-by date? Suppose someone else has got in just ahead of you, taking one container, after examining the dates. What's the probability that the two you take at random after that are ahead of their use-by dates?

Problem 5

The probability of a child being a boy (or a girl) is 0.5 (let us suppose). Consider all the families with exactly two children. What is the probability that such a family has two girls given that it has at least one girl?

© 2004 by Chapman & Hall/CRC Press LLC

Problem 6

The frequency of male births at the Royal Women's Hospital is about 51 in 100. On a particular day, the last eight births have been female. The probability that the next birth will be male is:

1. About 51%
2. Clearly greater than 51%
3. Clearly less than 51%
4. Almost certain
5. Nearly zero

Bayes' Theorem

Problem 7

After winning a race, an Olympic runner is tested for the presence of steroids. The test comes up positive, and the athlete is accused of doping. Suppose it is known that 5% of all victorious Olympic runners do use performance-enhancing drugs. For this particular test, the probability of a positive finding given that drugs are used is 95%. The probability of a false positive is 2%. What is the (posterior) probability that the athlete did in fact use steroids, given the positive outcome of the test?

Problem 8

You consider the probability that a coin is double-headed to be 0.01 (call this option h'); if it isn't double-headed, then it's a fair coin (call this option h). For whatever reason, you can only test the coin by flipping it and examining the coin (i.e., you can't simply examine both sides of the coin). In the worst case, how many tosses do you need before having a posterior probability for either h or h' that is greater than 0.99, i.e., what's the maximum number of tosses until that happens?

Problem 9

(Adapted from [83].) Two cab companies, the Blue and the Green, operate in a given city. Eighty-five percent of the cabs in the city are Blue; the remaining 15% are Green. A cab was involved in a hit-and-run accident at night. A witness identified the cab as a Green cab. The court tested the witness' ability to distinguish between Blue and Green cabs under night-time visibility conditions. It found that the witness was able to identify each color correctly about 80% of the time, but confused it with the other color about 20% of the time.

What are the chances that the errant cab was indeed Green, as the witness claimed?

© 2004 by Chapman & Hall/CRC Press LLC

Odds and Expected Value

Problem 10

Construct a Dutch book against someone who violates the Axiom of Additivity. That is, suppose a Mr. Fuzzy declares about the weather tomorrow that $P(Sunny) = 0.5, P(Inclement) = 0.5$, and $P(Sunny\ or\ inclement) = 0.5$. Mr. Fuzzy and you agree about what will count as sunny and as inclement weather and you both agree that they are incompatible states. How can you construct a Dutch book against Fuzzy, using only fair bets?

Problem 11

A bookie offers you a ticket for \$5.00 which pays \$6.00 if Manchester United beats Arsenal and nothing otherwise. What are the odds being offered? To what probability of Manchester United winning does that correspond?

Problem 12

You are offered a Keno ticket in a casino which will pay you \$1 million if you win! It only costs you \$1 to buy the ticket. You choose 4 numbers out of a 9x9 grid of distinct numbers. You win if all of your 4 numbers come up in a random draw of four from the 81 numbers. What is the expected dollar value of this gamble?

Applications

Problem 13

(Note: this is the case of Sally Clark, convicted in the UK in 1999, and found innocent on appeal in 2003; see [120].) A mother was arrested after her second baby died a few months old, apparently of sudden infant death syndrome (SIDS), exactly as her first child had died a year earlier. According to prosecution testimony, about 2 in 17200 babies die of SIDS. So, according to their argument, there is only a probability of $(2/17200)^2 \approx 1/72000000$ that two such deaths would happen in the same family by chance alone. In other words, according to the prosecution, the woman was guilty beyond a reasonable doubt. The jury returned a guilty verdict, even though there was no significant evidence of guilt presented beyond this argument. Which of the following is the truth of the matter? Why?

1. Given the facts presented, the probability that the woman is guilty is greater than 99%, so the jury decided correctly.
2. The argument presented by the prosecution is irrelevant to the mother's guilt or innocence.
3. The prosecution argument is relevant but inconclusive.
4. The prosecution argument only establishes a probability of guilt of about 16%.
5. Given the facts presented, guilt and innocence are equally likely.

© 2004 by Chapman & Hall/CRC Press LLC

Problem 14

A DNA match between the defendant and a crime scene blood sample has a probability of 1/100000 if the defendant is innocent. There is no other significant evidence.

1. What is the probability of guilt?
2. Suppose we agree that the prior probability of guilt under the (unspecified) circumstances is 10%. What then is the probability of guilt?
3. The suspect has been picked up through a universal screening program applied to all Australians seeking a Medicare card. So far, 10 million people have been screened. What then is the probability of guilt?

© 2004 by Chapman & Hall/CRC Press LLC

Part I

PROBABILISTIC REASONING

© 2004 by Chapman & Hall/CRC Press LLC

1

Bayesian Reasoning

1.1 Reasoning under uncertainty

Artificial intelligence (AI), should it ever exist, will be an intelligence developed by humans, implemented as an artifact. The level of intelligence demanded by Alan Turing's famous test [275] — the ability to fool ordinary (unfoolish) humans about whether the other end of a dialogue is being carried on by a human or by a computer — is some indication of what AI researchers are aiming for. Such an AI would surely transform our technology and economy. We would be able to automate a great deal of human drudgery and paperwork. Since computers are universal, programs can be effortlessly copied from one system to another (to the consternation of those worried about intellectual property rights!), and the labor savings of AI support for bureaucratic applications of rules, medical diagnosis, research assistance, manufacturing control, etc. promises to be enormous. If a serious AI is ever developed.

There is little doubt that an AI will need to be able to reason logically. An inability to discover, for example, that a system's conclusions have reached inconsistency is more likely to be debilitating than the discovery of an inconsistency itself. For a long time there has also been widespread recognition that practical AI systems shall have to cope with **uncertainty** — that is, they shall have to deal with incomplete evidence leading to beliefs that fall short of knowledge, with fallible conclusions and the need to recover from error, called **non-monotonic reasoning**. Nevertheless, the AI community has been slow to recognize that any serious, general-purpose AI will need to be able to reason probabilistically, what we call here **Bayesian reasoning**.

There are at least three distinct forms of uncertainty which an intelligent system operating in anything like our world shall need to cope with:

1. **Ignorance.** The limits of our knowledge lead us to be uncertain about many things. Does our poker opponent have a flush or is she bluffing?
2. **Physical randomness or indeterminism.** Even if we know everything that we might care to investigate about a coin and how we impart spin to it when we toss it, there will remain an inescapable degree of uncertainty about whether it will land heads or tails when we toss it. A die-hard determinist might claim otherwise, that some unimagined amount of detailed investigation might someday reveal which way the coin will fall; but such a view is for the forseeable future a mere act of scientistic faith. We are all practical indeterminists.

© 2004 by Chapman & Hall/CRC Press LLC

3. **Vagueness.** Many of the predicates we employ appear to be vague. It is often unclear whether to classify a dog as a spaniel or not, a human as brave or not, a thought as knowledge or opinion.

Bayesianism is the philosophy that asserts that in order to understand human opinion as it ought to be, constrained by ignorance and uncertainty, the probability calculus is the single most important tool for representing appropriate strengths of belief. In this text we shall present Bayesian computational tools for reasoning with and about strengths of belief as probabilities; we shall also present a Bayesian view of physical randomness. In particular we shall consider a probabilistic account of causality and its implications for an intelligent agent's reasoning about its physical environment. We will not address the third source of uncertainty above, vagueness, which is fundamentally a problem about semantics and one which has no good analysis so far as we are aware.

1.2 Uncertainty in AI

The successes of formal logic have been considerable over the past century and have been received by many as an indication that logic should be the primary vehicle for **knowledge representation** and reasoning within AI. **Logicism** in AI, as this has been called, dominated AI research in the 1960s and 1970s, only losing its grip in the 1980s when artificial neural networks came of age. Nevertheless, even during the heyday of logicism, any number of practical problems were encountered where logic would not suffice, because uncertain reasoning was a key feature of the problem. In the 1960s, medical diagnosis problems became one of the first attempted application areas of AI programming. But there is no symptom or prognosis in medicine which is strictly logically implied by the existence of any particular disease or syndrome; so the researchers involved quickly developed a set of "probabilistic" relations. Because probability calculations are hard — in fact, NP hard in the number of variables [52] (i.e., computationally intractable; see §1.11) — they resorted to implementing what has subsequently been called "naive Bayes" (or, "Idiot Bayes"), that is, probabilistic updating rules which assume that symptoms are independent of each other given diseases.

The independence constraints required for these systems were so extreme that the systems were received with no wide interest. On the other hand, a very popular set of expert systems in the 1970s and 1980s were based upon Buchanan and Shortliffe's MYCIN, or the uncertainty representation within MYCIN which they called **certainty factors** [34]. Certainty factors (CFs) were obtained by first eliciting from experts a "degree of increased belief" which some evidence e should imply for a hypothesis h, $MB(h, e) \in [0, 1]$, and also a corresponding "degree of increased

© 2004 by Chapman & Hall/CRC Press LLC

disbelief," $MD(h,e) \in [0,1]$. These were then combined:

$$CF(h,e) = MB(h,e) - MD(h,e) \in [-1,1]$$

This division of changes in "certainty" into changes in belief and disbelief reflects the curious notion that belief and disbelief are not necessarily related to one another (cf. [34, section 11.4]). A popular AI text, for example, sympathetically reports that "it is often the case that an expert might have confidence 0.7 (say) that some relationship is true and have no feeling about it being not true" [175, p.329]. The same point can be put more simply: experts are often inconsistent. Our goal in Bayesian modeling is, at least largely, to find the most accurate representation of a real system about which we may be receiving inconsistent expert advice, rather than finding ways of modeling the inconsistency itself.

Regardless of how we may react to this interpretation of certainty factors, no operational semantics for CFs were provided by Buchanan and Shortliffe. This meant that no real guidance could be given to experts whose opinions were being solicited. Most likely, they simply assumed that they were being asked for conditional probabilities of h given e and of $\neg h$ given e. And, indeed, there finally was a probabilistic semantics given for certainty factors: David Heckerman in 1986 [104] proved that a consistent probabilistic interpretation of certainty factors* would once again require strong independence assumptions: in particular that, when combining multiple pieces of evidence, the different pieces of evidence must always be independent of each other. Whereas this appears to be a desirable simplification of **rule-based** systems, allowing rules to be "modular," with the combined impact of diverse evidence being a compositional function of their separate impacts it is easy to demonstrate that the required independencies are frequently unavailable. The price of rule-based simplicity is irrelevance.

Bayesian networks provide a natural representation of probabilities which allow for (and take advantage of, as we shall see in Chapter 2) any independencies that may hold, while not being limited to problems satisfying strong independence requirements. The combination of substantial increases in computer power with the Bayesian network's ability to use any existing independencies to computational advantage make the approximations and restrictive assumptions of earlier uncertainty formalisms pointless. So we now turn to the main game: understanding and representing uncertainty with probabilities.

1.3 Probability calculus

The probability calculus allows us to represent the independencies which other systems require, but also allows us to represent any dependencies which we may need.

*In particular, a mapping of certainty factors into likelihood ratios.

© 2004 by Chapman & Hall/CRC Press LLC

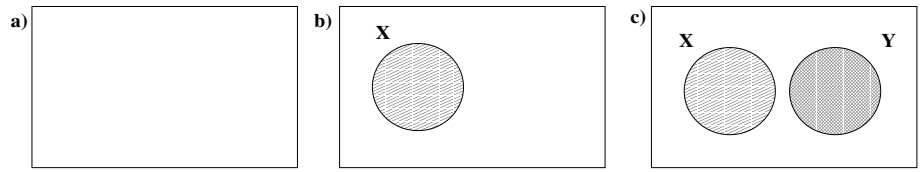


FIGURE 1.1
(a) The event space U; (b) $P(X)$; (c) $P(X \cup Y)$.

The probability calculus was specifically invented in the 17th century by Fermat and Pascal in order to deal with the problems of physical uncertainty introduced by gambling. But it did not take long before it was noticed that the concept of probability could be used in dealing also with the uncertainties introduced by ignorance, leading Bishop Butler to declare in the 18th century that "probability is the very guide to life." So now we introduce this formal language of probability, in a very simple way using Venn diagrams.

Let U be the universe of possible events; that is, if we are uncertain about which of a number of possibilities is true, we shall let U represent all of them collectively (see Figure 1.1(a)). Then the maximum probability must apply to the true event lying within U. By convention we set the maximum probability to 1, giving us Kolmogorov's first axiom for probability theory [153]:

Axiom 1.1 $P(U) = 1$

This probability mass, summing or integrating to 1, is distributed over U, perhaps evenly or perhaps unevenly. For simplicity we shall assume that it is spread evenly, so that the probability of any region is strictly proportional to its area. For any such region X its area cannot be negative, even if X is empty; hence we have the second axiom (Figure 1.1(b)):

Axiom 1.2 *For all* $X \subseteq U, P(X) \geq 0$

We need to be able to compute the probability of combined events, X and Y. This is trivial if the two events are mutually exclusive, giving us the third and last axiom (Figure 1.1(c)), known as **additivity**:

Axiom 1.3 *For all X,Y* $\subseteq U$, if $X \cap Y = \emptyset$, then $P(X \cup Y) = P(X) + P(Y)$

Any function over a field of subsets of U satisfying the above axioms will be a probability function[†].

A simple theorem extends addition to events which overlap (i.e., sets which intersect):

[†] A set-theoretic field is a set of sets containing U and $\emptyset$ and is closed under union, intersection and complementation.

© 2004 by Chapman & Hall/CRC Press LLC

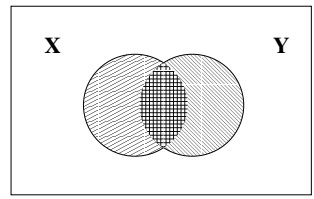


FIGURE 1.2
Conditional probability: $P(X|Y) = P(X \cap Y)/P(Y)$.

Theorem 1.1 *For all X,Y* $\subseteq U, P(X \cup Y) = P(X) + P(Y) - P(X \cap Y)$.

This can be intuitively grasped from Figure 1.2: the area of $X \cup Y$ is less than area of X plus the area of Y because when adding the area of intersection $X \cap Y$ has been counted twice; hence, we simply remove the excess to find $P(X \cup Y)$ for any two events X and Y.

The concept of **conditional probability** is crucial for the useful application of the probability calculus. It is usually introduced by definition:

Definition 1.1 Conditional probability

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)}$$

That is, given that the event Y has occurred, or will occur, the probability that X will also occur is $P(X|Y)$. Clearly, if Y is an event with zero probability, then this conditional probability is undefined. This is not an issue for probability distributions which are **positive**, since, by definition, they are non-zero over every event. A simple way to think about probabilities conditional upon Y is to imagine that the universe of events U has shrunk to Y. The conditional probability of X on Y is just the measure of what is left of X relative to what is left of Y; in Figure 1.2 this is just the ratio of the darker area (representing $X \cap Y$) to the area of Y. This way of understanding conditional probability is justified by the fact that the conditional function $P(\cdot|Y)$ is itself a probability function[‡] — that is, it provably satisfies the three axioms of probability.

Another final probability concept we need to introduce is that of **independence** (or, marginal independence). Two events X and Y are probabilistically independent (in notation, $X \perp\!\!\!\perp Y$) whenever conditioning on one leaves the probability of the other unchanged:

Definition 1.2 Independence $X \perp\!\!\!\perp Y \equiv P(X|Y) = P(X)$

This is provably symmetrical: $X \perp\!\!\!\perp Y \equiv Y \perp\!\!\!\perp X$. The simplest examples of independence come from gambling. For example, two rolls of dice are normally independent.

[‡] $P(\cdot|Y)$ is just the function equal to $P(X|Y)$ for all $X \subset U$.

© 2004 by Chapman & Hall/CRC Press LLC

Getting a one with the first roll will neither raise nor lower the probability of getting a one the second time. If two events are **dependent**, then one coming true will *alter* the probability of the other. Thus, the probability of getting a diamond flush in poker (five diamonds in five cards drawn) is *not* simply $(1/4)^5 = 1/1024$: the probability that the first card drawn being a diamond is $1/4$, but the probability of subsequent cards being diamonds is influenced by the fact that there are then fewer diamonds left in the deck.

Conditional independence generalizes this concept to X and Y being independent given some additional event Z:

Definition 1.3 Conditional independence $X \perp\!\!\!\perp Y | Z \equiv P(X|Y,Z) = P(X|Z)$

This is a true generalization because, of course, Z can be the empty set $\emptyset$, when it reduces to marginal independence. Conditional independence holds when the event Z tells us everything that Y does about X and possibly more; once you know Z, learning Y is uninformative. For example, suppose we have two diagnostic tests for cancer X, an inexpensive but less accurate one, Y, and a more expensive and more accurate one, Z. If Z is more accurate partly because it effectively incorporates all of the diagnostic information available from Y, then knowing the outcome of Z will render an additional test of Y irrelevant — Y will be "screened off" from X by Z.

1.3.1 Conditional probability theorems

We introduce without proof two theorems on conditional probability which will be of frequent use:

Theorem 1.2 Total Probability *Assume the set of events $\{A_i\}$ is a partition of U; i.e., $\bigcup_i A_i = U$ and for any distinct i and j $A_i \cap A_j = \emptyset$. Then*

$$P(U) = \sum_i P(A_i)$$

We can equally well partition the probability of any particular event B instead of the whole event space. In other words, under the above conditions (and if $\forall i A_i \neq \emptyset$),

$$P(B) = \sum_i P(B|A_i)$$

We shall refer to either formulation under the title "Total Probability."

Theorem 1.3 The Chain Rule *Given three events A, B, C,*

$$P(C|A) = P(C|B)P(B|A) + P(C|\neg B)P(\neg B|A)$$

assuming the conditional probabilities are defined. This allows us to divide the probabilistic influence of C on A across the different states of a third variable. (Here, the third variable is binary, but the theorem is easily generalized to variables of arbitrary arity.)

© 2004 by Chapman & Hall/CRC Press LLC

1.3.2 Variables

Although we have introduced probabilities over events, in most of our discussion we shall be concerned with probabilities over **random variables**. A random variable is a variable which reports the outcome of some measurement process. It can be related to events, to be sure. For example, instead of talking about which event in a partition $\{A_i\}$ turns out to be the case, we can equivalently talk about which state x_i the random variable X takes, which we write $X = x_i$. The set of states a variable X can take form its state space, written Ω_X, and its size (or arity) is $|\Omega_X|$.

The discussion thus far has been implicitly of discrete variables, those with a finite state space. However, we need also to introduce the concept of probability distributions over **continuous variables**, that is, variables which range over real numbers, like *Temperature*. For the most part in this text we shall be using probability distributions over discrete variables (events), for two reasons. First, the Bayesian network technology is primarily oriented towards handling discrete state variables, for example the inference algorithms of Chapter 3. Second, for most purposes continuous variables can be **discretized**. For example, temperatures can be divided into ranges of ± 5 degrees for many purposes; and if that is too crude, then they can be divided into ranges of ± 1 degree, etc.

Despite our ability to evade probabilities over continuous variables much of the time, we shall occasionally need to discuss them. We introduce these probabilities by first starting with a **density function** $f(X)$ defined over the continuous variable X. Intuitively, the density assigns a weight or measure to each possible value of X and can be approximated by a finely partitioned histogram reporting samples from X. Although the density is not itself a probability function, it can be used to generate one so long as $f(\cdot)$ satisfies the conditions:

$$f(x) \geq 0 \tag{1.1}$$

$$\int_{-\infty}^{+\infty} f(x)dx = 1 \tag{1.2}$$

In words: each point value is positive or zero and all values integrate to 1. In that case we can define the **cumulative probability distribution** $F(\cdot)$ by

$$F(x) = P(X \leq x) = \int_{x' \leq x} f(x')dx' \tag{1.3}$$

This function assigns probabilities to ranges from each possible value of x down to negative infinity. Note that we can analogously define probabilites over any continuous interval of values for X, so long as the interval is not degenerate (equal to a point). In effect, we obtain a probability distribution by discretizing the continuous variable — i.e., by looking at the mass of the density function over intervals.

© 2004 by Chapman & Hall/CRC Press LLC

1.4 Interpretations of probability

There have been two main contending views about how to understand probability. One asserts that probabilities are fundamentally dispositional properties of non-deterministic physical systems, the classical such systems being gambling devices, such as dice. This view is particularly associated with **frequentism**, advocated in the 19th century by John Venn [284], identifying probabilities with long-run frequencies of events. The obvious complaint that short-run frequencies clearly do not match probabilities (e.g., if we toss a coin only once, we would hardly conclude that it's probability of heads is either one or zero) does not actually get anywhere, since no claim is made identifying short-run frequencies with probabilities. A different complaint does bite, however, namely that the distinction between short-run and long-run is vague, leaving the commitments of this frequentist interpretation unclear. Richard von Mises in the early 20th century [286] fixed this problem by formalizing the frequency interpretation, identifying probabilities with frequency limits in infinite sequences satisfying certain assumptions about randomness. Some version of this frequency interpretation is commonly endorsed by statisticians.

A more satisfactory theoretical account of physical probability arises from Karl Popper's observation [220] that the frequency interpretation, precise though it was, fails to accommodate our intuition that probabilities of singular events exist and are meaningful. If, in fact, we do toss a coin once and once only, and if this toss should *not* participate in some infinitude (or even large number) of appropriately similar tosses, it would not for that reason fail to have some probability of landing heads. Popper identified physical probabilities with the **propensities** (dispositions) of physical systems ("chance setups") to produce particular outcomes, whether or not those dispositions were manifested repeatedly. An alternative that amounts to much the same thing is to identify probabilities with counterfactual frequencies generated by hypothetically infinite repetitions of an experiment [282].

Whether physical probability is relativized to infinite random sequences, infinite counterfactual sequences or chance setups, these accounts all have in common that the assertion of a probability is relativized to *some* definite physical process or the outcomes it generates.

The traditional alternative to the concept of physical probability is to think of probabilities as reporting our subjective degrees of belief. This view was expressed by Thomas Bayes [16] (Figure 1.3) and Pierre Simon de Laplace [68] two hundred years ago. This is a more general account of probability in that we have subjective belief in a huge variety of propositions, many of which are not at all clearly tied to a physical process capable even in principle of generating an infinite sequence of outcomes. For example, most of us have a pretty strong belief in the Copernican hypothesis that the earth orbits the sun, but this is based on evidence not obviously the same as the outcome of a sampling process. We are not in any position to generate solar systems repeatedly and observe the frequency with which their planets revolve around the sun, for example. Bayesians nevertheless are prepared to talk about the probability

© 2004 by Chapman & Hall/CRC Press LLC

of the truth of the Copernican thesis and can give an account of the relation between that probability and the evidence for and against it. Since these probabilities are typically subjective, not clearly tied to physical models, most frequentists (hence, most statisticians) deny their meaningfulness. It is not insignificant that this leaves their (usual) belief in Copernicanism unexplained.

The first thing to make clear about this dispute between physicalists and Bayesians is that Bayesianism can be viewed as *generalizing* physicalist accounts of probability. That is, it is perfectly compatible with the Bayesian view of probability as measuring degrees of subjective belief to adopt what David Lewis dubbed the **Principal Principle** [171]: whenever you learn that the physical probability of an outcome is r, set your subjective probability for that outcome to r. This is really just common sense: you may think that the probability of a friend shaving his head is 0.01, but if you learn that he will do so if and only if a fair coin yet to be flipped lands heads, you'll revise your opinion accordingly.

So, the Bayesian and physical interpretations of probability are compatible, with the Bayesian interpretation *extending* the application of probability beyond what is directly justifiable in physical terms. That is the view we adopt here. But what justifies this extension?

FIGURE 1.3
Reverend Thomas Bayes (1702–1761).

© 2004 by Chapman & Hall/CRC Press LLC

1.5 Bayesian philosophy

1.5.1 Bayes' theorem

The origin of Bayesian philosophy lies in an interpretation of **Bayes' Theorem:**

Theorem 1.4 Bayes' Theorem

$$P(h|e) = \frac{P(e|h)P(h)}{P(e)}$$

This is a non-controversial (and simple) theorem of the probability calculus. Under its usual Bayesian interpretation, it asserts that the probability of a hypothesis h conditioned upon some evidence e is equal to its **likelihood** $P(e|h)$ times its probability prior to any evidence $P(h)$, **normalized** by dividing by $P(e)$ (so that the conditional probabilities of all hypotheses sum to 1). So much is not controvertible.

The further claim that this is a right and proper way of adjusting our beliefs in our hypotheses given new evidence is called **conditionalization**, and it is controversial.

Definition 1.4 Conditionalization *After applying Bayes' theorem to obtain $P(h|e)$ adopt that as your posterior degree of belief in h — or, $Bel(h) = P(h|e)$.*

Conditionalization, in other words, advocates belief updating via probabilities conditional upon the available evidence. It identifies **posterior probability** (the probability function after incorporating the evidence, which we are writing $Bel(\cdot)$) with **conditional probability** (the prior probability function conditional upon the evidence, which is $P(\cdot|e)$). Put thus, conditionalization may also seem non-controvertible. But there are certainly situations where conditionalization very clearly does not work. The two most basic such situations simply violate what are frequently explicitly stated as assumptions of conditionalization: (1) There must exist joint priors over the hypothesis and evidence spaces. Without a joint prior, Bayes' theorem cannot be used, so conditionalization is a non-starter. (2) The evidence conditioned upon, e, is all and only the evidence learned. This is called the **total evidence condition**. It is a significant restriction, since in many settings it cannot be guaranteed.

The first assumption is also significant. Many take it as the single biggest objection to Bayesianism to raise the question "Where do the numbers come from?" For example, the famous anti-Bayesian Clark Glymour [94] doesn't complain about Bayesian reasoning involving gambling devices, when the outcomes are engineered to start out equiprobable, but doubts that numbers can be found for more interesting cases. To this kind of objection Bayesians react in a variety of ways. In fact, the different varieties of response pretty much identify the different schools of Bayesianism. Objectivists, such as Rudolf Carnap [39] and Ed Jaynes [122], attempt to define prior probabilities based upon the structure of language. Extreme subjectivists, such as de Finetti [67], assert that it makes no difference what source your priors have: given

© 2004 by Chapman & Hall/CRC Press LLC

that de Finetti's representation theorem shows that non-extreme priors converge in the limit (under reasonable constraints), it just doesn't matter what priors you adopt.

The practical application of Bayesian reasoning does not appear to depend upon settling this kind of philosophical problem. A great deal of useful application can be done simply by refusing to adopt a dogmatic position and accepting common-sense prior probabilities. For example, if there are ten possible suspects in a murder mystery, a fair starting point for any one of them is a 1 in 10 chance of guilt; or, again, if burglaries occur in your neighborhood of 10,000 homes about once a day, then the probability of your having been burglarized within the last 24 hours might reasonably be given a prior probability of 1/10000.

Colin Howson points out that conditionalization is a valid rule of inference if and only if $Bel(e|h) = P(e|h)$, that is, if and only if your prior and posterior probability functions share the relevant conditional probabilities (cf. [116]). This is certainly a pertinent observation, since encountering some possible evidence may well inform us more about defects in our own conditional probability structure than about the hypothesis at issue. Since Bayes' theorem has $P(h|e)$ being proportional to $P(e|h)$, if the evidence leads us to revise $P(e|h)$, we will be in no position to conditionalize.

How to generate prior probabilities or new conditional probability structure is not dictated by Bayesian principles. Bayesian principles advise how to update probabilities once such a conditional probability structure has been adopted, given appropriate priors. Expecting Bayesian principles to answer all questions about reasoning is expecting too much. Nevertheless, we shall show that Bayesian principles implemented in computer programs can deliver a great deal more than the nay-sayers have ever delivered.

Definition 1.5 Jeffrey conditionalization *Suppose your observational evidence does not correspond specifically to proposition e, but can be represented as a posterior shift in belief about e. In other words, posterior belief in e is not full but partial, having shifted from $P(e)$ to $Bel(e)$. Then, instead of Bayesian conditionalization, apply Jeffrey's update rule [123] for* **probability kinematics***: $Bel(h) = P(h|e)Bel(e) + P(h|\neg e)Bel(\neg e)$.*

Jeffrey's own example is one where your hypothesis is about the color of a cloth, the evidence proposition e describes the precise quality of your visual experience under good light, but you are afforded a view of the cloth only under candlelight, in such a way that you cannot exactly articulate what you have observed. Nevertheless, you have learned *something*, and this is reflected in a shift in belief about the quality of your visual experience. Jeffrey conditionalization is very intuitive, but again is not strictly valid. As a practical matter, the need for such partial updating is common in Bayesian modeling.

1.5.2 Betting and odds

Odds are the ratio between the cost of a bet in favor of a proposition and the reward should the bet be won. Thus, assuming a stake of \$1 (and otherwise simply rescaling

© 2004 by Chapman & Hall/CRC Press LLC

the terms of the bet), a bet at 1:19 odds costs \$1 and returns \$20 should the proposition come true (with the reward being \$20 minus the cost of the bet)[§]. The odds may be set at any ratio and may, or may not, have something to do with one's probabilities. Bookies typically set odds for and against events at a slight discrepancy with their best estimate of the probabilities, for their profit lies in the difference between the odds for and against.

While odds and probabilities may deviate, probabilities and **fair odds** $O(\cdot)$ are strictly interchangeable concepts. The fair odds in favor of h are defined simply as the ratio of the probability that h is true to the probability that it is not:

Definition 1.6 Fair odds

$$O(h) = \frac{P(h)}{1 - P(h)}$$

Given this, it is an elementary matter of algebraic manipulation to find $P(h)$ in terms of odds:

$$P(h) = \frac{O(h)}{1 + O(h)} \tag{1.4}$$

Thus, if a coin is fair, the probability of heads is 1/2, so the odds in favor of heads are 1:1 (usually described as "50:50"). Or, if the odds of getting "snake eyes" (two 1's) on the roll of two dice are 1:35, then the probability of this is:

$$\frac{1/35}{1 + 1/35} = \frac{1/35}{36/35} = 1/36$$

as will always be the case with fair dice. Or, finally, suppose that the probability an agent ascribes to the Copernican hypothesis (CH) is zero; then the odds that agent is giving to Copernicus having been wrong ($\neg CH$) are *infinite*:

$$O(\neg CH) = \frac{1}{0} = \infty$$

At these odds, incidentally, it is trivial that the agent can never reach a degree of belief in CH above zero on any finite amount of evidence, if relying upon conditionalization for updating belief.

With the concept of fair odds in hand, we can reformulate Bayes' theorem in terms of (fair) odds, which is often useful:

Theorem 1.5 Odds-Likelihood Bayes' Theorem

$$O(h|e) = \frac{P(e|h)}{P(e|\neg h)} O(h)$$

This is readily proven to be equivalent to Theorem 1.4. In English it asserts that the odds on h conditional upon the evidence e are equal to the prior odds on h times the **likelihood ratio** $P(e|h) : P(e|\neg h)$. Clearly, the fair odds in favor of h will rise if and only if the likelihood ratio is greater than one.

[§] It is common in sports betting to invert the odds, quoting the odds *against* a team winning, for example. This makes no difference; the ratio is simply reversed.

© 2004 by Chapman & Hall/CRC Press LLC

1.5.3 Expected utility

Generally, agents are able to assign utility (or, value) to the situations in which they find themselves. We know what we like, we know what we dislike, and we also know when we are experiencing neither of these. Given a general ability to order situations, and bets with definite probabilities of yielding particular situations, Frank Ramsey [231] demonstrated that we can identify particular utilities with each possible situation, yielding a **utility function**.

If we have a utility function $U(O_i|A)$ over every possible outcome of a particular action A we are contemplating, and if we have a probability for each such outcome $P(O_i|A)$, then we can compute the probability-weighted average utility for that action — otherwise known as the **expected utility** of the action:

Definition 1.7 Expected utility

$$EU(A) = \sum_i U(O_i|A) \times P(O_i|A)$$

It is commonly taken as axiomatic by Bayesians that agents ought to *maximize their expected utility*. That is, when contemplating a number of alternative actions, agents ought to decide to take that action which has the maximum expected utility. If you are contemplating eating strawberry ice cream or else eating chocolate ice cream, presumably you will choose that flavor which you prefer, other things being equal. Indeed, if you chose the flavor you liked *less*, we should be inclined to think that other things are *not* equal — for example, you are under some kind of external compulsion — or perhaps that you are not being honest about your preferences. Utilities have behavioral consequences *essentially*: any agent who consistently ignores the putative utility of an action or situation arguably does not have that utility.

Regardless of such foundational issues, we now have the conceptual tools necessary to understand what is fair about fair betting. **Fair bets** are fair because their expected utility is zero. Suppose we are contemplating taking the fair bet B on proposition h for which we assign probability $P(h)$. Then the expected utility of the bet is:

$$EU(B) = U(h|B)P(h|B) + U(\neg h|B)P(\neg h|B)$$

Typically, betting on a proposition has no effect on the probability that it is true (although this is not necessarily the case!), so $P(h|B) = P(h)$. Hence,

$$EU(B) = U(h|B)P(h) + U(\neg h|B)(1 - P(h))$$

Assuming a stake of 1 unit for simplicity, then by definition $U(h|B) = 1 - P(h)$ (i.e., this is the utility of h being true given the bet for h) while $U(\neg h|B) = -P(h)$, so,

$$EU(B) = (1 - P(h))P(h) - P(h)(1 - P(h)) = 0$$

Given that the bet has zero expected utility, the agent should be no more inclined to take the bet in favor of h than to take the opposite bet against h.

© 2004 by Chapman & Hall/CRC Press LLC

1.5.4 Dutch books

The original Dutch book argument of Ramsey [231] (see also [67]) claims to show that subjective degrees of belief, if they are to be rational, *must* obey the probability calculus. It has the form of a *reductio ad absurdum* argument:

1. A rational agent should be willing to take either side of any combination of fair bets.
2. A rational agent should never be willing to take a combination of bets which guarantees a loss.
3. Suppose a rational agent's degrees of belief violate one or more of the axioms of probability.
4. Then it is provable that some combination of fair bets will lead to a guaranteed loss.
5. Therefore, the agent is both willing and not willing to take this combination of bets.

Now, the inferences to (4) and (5) in this argument are not in dispute (see §1.11 for a simple demonstration of (4) for one case). A *reductio* argument needs to be resolved by finding a prior assumption to blame, and concluding that it is false. Ramsey, and most Bayesians to date, supposed that the most plausible way of relieving the contradiction of (5) is by refusing to suppose that a rational agent's degrees of belief may violate the axioms of probability. This result can then be generalized beyond settings of explicit betting by taking "bets with nature" as a metaphor for decision-making generally. For example, walking across the street is in some sense a bet about our chances of reaching the other side.

Some anti-Bayesians have preferred to deny (1), insisting for example that it would be uneconomic to invest in bets with zero expected value (e.g., [48]). But the ascription of the radical incoherence in (5) simply to the willingness of, say, bored aristocrats to place bets that will net them nothing clearly will not do: the effect of incoherence is entirely out of proportion with the proposed cause of effeteness.

Alan Hájek [99] has recently pointed out a more plausible objection to (2). In the scenarios presented in Dutch books there is always some combination of bets which guarantees a net loss whatever the outcomes on the individual bets. But equally there is always some combination of bets which guarantees a net gain — a "Good Book." So, one agent's half-empty glass is another's half-full glass! Rather than dismiss the Dutch-bookable agent as irrational, we might commend it for being open to a guaranteed win! So, Hájek's point seems to be that there is a fundamental symmetry in Dutch book arguments which leaves open the question whether violating probability axioms is rational or not. Certainly, when metaphorically extending betting to a "struggle" with Nature, it becomes rather implausible that She is really out to Dutch book us!

Hájek's own solution to the problem posed by his argument is to point out that whenever an agent violates the probability axioms there will be some variation of its system of beliefs which is guaranteed to win money whenever the original system is

© 2004 by Chapman & Hall/CRC Press LLC

guaranteed to win, and which is also capable of winning in some situations when the original system is not. So the variant system of belief in some sense dominates the original: it is everywhere at least as good as the original and in some places better. In order to guarantee that your system of beliefs cannot be dominated, you must be probabilistically coherent (see §1.11). This, we believe, successfully rehabilitates the Dutch book in a new form.

Rather than rehabilitate, a more obviously Bayesian response is to consider the probability of a bookie hanging around who has the smarts to pump our agent of its money and, again, of a simpleton hanging around who will sign up the agent for guaranteed winnings. In other words, for rational choice surely what matters is the relative expected utility of the choice. Suppose, for example, that we are offered a set of bets which has a guaranteed loss of $10. Should we take it? The Dutch book assumes that accepting the bet is irrational. But, if the one and only alternative available is another bet with an expected loss of $1,000, then it no longer seems so irrational. An implicit assumption of the Dutch book has always been that betting is voluntary and when all offered bets are turned down the expected utility is zero. The further implicit assumption pointed out by Hájek's argument is that there is always a shifty bookie hanging around ready to take advantage of us. No doubt that is not always the case, and instead there is only some probability of it. Yet referring the whole matter of justifying the use of Bayesian probability to expected utility smacks of circularity, since expectation is understood in terms of Bayesian probability.

Aside from invoking the rehabilitated Dutch book, there is a more pragmatic approach to justifying Bayesianism, by looking at its importance for dealing with cases of practical problem solving. We take Bayesian principles to be normative, and especially to be a proper guide, under some range of circumstances, to evaluating hypotheses in the light of evidence. The form of justification that we think is ultimately most compelling is the "method of reflective equilibrium," generally attributed to Goodman [96] and Rawls [232], but first set out by Aristotle [10]. In a nutshell, it asserts that the normative principles to accept are those which best accommodate our basic, unshakable intuitions about what is good and bad (e.g., paradigmatic judgments of correct inference in simple domains, such as gambling) and which best integrate with relevant theory and practice. We now present some cases which Bayesian principle handles readily, and better than any alternative normative theory.

1.5.5 Bayesian reasoning examples

1.5.5.1 Breast Cancer

Suppose the women attending a particular clinic show a long-term chance of 1 in 100 of having breast cancer. Suppose also that the initial screening test used at the clinic has a false positive rate of 0.2 (that is, 20% of women without cancer will test positive for cancer) and that it has a false negative rate of 0.1 (that is, 10% of women with cancer will test negative). The laws of probability dictate from this last fact that the probability of a positive test given cancer is 90%. Now suppose that you are such a woman who has just tested positive. What is the probability that you have cancer?

© 2004 by Chapman & Hall/CRC Press LLC

This problem is one of a class of probability problems which has become notorious in the cognitive psychology literature (cf. [277]). It seems that very few people confronted with such problems bother to pull out pen and paper and compute the right answer via Bayes' theorem; even fewer can get the right answer without pen and paper. It appears that for many the probability of a positive test (which is observed) given cancer (i.e., 90%) dominates things, so they figure that they have quite a high chance of having cancer. But substituting into Theorem 1.4 gives us:

$$P(Cancer|Pos) = \frac{P(Pos|Cancer)P(Cancer)}{P(Pos)}$$

Note that the probability of *Pos* given *Cancer* — which is the likelihood 0.9 — is only *one* term on the right hand side; the other crucial term is the prior probability of cancer. Cognitive psychologists studying such reasoning have dubbed the dominance of likelihoods in such scenarios "base-rate neglect," since the base rate (prior probability) is being suppressed [137]. Filling in the formula and computing the conditional probability of *Cancer* given *Pos* gives us quite a different story:

$$\begin{aligned} P(Cancer|Pos) &= \frac{P(Pos|Cancer)P(Cancer)}{P(Pos)} \\ &= \frac{P(Pos|Cancer)P(Cancer)}{P(Pos|Cancer)P(Cancer) + P(Pos|\neg Cancer)P(\neg Cancer)} \\ &= \frac{0.9 \times 0.01}{0.9 \times 0.01 + 0.2 \times 0.99} \\ &= \frac{0.009}{0.009 + 0.198} \\ &\approx 0.043 \end{aligned}$$

Now the discrepancy between 4% and 80 or 90% is no small matter, particularly if the consequence of an error involves either unnecessary surgery or (in the reverse case) leaving a cancer untreated. But decisions similar to these are constantly being made based upon "intuitive feel" — i.e., without the benefit of paper and pen, let alone Bayesian networks (which are simpler to use than paper and pen!).

1.5.5.2 People v. Collins

The legal system is replete with misapplications of probability and with incorrect claims of the irrelevance of probabilistic reasoning as well.

In 1964 an interracial couple was convicted of robbery in Los Angeles, largely on the grounds that they matched a highly improbable profile, a profile which fit witness reports [272]. In particular, the two robbers were reported to be

- A man with a mustache
- Who was black and had a beard
- And a woman with a ponytail
- Who was blonde

© 2004 by Chapman & Hall/CRC Press LLC

- The couple was interracial
- And were driving a yellow car

The prosecution suggested that these characteristics had the following probabilities of being observed at random in the LA area:

1. A man with a mustache 1/4
2. Who was black and had a beard 1/10
3. And a woman with a ponytail 1/10
4. Who was blonde 1/3
5. The couple was interracial 1/1000
6. And were driving a yellow car 1/10

The prosecution called an instructor of mathematics from a state university who apparently testified that the "product rule" applies to this case: where mutually independent events are being considered jointly, the joint probability is the product of the individual probabilities[¶]. This last claim is, in fact, correct (see Problem 2 below); what is false is the idea that the product rule is relevant to this case. If we label the individual items of evidence e_i $(i = 1, \ldots, 6)$, the joint evidence e, and the hypothesis that the couple was guilty h, then what is claimed is

$$P(e|\neg h) = \prod_i P(e_i|\neg h) = 1/12000000$$

The prosecution, having made this inference, went on to assert that the probability the couple were innocent was no more than 1/12000000. The jury convicted.

As we have already suggested, the product rule does *not* apply in this case. Why not? Well, because the individual pieces of evidence are obviously *not* independent. If, for example, we know of the occupants of a car that one is black and the other has blonde hair, what then is the probability that the occupants are an interracial couple? Clearly not 1/1000! If we know of a man that he has a mustache, is the probability of having a beard unchanged? These claims are preposterous, and it is simply shameful that a judge, prosecutor and defence attorney could not recognize how preposterous they are — let alone the mathematics "expert" who testified to them. Since e_2 implies e_1, while e_2, e_3, e_4 jointly imply e_5 (to a fair approximation), a far better estimate for $P(e|\neg h)$ is $P(e_2|\neg h)P(e_3|\neg h)P(e_4|\neg h)P(e_6|\neg h) = 1/3000$.

To be sure, if we accepted that the probability of innocence were a mere 1/3000 we might well accept the verdict. But there is a more fundamental error in the prosecution reasoning than neglecting the conditional dependencies in the evidence. If, unlike the judge, prosecution and jury, we take a peek at Bayes' theorem, we discover that the probability of guilt $P(h|e)$ is *not* equal to $1 - P(e|\neg h)$; instead

$$P(h|e) = \frac{P(e|h)P(h)}{P(e|h)P(h) + P(e|\neg h)P(\neg h)}$$

[¶] Coincidentally, this is just the kind of independence required for certainty factors to apply.

© 2004 by Chapman & Hall/CRC Press LLC

Now if the couple in question *were* guilty, what are the chances the evidence accumulated would have been observed? That's a rather hard question to answer, but feeling generous towards the prosecution, let us simplify and say 1. That is, let us accept that $P(e|h) = 1$. Plugging in our assumptions we have thus far:

$$P(h|e) = \frac{P(h)}{P(h) + P(\neg h)/3000}$$

We are missing the crucial prior probability of a random couple being guilty of the robbery. Note that we cannot here use the prior probability of, for example, an interracial couple being guilty, since the fact that they are interracial is a piece of the evidence. The most plausible approach to generating a prior of the needed type is to count the number of couples in the LA area and give them an equal prior probability. In other words, if N is the number of possible couples in the LA area, $P(h) = 1/N$. So, what is N? The population at the time was about 6.5 million people [73]. If we conservatively take half of them as being eligible to be counted (e.g., being adult humans), this gives us 1,625,000 eligible males and as many females. If we simplify by supposing that they are all in heterosexual partnerships, that will introduce a slight bias in favor of innocence; if we also simplify by ignoring the possibility of people traveling in cars with friends, this will introduce a larger bias in favor of guilt. The two together give us 1,625,000 available couples, suggesting a prior probability of guilt of 1/1625000. Plugging this in we get:

$$P(h|e) = \frac{1/1625000}{1/1625000 + (1 - 1/1625000)/3000} \approx 0.002$$

In other words, even ignoring the huge number of trips with friends rather than partners, we obtain a 99.8% chance of innocence and so a very large probability of a nasty error in judgment. The good news is that the conviction (of the man only!) was subsequently overturned, partly on the basis that the independence assumptions are false. The bad news is that the appellate court finding also suggested that probabilistic reasoning is just irrelevant to the task of establishing guilt, which is a nonsense. One right conclusion about this case is that, assuming the likelihood has been *properly* worked out, a sensible prior probability must also be taken into account. In some cases judges have specifically ruled out all consideration of prior probabilities, while allowing testimony about likelihoods! Probabilistic reasoning which simply ignores half of Bayes' theorem is dangerous indeed!

Note that we do not claim that 99.8% is the best probability of innocence that can be arrived at for the case of People v. Collins. What we *do* claim is that, for the particular facts represented as having a particular probabilistic interpretation, this is far closer to a reasonable probability than that offered by the prosecution, namely 1/12000000. We also claim that the forms of reasoning we have here illustrated are *crucial* for interpreting evidence in general: namely, whether the offered items of evidence are conditionally independent and what the prior probability of guilt may be.

© 2004 by Chapman & Hall/CRC Press LLC

1.6 The goal of Bayesian AI

The most commonly stated goal for artificial intelligence is that of producing an artifact which performs difficult intellectual tasks at or beyond a human level of performance. Of course, machine chess programs have satisfied this criterion for some time now. Although some AI researchers have claimed that therefore an AI has been produced — that denying this is an unfair shifting of the goal line — it is absurd to think that we ought to be satisfied with programs which are strictly special-purpose and which achieve their performance using techniques that deliver nothing when applied to most areas of human intellectual endeavor.

Turing's test for intelligence appears to be closer to satisfactory: fooling ordinary humans with verbal behavior not restricted to any domain would surely demonstrate some important *general* reasoning ability. Many have pointed out that the conditions for Turing's test, strictly verbal behavior without any afferent or efferent nervous activity, yield at best some kind of disembodied, ungrounded intelligence. John Searle's Chinese Room argument, for example, can be interpreted as making such a case ([246]; for this kind of interpretation of Searle see [100] and [156]). A more convincing criterion for human-like intelligence is to require of an artificial intelligence that it be capable of powering a robot-in-the-world in such a way that the robot's performance cannot be distinguished from human performance in terms of behavior (disregarding, for example, whether the skin can be so distinguished). The program that can achieve this would surely satisfy any sensible AI researcher, or critic, that an AI had been achieved.

We are not, however, actually motivated by the idea of behaviorally cloning humans. If all we wish to do is reproduce humans, we would be better advised to employ the tried and true methods we have always had available. Our motive is to understand *how* such performance can be achieved. We are interested in knowing how humans perform the many interesting and difficult cognitive tasks encompassed by AI — such as, natural language understanding and generation, planning, learning, decision making — but we are also interested in knowing how they might be performed otherwise, and in knowing how they might be performed optimally. By building artifacts which model our best understanding of how humans do these things (which can be called **descriptive artificial intelligence**) and also building artifacts which model our best understanding of what is optimal in these activities (**normative artificial intelligence**), we can further our understanding of the nature of intelligence and also produce some very useful tools for science, government and industry.

As we have indicated through example, medical, legal, scientific, political and most other varieties of human reasoning either consider the relevant probabilistic factors and accommodate them or run the risk of introducing egregious and damaging errors. The goal of a Bayesian artificial intelligence is to produce a thinking agent which does as well or better than humans in such tasks, which can adapt to stochastic and changing environments, recognize its own limited knowledge and cope sensibly with these varied sources of uncertainty.

© 2004 by Chapman & Hall/CRC Press LLC

1.7 Achieving Bayesian AI

Given that we have this goal, how can we achieve it? The first step is to develop algorithms for doing Bayesian conditionalization properly and, insofar as possible, efficiently. This step has already been achieved, and the relevant algorithms are described in Chapters 2 and 3. The next step is to incorporate methods for computing expected utilities and develop methods for maximizing utility in decision making. We describe algorithms for this in Chapter 4. We would like to test these ideas in application: we describe some Bayesian network applications in Chapter 5.

These methods for probability computation are fairly well developed and their improvement remains an active area of research in AI today. The biggest obstacles to Bayesian AI having a broad and deep impact outside of the research community are the difficulties in developing applications, difficulties with eliciting knowledge from experts, and integrating and validating the results. One issue is that there is no clear methodology for developing, testing and deploying Bayesian network technology in industry and government — there is no recognized discipline of "software engineering" for Bayesian networks. We make a preliminary effort at describing one — Knowledge Engineering with Bayesian Networks (KEBN) in Part III, including its illustration in case studies of Bayesian network development in Chapter 11.

Another important response to the difficulty of building Bayesian networks by hand is the development of methods for their automated learning — the machine learning of Bayesian networks (aka "data mining"). In Part II we introduce and develop the main methods for learning Bayesian networks with reference to the theory of causality underlying them. These techniques logically come before the knowledge engineering methodology, since that draws upon and integrates machine learning with expert elicitation.

1.8 Are Bayesian networks Bayesian?

Many AI researchers like to point out that Bayesian networks are not inherently Bayesian at all; some have even claimed that the label is a misnomer. At the 2002 Australasian Data Mining Workshop, for example, Geoff Webb made the former claim. Under questioning it turned out he had two points in mind: (1) Bayesian networks are frequently "data mined" (i.e., learned by some computer program) via non-Bayesian methods. (2) Bayesian networks at bottom represent probabilities; but probabilities can be interpreted in any number of ways, including as some form of frequency; hence, the networks are not intrinsically either Bayesian or non-Bayesian, they simply represent values needing further interpretation.

These two points are entirely correct. We shall ourselves present non-Bayesian methods for automating the learning of Bayesian networks from statistical data. We

© 2004 by Chapman & Hall/CRC Press LLC

shall also present Bayesian methods for the same, together with some evidence of their superiority. The interpretation of the probabilities represented by Bayesian networks is open so long as the philosophy of probability is considered an open question. Indeed, much of the work presented here ultimately depends upon the probabilities being understood as *physical probabilities*, and in particular as propensities or probabilities determined by propensities. Nevertheless, we happily invoke the Principal Principle: where we are convinced that the probabilities at issue reflect the true propensities in a physical system we are certainly going to use them in assessing our own degrees of belief.

The advantages of the Bayesian network representations are largely in simplifying conditionalization, planning decisions under uncertainty and explaining the outcome of stochastic processes. These purposes all come within the purview of a clearly Bayesian interpretation of what the probabilities mean, and so, we claim, the Bayesian network technology which we here introduce is aptly named: it provides the technical foundation for a truly Bayesian artificial intelligence.

1.9 Summary

How best to reason about uncertain situations has always been of concern. From the 17th century we have had available the basic formalism of probability calculus, which is far and away the most promising formalism for coping with uncertainty. Probability theory has been used widely, but not deeply, since then. That is, the elementary ideas have been applied to a great variety of problems — e.g., actuarial calculations for life insurance, coping with noise in measurement, business decision making, testing scientific theories, gambling — but the problems have typically been of highly constrained size, because of the computational infeasibility of conditionalization when dealing with large problems. Even in dealing with simplified problems, humans have had difficulty handling the probability computations. The development of Bayesian network technology automates the process and so promises to free us from such difficulties. At the same time, improvements in computer capacity, together with the ability of Bayesian networks to take computational advantage of any available independencies between variables, promise to both widen and deepen the domain of probabilistic reasoning.

1.10 Bibliographic notes

An excellent source of information about different attempts to formalize reasoning about uncertainty — including certainty factors, non-monotonic logics, Dempster-

© 2004 by Chapman & Hall/CRC Press LLC

Shafer calculus, as well as probability — is the anthology *Readings in Uncertain Reasoning* edited by Shafer and Pearl [253]. Three polemics against non-Bayesian approaches to uncertainty are those by Drew McDermott [185], Peter Cheeseman [42] and Kevin Korb [159]. For understanding Bayesian philosophy, Ramsey's original paper "Truth and Probability" is beautifully written, original and compelling [231]; for a more comprehensive and recent presentation of Bayesianism see Howson and Urbach's *Scientific Reasoning* [117] (a third edition is under preparation). For Bayesian decision analysis see Richard Jeffrey's *The Logic of Decision* [123]. DeGroot and Schervish [72] provide an accessible introduction to both the probability calculus and statistics.

Karl Popper's original presentation of the propensity interpretation of probability is [220]. This view is related to the elaboration of a probabilistic account of causality in recent decades. Wesley Salmon [243] provides an overview of probabilistic causality.

Naive Bayes models, despite their simplicity, have done surprisingly well as predictive classifiers for data mining problems; see Mitchell's *Machine Learning* [192] for a discussion and comparison with other classifiers.

1.11 Technical notes

A Dutch book

Here is a simple Dutch book. Suppose someone assigns $P(A) = -0.1$, violating probability Axiom 2. Then $O(A) = -0.1/(1 - (-0.1)) = -0.1/1.1$. The reward for a bet on A with a \$1 stake is $\$(1 - P(U)) = \$1.1$ if A comes true and $\$ - P(U) = \$0.1$ if A is false. That's everywhere positive and so is a "Good Book." The Dutch book simply requires this agent to take the fair bet *against* A, which has the payoffs $-\$1.1$ if A is true and $-\$0.1$ otherwise.

The rehabilitated Dutch book

Following Hájek, we can show that incoherence (violating the probability axioms) leads to being "dominated" by someone who is coherent — that is, the coherent bettor can take advantage of offered bets that the incoherent bettor cannot and otherwise will do as well.

Suppose Ms. Incoherent assigns $P_I(U) < 1$ (where U is the universal event that *must* occur), for example. Then Ms. Incoherent will take any bet for U at odds of $P_I(U)/(1 - P_I(U))$ or greater. But Ms. Coherent has assigned $P_C(U) = 1$, of course, and so can take any bet for U at any odds offered greater than zero. So for the odds within the range $[0, \frac{P_I(U)}{1-P_I(U)}]$ Ms. Coherent is guaranteed a profit whereas Ms. Incoherent is sitting on her hands.

© 2004 by Chapman & Hall/CRC Press LLC

NP hardness
A problem is Non-deterministic Polynomial-time (NP) if it is solvable in polynomial time on a non-deterministic Turing machine. A problem is Non-deterministic Polynomial time hard (NP hard) if every problem that is NP can be translated into this NP hard problem in polynomial time. If there is a polynomial time solution to any NP hard problem, then because of polynomial time translatability for all other NP problems, there must be a polynomial time solution to all NP problems. No one knows of a polynomial time solution to any NP hard problem; the best known solutions are exponentially explosive. Thus, "NP hard" problems are generally regarded as computationally intractable. (The classic introduction to computational complexity is [89].)

1.12 Problems

Probability Theory

Problem 1

Prove that the conditional probability function $P(\cdot|e)$, if well defined, is a probability function (i.e., satisfies the three axioms of Kolmogorov).

Problem 2

Given that two pieces of evidence e_1 and e_2 are conditionally independent given the hypothesis — i.e., $P(e_1|e_2, h) = P(e_1|h)$ — prove the "product rule": $P(e_1, e_2|h) = P(e_1|h) \times P(e_2|h)$.

Problem 3

Prove the theorems of §1.3.1, namely the Total Probability theorem and the Chain Rule.

Problem 4

There are five containers of milk on a shelf; unbeknownst to you, two of them have passed their use-by date. You grab two at random. What's the probability that neither have passed their use-by date? Suppose someone else has got in just ahead of you, taking one container, after examining the dates. What's the probability that the two you take at random after that are ahead of their use-by dates?

Problem 5

The probability of a child being a boy (or a girl) is 0.5 (let us suppose). Consider all the families with exactly two children. What is the probability that such a family has two girls given that it has at least one girl?

© 2004 by Chapman & Hall/CRC Press LLC

Problem 6

The frequency of male births at the Royal Women's Hospital is about 51 in 100. On a particular day, the last eight births have been female. The probability that the next birth will be male is:

1. About 51%
2. Clearly greater than 51%
3. Clearly less than 51%
4. Almost certain
5. Nearly zero

Bayes' Theorem

Problem 7

After winning a race, an Olympic runner is tested for the presence of steroids. The test comes up positive, and the athlete is accused of doping. Suppose it is known that 5% of all victorious Olympic runners do use performance-enhancing drugs. For this particular test, the probability of a positive finding given that drugs are used is 95%. The probability of a false positive is 2%. What is the (posterior) probability that the athlete did in fact use steroids, given the positive outcome of the test?

Problem 8

You consider the probability that a coin is double-headed to be 0.01 (call this option h'); if it isn't double-headed, then it's a fair coin (call this option h). For whatever reason, you can only test the coin by flipping it and examining the coin (i.e., you can't simply examine both sides of the coin). In the worst case, how many tosses do you need before having a posterior probability for either h or h' that is greater than 0.99, i.e., what's the maximum number of tosses until that happens?

Problem 9

(Adapted from [83].) Two cab companies, the Blue and the Green, operate in a given city. Eighty-five percent of the cabs in the city are Blue; the remaining 15% are Green. A cab was involved in a hit-and-run accident at night. A witness identified the cab as a Green cab. The court tested the witness' ability to distinguish between Blue and Green cabs under night-time visibility conditions. It found that the witness was able to identify each color correctly about 80% of the time, but confused it with the other color about 20% of the time.

What are the chances that the errant cab was indeed Green, as the witness claimed?

© 2004 by Chapman & Hall/CRC Press LLC

Odds and Expected Value

Problem 10

Construct a Dutch book against someone who violates the Axiom of Additivity. That is, suppose a Mr. Fuzzy declares about the weather tomorrow that $P(Sunny) = 0.5, P(Inclement) = 0.5$, and $P(Sunny\ or\ inclement) = 0.5$. Mr. Fuzzy and you agree about what will count as sunny and as inclement weather and you both agree that they are incompatible states. How can you construct a Dutch book against Fuzzy, using only fair bets?

Problem 11

A bookie offers you a ticket for \$5.00 which pays \$6.00 if Manchester United beats Arsenal and nothing otherwise. What are the odds being offered? To what probability of Manchester United winning does that correspond?

Problem 12

You are offered a Keno ticket in a casino which will pay you \$1 million if you win! It only costs you \$1 to buy the ticket. You choose 4 numbers out of a 9x9 grid of distinct numbers. You win if all of your 4 numbers come up in a random draw of four from the 81 numbers. What is the expected dollar value of this gamble?

Applications

Problem 13

(Note: this is the case of Sally Clark, convicted in the UK in 1999, and found innocent on appeal in 2003; see [120].) A mother was arrested after her second baby died a few months old, apparently of sudden infant death syndrome (SIDS), exactly as her first child had died a year earlier. According to prosecution testimony, about 2 in 17200 babies die of SIDS. So, according to their argument, there is only a probability of $(2/17200)^2 \approx 1/72000000$ that two such deaths would happen in the same family by chance alone. In other words, according to the prosecution, the woman was guilty beyond a reasonable doubt. The jury returned a guilty verdict, even though there was no significant evidence of guilt presented beyond this argument. Which of the following is the truth of the matter? Why?

1. Given the facts presented, the probability that the woman is guilty is greater than 99%, so the jury decided correctly.
2. The argument presented by the prosecution is irrelevant to the mother's guilt or innocence.
3. The prosecution argument is relevant but inconclusive.
4. The prosecution argument only establishes a probability of guilt of about 16%.
5. Given the facts presented, guilt and innocence are equally likely.

© 2004 by Chapman & Hall/CRC Press LLC

Problem 14

A DNA match between the defendant and a crime scene blood sample has a probability of 1/100000 if the defendant is innocent. There is no other significant evidence.

1. What is the probability of guilt?
2. Suppose we agree that the prior probability of guilt under the (unspecified) circumstances is 10%. What then is the probability of guilt?
3. The suspect has been picked up through a universal screening program applied to all Australians seeking a Medicare card. So far, 10 million people have been screened. What then is the probability of guilt?

© 2004 by Chapman & Hall/CRC Press LLC

2

Introducing Bayesian Networks

2.1 Introduction

Having presented both theoretical and practical reasons for artificial intelligence to use probabilistic reasoning, we now introduce the key computer technology for dealing with probabilities in AI, namely **Bayesian networks.** Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections between them. These direct connections are often causal connections. In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available.

In this chapter we will describe how Bayesian networks are put together (the **syntax**) and how to interpret the information encoded in a network (the **semantics**). We will look at how to model a problem with a Bayesian network and the types of reasoning that can be performed.

2.2 Bayesian network basics

A **Bayesian network** is a graphical structure that allows us to represent and reason about an uncertain domain. The nodes in a Bayesian network represent a set of random variables from the domain. A set of directed **arcs** (or links) connects pairs of nodes, representing the direct dependencies between variables. Assuming discrete variables, the strength of the relationship between variables is quantified by conditional probability distributions associated with each node. The only constraint on the arcs allowed in a BN is that there must not be any directed cycles: you cannot return to a node simply by following directed arcs. Such networks are called directed acyclic graphs, or simply **dags**.

There are a number of steps that a **knowledge engineer**[*] must undertake when building a Bayesian network. At this stage we will present these steps as a sequence;

[*] Knowledge engineer in the jargon of AI means a practitioner applying AI technology.

© 2004 by Chapman & Hall/CRC Press LLC

however it is important to note that in the real-world the process is not so simple. In Chapter 9 we provide a fuller description of BN knowledge engineering.

Throughout the remainder of this section we will use the following simple medical diagnosis problem.

Example problem: Lung cancer. *A patient has been suffering from shortness of breath (called* dyspnoea*) and visits the doctor, worried that he has lung cancer. The doctor knows that other diseases, such as tuberculosis and bronchitis, are possible causes, as well as lung cancer. She also knows that other relevant information includes whether or not the patient is a smoker (increasing the chances of cancer and bronchitis) and what sort of air pollution he has been exposed to. A positive X-ray would indicate either TB or lung cancer*[†].

2.2.1 Nodes and values

First, the knowledge engineer must identify the variables of interest. This involves answering the question: what are the nodes to represent and what values can they take? For now we will consider only nodes that take discrete values. The values should be both **mutually exclusive** and **exhaustive**, which means that the variable must take on exactly one of these values at a time. Common types of discrete nodes include:

- Boolean nodes, which represent propositions, taking the binary values true (T) and false (F). In a medical diagnosis domain, the node *Cancer* would represent the proposition that a patient has cancer.
- Ordered values. For example, a node *Pollution* might represent a patient's pollution exposure and take the values $\{low, medium, high\}$.
- Integral values. For example, a node called *Age* might represent a patient's age and have possible values from 1 to 120.

Even at this early stage, modeling choices are being made. For example, an alternative to representing a patient's exact age might be to clump patients into different age groups, such as $\{baby, child, adolescent, young, middleaged, old\}$. The trick is to choose values that represent the domain efficiently, but with enough detail to perform the reasoning required. More on this later!

For our example, we will begin with the restricted set of nodes and values shown in Table 2.1. These choices already limit what can be represented in the network. For instance, there is no representation of other diseases, such as TB or bronchitis, so the system will not be able to provide the probability of the patient having them. Another limitation is a lack of differentiation, for example between a heavy or a light smoker, and again the model assumes at least some exposure to pollution. Note that all these nodes have only two values, which keeps the model simple, but in general there is no limit to the number of discrete values.

[†] This is a modified version of the so-called "Asia" problem [169], given in §2.5.3.

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 2.1
Preliminary choices of nodes and values for the lung cancer example

Node name	Type	Values
Pollution	Binary	{*low, high*}
Smoker	Boolean	{*T, F*}
Cancer	Boolean	{*T, F*}
Dyspnoea	Boolean	{*T, F*}
X-ray	Binary	{*pos, neg*}

2.2.2 Structure

The structure, or topology, of the network should capture qualitative relationships between variables. In particular, two nodes should be connected directly if one affects or causes the other, with the arc indicating the direction of the effect. So, in our medical diagnosis example, we might ask what factors affect a patient's chance of having cancer? If the answer is "Pollution and smoking," then we should add arcs from *Pollution* and *Smoker* to *Cancer*. Similarly, having cancer will affect the patient's breathing and the chances of having a positive X-ray result. So we add arcs from *Cancer* to *Dyspnoea*. The resultant structure is shown in Figure 2.1. It is important to note that this is just one possible structure for the problem; we look at alternative network structures in §2.4.3.

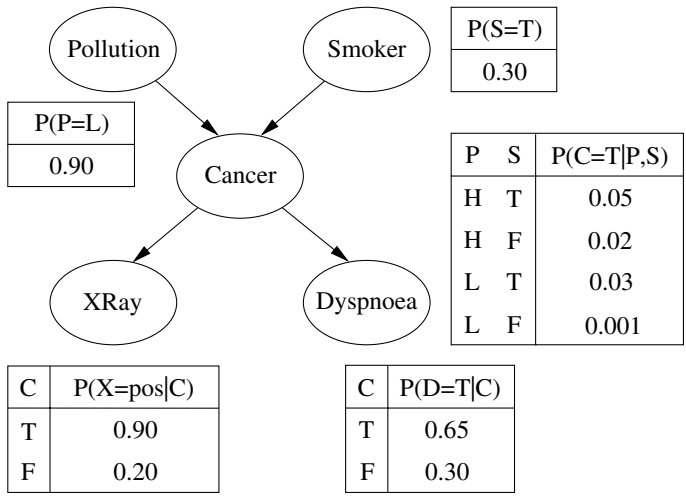


FIGURE 2.1
A BN for the lung cancer problem.

© 2004 by Chapman & Hall/CRC Press LLC

Structure terminology and layout

In talking about network structure it is useful to employ a family metaphor: a node is a **parent** of a **child**, if there is an arc from the former to the latter. Extending the metaphor, if there is a directed chain of nodes, one node is an **ancestor** of another if it appears earlier in the chain, whereas a node is a **descendant** of another node if it comes later in the chain. In our example, the *Cancer* node has two parents, *Pollution* and *Smoker*, while *Smoker* is an ancestor of both *X-ray* and *Dyspnoea*. Similarly, *X-ray* is a child of *Cancer* and descendant of *Smoker* and *Pollution*. The set of parent nodes of a node X is given by *Parents(X)*.

Another useful concept is that of the **Markov blanket** of a node, which consists of the node's parents, its children, and its children's parents. Other terminology commonly used comes from the "tree" analogy (even though Bayesian networks in general are graphs rather than trees): any node without parents is called a **root** node, while any node without children is called a **leaf** node. Any other node (non-leaf and non-root) is called an **intermediate node**. Given a causal understanding of the BN structure, this means that root nodes represent original causes, while leaf nodes represent final effects. In our cancer example, the causes *Pollution* and *Smoker* are root nodes, while the effects *X-ray* and *Dyspnoea* are leaf nodes.

By convention, for easier visual examination of BN structure, networks are usually laid out so that the arcs generally point from top to bottom. This means that the BN "tree" is usually depicted upside down, with roots at the top and leaves at the bottom[‡]!

2.2.3 Conditional probabilities

Once the topology of the BN is specified, the next step is to quantify the relationships between connected nodes – this is done by specifying a conditional probability distribution for each node. As we are only considering discrete variables at this stage, this takes the form of a conditional probability *table* (CPT).

First, for each node we need to look at all the possible combinations of values of those parent nodes. Each such combination is called an **instantiation** of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take each of its values.

For example, consider the *Cancer* node of Figure 2.1. Its parents are *Pollution* and *Smoking* and take the possible joint values $\{< H, T >, < H, F >, < L, T >, < L, F >\}$. The conditional probability table specifies in order the probability of cancer for each of these cases to be: $< 0.05, 0.02, 0.03, 0.001 >$. Since these *are* probabilities, and must sum to one over all possible states of the *Cancer* variable, the probability of no cancer is already implicitly given as one minus the above probabilities in each case; i.e., the probability of no cancer in the four possible parent instantiations is $< 0.95, 0.98, 0.97, 0.999 >$.

[‡] Oddly, this is the antipodean standard in computer science; we'll let you decide what that may mean about computer scientists!

© 2004 by Chapman & Hall/CRC Press LLC

Root nodes also have an associated CPT, although it is degenerate, containing only one row representing its prior probabilities. In our example, the prior for a patient being a smoker is given as 0.3, indicating that 30% of the population that the doctor sees are smokers, while 90% of the population are exposed to only low levels of pollution.

Clearly, if a node has many parents or if the parents can take a large number of values, the CPT can get very large! The size of the CPT is, in fact, exponential in the number of parents. Thus, for Boolean networks a variable with n parents requires a CPT with 2^{n+1} probabilities.

2.2.4 The Markov property

In general, modeling with Bayesian networks requires the assumption of the **Markov property**: there are no direct dependencies in the system being modeled which are not already explicitly shown via arcs. In our *Cancer* case, for example, there is no way for smoking to influence dyspnoea except by way of causing cancer (or not) — there is no hidden "backdoor" from smoking to dyspnoea. Bayesian networks which have the Markov property are also called **Independence-maps** (or, **I-maps** for short), since every independence suggested by the lack of an arc is real in the system.

Whereas the independencies suggested by a lack of arcs are generally required to exist in the system being modeled, it is not generally required that the arcs in a BN correspond to real dependencies in the system. The CPTs may be parameterized in such a way as to nullify any dependence. Thus, for example, every fully-connected Bayesian network can represent, perhaps in a wasteful fashion, any joint probability distribution over the variables being modeled. Of course, we shall prefer **minimal models** and, in particular, **minimal I-maps**, which are I-maps such that the deletion of any arc violates I-mapness by implying a non-existent independence in the system.

If, in fact, every arc in a BN happens to correspond to a direct dependence in the system, then the BN is said to be a **Dependence-map** (or, **D-map** for short). A BN which is both an I-map and a D-map is said to be a **perfect map**.

2.3 Reasoning with Bayesian networks

Now that we know how a domain and its uncertainty may be represented in a Bayesian network, we will look at how to use the Bayesian network to reason about the domain. In particular, when we observe the value of some variable, we would like to **condition** upon the new information. The process of conditioning (also called **probability propagation** or **inference** or **belief updating**) is performed via a "flow of information" through the network. Note that this information flow is *not* limited to the directions of the arcs. In our probabilistic system, this becomes the task of com-

© 2004 by Chapman & Hall/CRC Press LLC

puting the posterior probability distribution for a set of **query** nodes, given values for some **evidence** (or **observation**) nodes.

2.3.1 Types of reasoning

Bayesian networks provide full representations of probability distributions over their variables. That implies that they can be conditioned upon any subset of their variables, supporting any direction of reasoning.

For example, one can perform **diagnostic reasoning**, i.e., reasoning from symptoms to cause, such as when a doctor observes *Dyspnoea* and then updates his belief about *Cancer* and whether the patient is a *Smoker*. Note that this reasoning occurs in the *opposite* direction to the network arcs.

Or again, one can perform **predictive reasoning**, reasoning from new information about causes to new beliefs about effects, following the directions of the network arcs. For example, the patient may tell his physician that he is a smoker; even before any symptoms have been assessed, the physician knows this will increase the chances of the patient having cancer. It will also change the physician's expectations that the patient will exhibit other symptoms, such as shortness of breath or having a positive X-ray result.

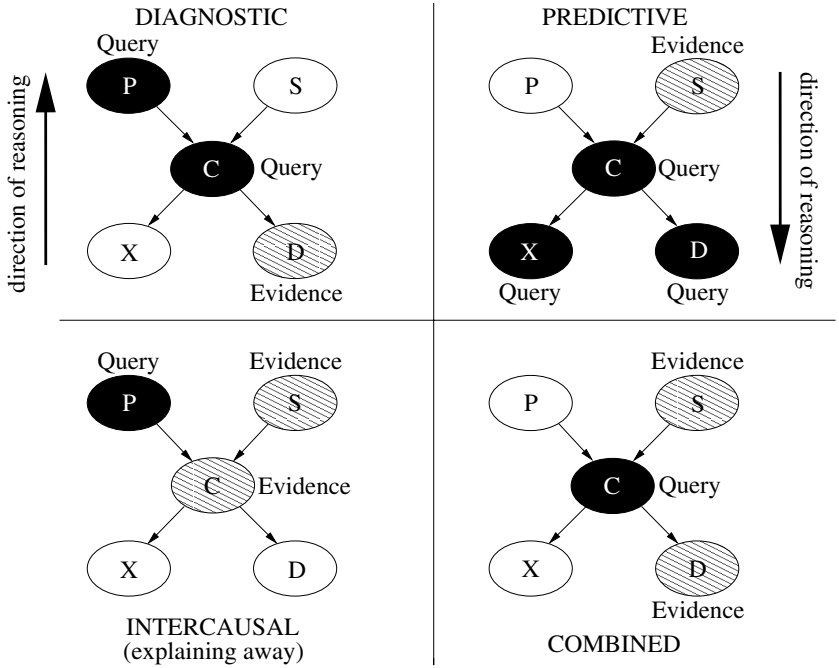


FIGURE 2.2
Types of reasoning.

© 2004 by Chapman & Hall/CRC Press LLC

A further form of reasoning involves reasoning about the mutual causes of a common effect; this has been called **intercausal reasoning**. A particular type called **explaining away** is of some interest. Suppose that there are exactly two possible causes of a particular effect, represented by a **v-structure** in the BN. This situation occurs in our model of Figure 2.1 with the causes *Smoker* and *Pollution* which have a common effect, *Cancer* (of course, reality is more complex than our example!). Initially, according to the model, these two causes are independent of each other; that is, a patient smoking (or not) does not change the probability of the patient being subject to pollution. Suppose, however, that we learn that Mr. Smith has cancer. This will raise our probability for both possible causes of cancer, increasing the chances both that he is a smoker and that he has been exposed to pollution. Suppose then that we discover that he is a smoker. This new information explains the observed cancer, which in turn *lowers* the probability that he has been exposed to high levels of pollution. So, even though the two causes are initially independent, with knowledge of the effect the presence of one explanatory cause renders an alternative cause less likely. In other words, the alternative cause has been *explained away*.

Since any nodes may be query nodes and any may be evidence nodes, sometimes the reasoning does not fit neatly into one of the types described above. Indeed, we can combine the above types of reasoning in any way. Figure 2.2 shows the different varieties of reasoning using the Cancer BN. Note that the last combination shows the simultaneous use of diagnostic and predictive reasoning.

2.3.2 Types of evidence

So Bayesian networks can be used for calculating new beliefs when new information – which we have been calling **evidence** – is available. In our examples to date, we have considered evidence as a definite finding that a node X has a particular value, x, which we write as $X = x$. This is sometimes referred to as **specific evidence**. For example, suppose we discover the patient is a smoker, then *Smoker=T*, which is specific evidence.

However, sometimes evidence is available that is not so definite. The evidence might be that a node Y has the value y_1 *or* y_2 (implying that all other values are impossible). Or the evidence might be that Y is *not* in state y_1 (but may take any of its other values); this is sometimes called a **negative evidence**.

In fact, the new information might simply be any new probability distribution over Y. Suppose, for example, that the radiologist who has taken and analyzed the X-ray in our cancer example is uncertain. He thinks that the X-ray looks positive, but is only 80% sure. Such information can be incorporated equivalently to Jeffrey conditionalization of §1.5.1, in which case it would correspond to adopting a new posterior distribution for the node in question. In Bayesian networks this is also known as **virtual evidence**. Since it is handled via likelihood information, it is also known as **likelihood evidence**. We defer further discussion of virtual evidence until Chapter 3, where we can explain it through the effect on belief updating.

© 2004 by Chapman & Hall/CRC Press LLC

2.3.3 Reasoning with numbers

Now that we have described qualitatively the types of reasoning that are possible using BNs, and types of evidence, let's look at the actual numbers. Even before we obtain any evidence, we can compute a prior belief for the value of each node; this is the node's prior probability distribution. We will use the notation Bel(X) for the posterior probability distribution over a variable X, to distinguish it from the prior and conditional probability distributions (i.e., $P(X)$, $P(X|Y)$).

The exact numbers for the updated beliefs for each of the reasoning cases described above are given in Table 2.2. The first set are for the priors and conditional probabilities originally specified in Figure 2.1. The second set of numbers shows what happens if the smoking rate in the population increases from 30% to 50%, as represented by a change in the prior for the *Smoker* node. Note that, since the two cases differ only in the prior probability of smoking ($P(S = T) = 0.3$ versus $P(S = T) = 0.5$), when the evidence itself is about the patient being a smoker, then the prior becomes irrelevant and both networks give the same numbers.

TABLE 2.2
Updated beliefs given new information with smoking rate 0.3 (top set) and 0.5 (bottom set)

Node	No	Reasoning Case				
P(S)=0.3	Evidence	Diagnostic	Predictive	Intercausal		Combined
		D=T	S=T	C=T	C=T S=T	D=T S=T
Bel(P=high)	0.100	0.102	0.100	0.249	0.156	0.102
Bel(S=T)	0.300	0.307	1	0.825	1	1
Bel(C=T)	0.011	0.025	0.032	1	1	0.067
Bel(X=pos)	0.208	0.217	0.222	0.900	0.900	0.247
Bel(D=T)	0.304	1	0.311	0.650	0.650	1
P(S)=0.5						
Bel(P=high)	0.100	0.102	0.100	0.201	0.156	0.102
Bel(S=T)	0.500	0.508	1	0.917	1	1
Bel(C=T)	0.174	0.037	0.032	1	1	0.067
Bel(X=pos)	0.212	0.226	0.311	0.900	0.900	0.247
Bel(D=T)	0.306	1	0.222	0.650	0.650	1

Belief updating can be done using a number of exact and approximate inference algorithms. We give details of these algorithms in Chapter 3, with particular emphasis on how choosing different algorithms can affect the efficiency of both the knowledge engineering process and the automated reasoning in the deployed system. However, most existing BN software packages use essentially the same algorithm and it is quite possible to build and use BNs without knowing the details of the belief updating algorithms.

© 2004 by Chapman & Hall/CRC Press LLC

2.4 Understanding Bayesian networks

We now consider how to interpret the information encoded in a BN — the probabilistic **semantics** of Bayesian networks.

2.4.1 Representing the joint probability distribution

Most commonly, BNs are considered to be representations of joint probability distributions. There is a fundamental assumption that there is a useful underlying structure to the problem being modeled that can be captured with a BN, i.e., that not every node is connected to every other node. If such domain structure exists, a BN gives a more compact representation than simply describing the probability of every joint instantiation of all variables. **Sparse** Bayesian networks (those with relatively few arcs, which means few parents for each node) represent probability distributions in a computationally tractable way.

Consider a BN containing the n nodes, X_1 to X_n, taken in that order. A particular value in the joint distribution is represented by $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$, or more compactly, $P(x_1, x_2, \ldots, x_n)$. The **chain rule** of probability theory allows us to factorize joint probabilities so:

$$\begin{aligned} P(x_1, x_2, \ldots, x_n) &= P(x_1) \times P(x_2|x_1) \ldots, \times P(x_n|x_1, \ldots, x_{n-1}) \\ &= \prod_i P(x_i|x_1, \ldots, x_{i-1}) \end{aligned} \tag{2.1}$$

Recalling from §2.2.4 that the structure of a BN implies that the value of a particular node is conditional *only* on the values of its parent nodes, this reduces to

$$P(x_1, x_2, \ldots, x_n) = \prod_i P(x_i|Parents(X_i))$$

provided $Parents(X_i) \subseteq \{x_1, \ldots, x_{i-1}\}$. For example, by examining Figure 2.1, we can simplify its joint probability expressions. E.g.,

$$\begin{aligned} &P(X = pos \wedge D = T \wedge C = T \wedge P = low \wedge S = F) \\ &\quad = P(X = pos|D = T, C = T, P = low, S = F) \\ &\qquad \times P(D = T|C = T, P = low, S = F) \\ &\qquad \times P(C = T|P = low, S = F) P(P = low|S = F) P(S = F) \\ &\quad = P(X = pos|C = T) P(D = T|C = T) P(C = T|P = low, S = F) \\ &\qquad \times P(P = low) P(S = F) \end{aligned}$$

2.4.2 Pearl's network construction algorithm

The condition that $Parents(X_i) \subseteq \{x_1, \ldots, x_{i-1}\}$ allows us to construct a network from a given ordering of nodes using Pearl's network construction algorithm [217,

© 2004 by Chapman & Hall/CRC Press LLC

section 3.3]. Furthermore, the resultant network will be a unique minimal I-map, assuming the probability distribution is positive. The construction algorithm (Algorithm 2.1) simply processes each node in order, adding it to the existing network and adding arcs from a minimal set of parents such that the parent set renders the current node conditionally independent of every other node preceding it.

ALGORITHM 2.1
Pearl's Network Construction Algorithm

1. *Choose the set of relevant variables* $\{X_i\}$ *that describe the domain.*
2. *Choose an ordering for the variables,* $< X_1, \ldots, X_n >$.
3. *While there are variables left:*
   (a) *Add the next variable* X_i *to the network.*
   (b) *Add arcs to the* X_i *node from some minimal set of nodes already in the net,* $Parents(X_i)$*, such that the following conditional independence property is satisfied:*

$$P(X_i|X'_1, \ldots, X'_m) = P(X_i|Parents(X_i))$$

   where $X'_1, \ldots, X'_m$ *are all the variables preceding* X_i *that are not in* $Parents(X_i)$.
   (c) *Define the CPT for* X_i.

2.4.3 Compactness and node ordering

Using this construction algorithm, it is clear that a different node order may result in a different network structure, with both nevertheless representing the same joint probability distribution.

In our example, several different orderings will give the original network structure: *Pollution* and *Smoker* must be added first, but in either order, then *Cancer*, and then *Dyspnoea* and *X-ray*, again in either order.

On the other hand, if we add the symptoms first, we will get a markedly different network. Consider the order $< D, X, C, P, S >$. D is now the new root node. When adding *X*, we must consider "Is *X-ray* independent of *Dyspnoea*?" Since they have a common cause in *Cancer*, they will be dependent: learning the presence of one symptom, for example, raises the probability of the other being present. Hence, we have to add an arc from D to X. When adding *Cancer*, we note that *Cancer* is directly dependent upon both *Dyspnoea* and *X-ray*, so we must add arcs from both. For *Pollution*, an arc is required from C to P to carry the direct dependency. When the final node, *Smoker*, is added, not only is an arc required from C to S, but another from P to S. In our story S and P are independent, but in the new network, without this final arc, P and S are made dependent by having a common cause, so that effect must be counterbalanced by an additional arc. The result is two additional arcs and three new probability values associated with them, as shown in Figure 2.3(a). Given the order $< D, X, P, S, C >$, we get Figure 2.3(b), which is

© 2004 by Chapman & Hall/CRC Press LLC

fully connected and requires as many CPT entries as a brute force specification of the full joint distribution! In such cases, the use of Bayesian networks offers no representational, or computational, advantage.

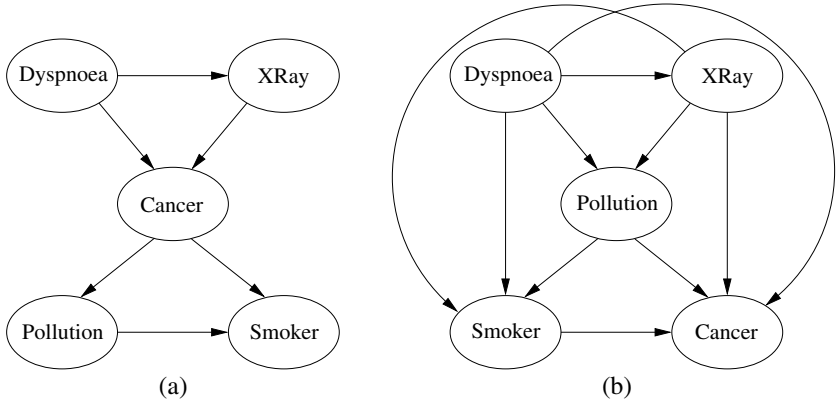


FIGURE 2.3
Alternative structures obtained using Pearl's network construction algorithm with orderings: (a) $< D, X, C, P, S >$; (b) $< D, X, P, S, C >$.

It is desirable to build the most compact BN possible, for three reasons. First, the more compact the model, the more tractable it is. It will have fewer probability values requiring specification; it will occupy less computer memory; probability updates will be more computationally efficient. Second, overly dense networks fail to represent independencies explicitly. And third, overly dense networks fail to represent the *causal* dependencies in the domain. We will discuss these last two points just below.

We can see from the examples that the compactness of the BN depends on getting the node ordering "right." The optimal order is to add the root causes first, then the variable(s) they influence directly, and continue until leaves are reached. To understand *why*, we need to consider the relation between probabilistic and causal dependence.

2.4.4 Conditional independence

Bayesian networks which satisfy the Markov property (and so are I-maps) explicitly express conditional independencies in probability distributions. The relation between conditional independence and Bayesian network structure is important for understanding how BNs work.

© 2004 by Chapman & Hall/CRC Press LLC

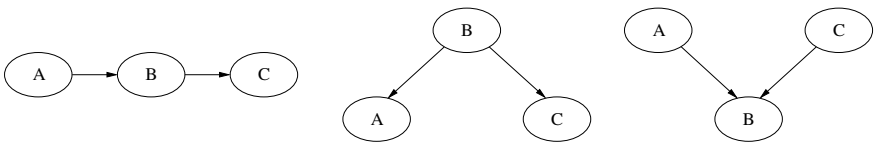
2.4.4.1 Causal chains

Consider a causal chain of three nodes, where A causes B which in turn causes C, as shown in Figure 2.4(a). In our medical diagnosis example, one such causal chain is "smoking causes cancer which causes dyspnoea." Causal chains give rise to conditional independence, such as for Figure 2.4(a):

$$P(C|A \wedge B) = P(C|B)$$

This means that the probability of C, given B, is exactly the same as the probability of C, given both B *and* A. Knowing that A has occurred doesn't make any difference to our beliefs about C *if we already know that B has occurred*. We also write this conditional independence as: $A \perp\!\!\!\perp C|B$.

In Figure 2.1(a), the probability that someone has dyspnoea depends directly only on whether they have cancer. If we don't know whether some woman has cancer, but we do find out she is a smoker, that would increase our belief both that she has cancer and that she suffers from shortness of breath. However, if we already *knew* she had cancer, then her smoking wouldn't make any difference to the probability of dyspnoea. That is, dyspnoea is conditionally independent of being a smoker *given* the patient has cancer.



(a) (b) (c)

FIGURE 2.4
(a) Causal chain; (b) common cause; (c) common effect.

2.4.4.2 Common causes

Two variables A and C having a common cause B is represented in Figure 2.4(b). In our example, cancer is a common cause of the two symptoms, a positive X-ray result and dyspnoea. Common causes (or common ancestors) give rise to the same conditional independence structure as chains:

$$P(C|A \wedge B) = P(C|B) \equiv A \perp\!\!\!\perp C|B$$

If there is no evidence or information about cancer, then learning that one symptom is present will increase the chances of cancer which in turn will increase the probability

© 2004 by Chapman & Hall/CRC Press LLC

of the other symptom. However, if we already know about cancer, then an additional positive X-ray won't tell us anything new about the chances of dyspnoea.

2.4.4.3 Common effects

A common effect is represented by a network v-structure, as in Figure 2.4(c). This represents the situation where a node (the effect) has two causes. Common effects (or their descendants) produce the exact opposite conditional independence structure to that of chains and common causes. That is, the parents are marginally independent ($A \perp\!\!\!\perp C$), but become dependent given information about the common effect (i.e., they are **conditionally dependent**):

$$P(A|C \wedge B) \neq P(A|C) \equiv \neg(A \perp\!\!\!\perp C|B)$$

Thus, if we observe the effect (e.g., cancer), and then, say, we find out that one of the causes is absent (e.g., the patient does not smoke), this *raises* the probability of the other cause (e.g., that he lives in a polluted area) — which is just the inverse of explaining away.

Compactness again

So we can now see *why* building networks with an order violating causal order can, and generally will, lead to additional complexity in the form of extra arcs. Consider just the subnetwork { *Pollution, Smoker, Cancer* } of Figure 2.1. If we build the subnetwork in that order we get the simple v-structure *Pollution* $\rightarrow$ *Smoker* $\leftarrow$ *Cancer*. However, if we build it in the order $<$ *Cancer, Pollution, Smoker* $>$, we will first get *Cancer* $\rightarrow$ *Pollution*, because they are dependent. When we add *Smoker*, it will be dependent upon *Cancer*, because in reality there is a direct dependency there. But we shall also have to add a spurious arc to *Pollution*, because otherwise *Cancer* will act as a common cause, inducing a spurious dependency between *Smoker* and *Pollution*; the extra arc is necessary to reestablish marginal independence between the two.

2.4.5 d-separation

We have seen how Bayesian networks represent conditional independencies and how these independencies affect belief change during updating. The conditional independence in $A \perp\!\!\!\perp C|B$ means that knowing the value of B **blocks** information about C being relevant to A, and vice versa. Or, in the case of Figure 2.4(c), *lack* of information about B blocks the relevance of C to A, whereas learning about B **activates** the relation between C and A.

These concepts apply not only between pairs of nodes, but also between sets of nodes. More generally, given the Markov property, it is possible to determine whether a set of nodes **X** is independent of another set **Y**, given a set of evidence nodes **E**. To do this, we introduce the notion of **d-separation** (from **direction-dependent separation**).

© 2004 by Chapman & Hall/CRC Press LLC

Definition 2.1 Path (Undirected Path) *A* **path** *between two sets of nodes* **X** *and* **Y** *is any sequence of nodes between a member of* **X** *and a member of* **Y** *such that every adjacent pair of nodes is connected by an arc (regardless of direction) and no node appears in the sequence twice.*

Definition 2.2 Blocked path *A path is* **blocked***, given a set of nodes* **E***, if there is a node Z on the path for which at least one of three conditions holds:*

1. *Z is in* **E** *and Z has one arc on the path leading in and one arc out (chain).*
2. *Z is in* **E** *and Z has both path arcs leading out (common cause).*
3. *Neither Z nor any descendant of Z is in* **E***, and both path arcs lead in to Z (common effect).*

Definition 2.3 d-separation *A set of nodes* **E d-separates** *two other sets of nodes* **X** *and* **Y** *if every path from a node in* **X** *to a node in* **Y** *is* **blocked** *given* **E***.*

If **X** and **Y** are **d-separated** by **E**, then **X** and **Y** are **conditionally independent** given **E** (given the Markov property). Examples of these three blocking situations are shown in Figure 2.5. Note that we have simplified by using single nodes rather than sets of nodes; also note that the evidence nodes **E** are shaded.

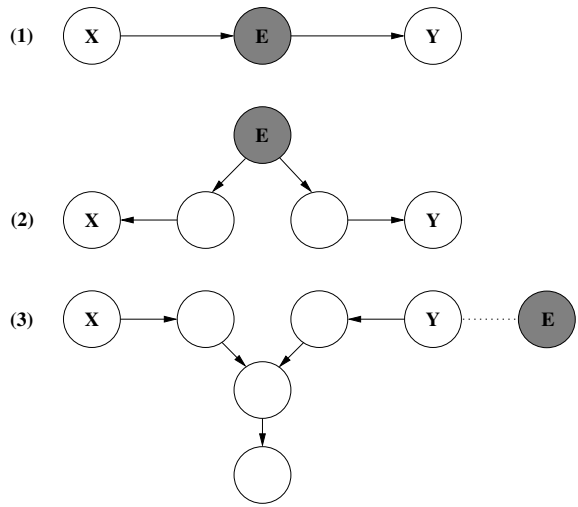


FIGURE 2.5
Examples of the three types of situations in which the path from X to Y can be blocked, given evidence E. In each case, X and Y are **d-separated** by E.

Consider d-separation in our cancer diagnosis example of Figure 2.1. Suppose an observation of the Cancer node is our evidence. Then:

1. P is d-separated from X and D. Likewise, S is d-separated from X and D (blocking condition 1).

© 2004 by Chapman & Hall/CRC Press LLC

2. While X is d-separated from D (condition 2).
3. However, if C had not been observed (and also not X or D), then S would have been d-separated from P (condition 3).

2.5 More examples

In this section we present further simple examples of BN modeling from the literature. We encourage the reader to work through these examples using BN software (see §B.4).

2.5.1 Earthquake

Example statement: *You have a new burglar alarm installed. It reliably detects burglary, but also responds to minor earthquakes. Two neighbors, John and Mary, promise to call the police when they hear the alarm. John always calls when he hears the alarm, but sometimes confuses the alarm with the phone ringing and calls then also. On the other hand, Mary likes loud music and sometimes doesn't hear the alarm. Given evidence about who has and hasn't called, you'd like to estimate the probability of a burglary (from [217]).*

A BN representation of this example is shown in Figure 2.6. All the nodes in this BN are Boolean, representing the true/false alternatives for the corresponding propositions. This BN models the assumptions that John and Mary do not perceive a burglary directly and they do not feel minor earthquakes. There is no explicit representation of loud music preventing Mary from hearing the alarm, nor of John's confusion of alarms and telephones; this information is summarized in the probabilities in the arcs from *Alarm* to *JohnCalls* and *MaryCalls*.

2.5.2 Metastatic cancer

Example statement: *Metastatic cancer is a possible cause of brain tumors and is also an explanation for increased total serum calcium. In turn, either of these could explain a patient falling into a coma. Severe headache is also associated with brain tumors. (This example has a long history in the literature [51, 217, 262].)*

A BN representation of this metastatic cancer example is shown in Figure 2.7. All the nodes are Booleans. Note that this is a *graph*, not a tree, in that there is more than one path between the two nodes M and C (via S and B).

2.5.3 Asia

Example Statement: *Suppose that we wanted to expand our original medical diagnosis example to represent explicitly some other possible causes of shortness of*

© 2004 by Chapman & Hall/CRC Press LLC

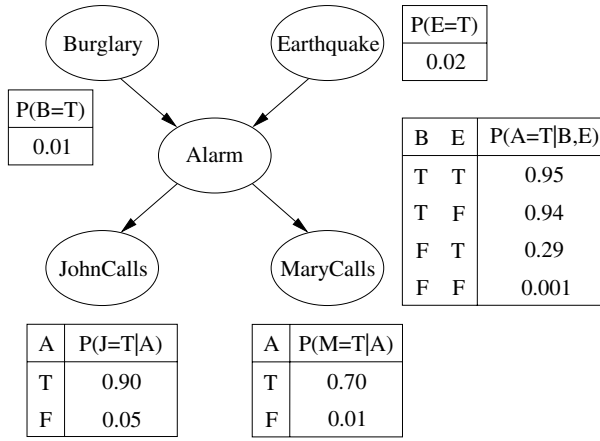


FIGURE 2.6
Pearl's Earthquake BN.

breath, namely tuberculosis and bronchitis. Suppose also that whether the patient has recently visited Asia is also relevant, since TB is more prevalent there.

Two alternative BN structures for the so-called Asia example are shown in Figure 2.8. In both networks all the nodes are Boolean. The left-hand network is based on the Asia network of [169]. Note the slightly odd intermediate node *TBorC*, indicating that the patient has either tuberculosis or bronchitis. This node is not strictly necessary; however it reduces the number of arcs elsewhere, by summarizing the similarities between TB and lung cancer in terms of their relationship to positive X-ray results and dyspnoea. Without this node, as can be seen on the right, there are two parents for *X-ray* and three for *Dyspnoea*, with the same probabilities repeated in different parts of the CPT. The use of such an intermediate node is an example of "divorcing," a model structuring method described in §9.3.2.

2.6 Summary

Bayes' theorem allows us to update the probabilities of variables whose state has not been observed given some set of new observations. Bayesian networks automate this process, allowing reasoning to proceed in any direction across the network of variables. They do this by combining qualitative information about direct dependencies (perhaps causal relations) in arcs and quantitative information about the strengths of those dependencies in conditional probability distributions. Computational speed gains in updating accrue when the network is sparse, allowing d-separation to take advantage of conditional independencies in the domain (so long as the Markov prop-

© 2004 by Chapman & Hall/CRC Press LLC

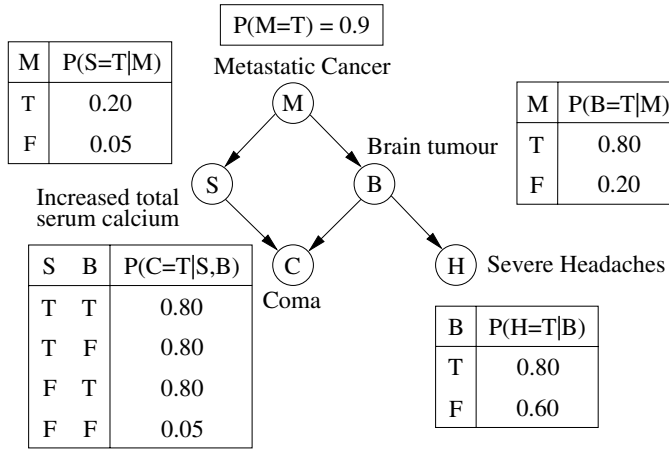


FIGURE 2.7
Metastatic cancer BN.

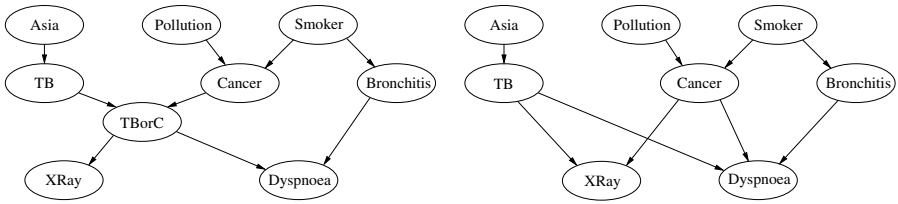


FIGURE 2.8
Alternative BNs for the "Asia" example.

erty holds). Given a known set of conditional independencies, Pearl's network construction algorithm guarantees the development of a minimal network, without redundant arcs. In the next chapter, we turn to specifics about the algorithms used to update Bayesian networks.

2.7 Bibliographic notes

The text that marked the new era of Bayesian methods in artificial intelligence is Judea Pearl's *Probabilistic Reasoning in Intelligent Systems* [217]. This text played no small part in attracting the authors to the field, amongst many others. Richard Neapolitan's *Probabilistic Reasoning in Expert Systems* [199] complements Pearl's book nicely, and it lays out the algorithms underlying the technology particularly well. A more current introduction is Finn Jensen's *Bayesian Networks and Decision*

© 2004 by Chapman & Hall/CRC Press LLC

A Quick Guide to Using Netica

Installation: Web Site `www.norsys.com`. Download Netica, which is available for MS Windows (95 / 98 / NT4 / 2000 / XP), and PowerPC MacIntosh. This gives you `Netica_Win.exe`, a self-extracting zip archive. Double-clicking will start the extraction process.

Network Files: BNs are stored in `.dne` files, with icon  . Netica comes with a folder of example networks, plus a folder of tutorial examples. To open an existing network:

- Select  ; or
- Select `File`→`Open` menu option; or
- Double-click on the BN `.dne` file.

Compilation: Once a Netica BN has been opened, before you can see the initial beliefs or add evidence, you must first compile it:

- Click on  ; or
- Select `Network`→`Compile` menu option.

Once the network is compiled, numbers and bars will appear for each node state. Note that Netica beliefs are given out of 100, not as direct probabilities (i.e., not numbers between 0 and 1).

Evidence: To add evidence:

- Left-click on the node state name; or
- Right-click on node and select particular state name.

To remove evidence:

- Right-click on node and select `unknown`; or
- Select  ; or
- Select `Network`→`Remove findings` menu option.

There is an option (`Network`→`Automatic Update`) to automatically re-compile and update beliefs when new evidence is set.

Editing/Creating a BN: Double-clicking on a node will bring up a window showing node features.

- Add a node by selecting either  , or `Modify`→`Add nature node`, then "drag-and-drop" with the mouse.
- Add an arc by selecting either  , or `Modify`→`Add link`, then left-click first on the parent node, then the child node.
- Double-click on node, then click on the `Table` button to bring up the CPT. Entries can be added or changed by clicking on the particular cells.

Saving a BN: Select  or the `File`→`Save` menu option. Note that the Netica Demonstration version only allows you to save networks with up to 15 nodes. For larger networks, you need to buy a license.

FIGURE 2.9

A quick guide to using Netica.

© 2004 by Chapman & Hall/CRC Press LLC

Graphs [128]. Its level and treatment is similar to ours; however, it does not go far with the machine learning and knowledge engineering issues we treat later. Two more technical discussions can be found in Cowell et al.'s *Probabilistic Networks and Expert Systems* [61] and Richard Neapolitan's *Learning Bayesian Networks* [200].

2.8 Problems

Modeling

These modeling exercises should be done using a BN software package (see **A Quick Guide to Using Netica** in Figure 2.9, or **A Quick Guide to Using Hugin** in Figure 3.14, and also Appendix B).

Also note that various information, including Bayesian network examples in Netica's `.dne` format, can be found at the book Web site:

`http://www.csse.monash.edu.au/bai`

Problem 1

Construct a network in which explaining away operates, for example, incorporating multiple diseases sharing a symptom. Operate and demonstrate the effect of explaining away. *Must* one cause explain away the other? Or, can the network be parameterized so that this doesn't happen?

Problem 2

"Fred's LISP dilemma." *Fred is debugging a LISP program. He just typed an expression to the LISP interpreter and now it will not respond to any further typing. He can't see the visual prompt that usually indicates the interpreter is waiting for further input. As far as Fred knows, there are* only *two situations that could cause the LISP interpreter to stop running: (1) there are problems with the computer hardware; (2) there is a bug in Fred's code. Fred is also running an editor in which he is writing and editing his LISP code; if the hardware is functioning properly, then the text editor should still be running. And if the editor is running, the editor's cursor should be flashing. Additional information is that the hardware is pretty reliable, and is OK about 99% of the time, whereas Fred's LISP code is often buggy, say 40% of the time*[§].

1. Construct a Belief Network to represent and draw inferences about Fred's dilemma.

[§]Based on an example used in Dean, T., Allen, J. and Aloimonos, Y. *Artificial Intelligence Theory and Practice* (Chapter 8), Benjamin/Cumming Publishers, Redwood City, CA. 1995. With permission.

© 2004 by Chapman & Hall/CRC Press LLC

First decide what your domain variables are; these will be your network nodes. Hint: 5 or 6 Boolean variables should be sufficient. Then decide what the causal relationships are between the domain variables and add directed arcs in the network from cause to effect. Finanly, you have to add the conditional probabilities for nodes that have parents, and the prior probabilities for nodes without parents. Use the information about the hardware reliability and how often Fred's code is buggy. Other probabilities haven't been given to you explicitly; choose values that seem reasonable and explain why in your documentation.

2. Show the belief of each variable before adding any evidence, i.e., about the LISP visual prompt not being displayed.
3. Add the evidence about the LISP visual prompt not being displayed. After doing belief updating on the network, what is Fred's belief that he has a bug in his code?
4. Suppose that Fred checks the screen and the editor's cursor is still flashing. What effect does this have on his belief that the LISP interpreter is misbehaving because of a bug in his code? Explain the change in terms of diagnostic and predictive reasoning.

Problem 3

"A Lecturer's Life." *Dr. Ann Nicholson spends 60% of her work time in her office. The rest of her work time is spent elsewhere. When Ann is in her office, half the time her light is off (when she is trying to hide from students and get research done). When she is not in her office, she leaves her light on only 5% of the time. 80% of the time she is in her office, Ann is logged onto the computer. Because she sometimes logs onto the computer from home, 10% of the time she is not in her office, she is still logged onto the computer.*

1. Construct a Bayesian network to represent the "Lecturer's Life" scenario just described.
2. Suppose a student checks Dr. Nicholson's login status and sees that she is logged on. What effect does this have on the student's belief that Dr. Nicholson's light is on?

Problem 4

"Jason the Juggler." *Jason, the robot juggler, drops balls quite often when its battery is low. In previous trials, it has been determined that when its battery is low it will drop the ball 9 times out of 10. On the other hand when its battery is not low, the chance that it drops a ball is much lower, about 1 in 100. The battery was recharged recently, so there is only a 5% chance that the battery is low. Another robot, Olga the observer, reports on whether or not Jason has dropped the ball. Unfortunately Olga's vision system is somewhat unreliable. Based on information from Olga, the*

© 2004 by Chapman & Hall/CRC Press LLC

task is to represent and draw inferences about whether the battery is low depending on how well Jason is juggling[¶].

1. Construct a Bayesian network to represent the problem.
2. Which probability tables show where the information on how Jason's success is related to the battery level, and Olga's observational accuracy, are encoded in the network?
3. Suppose that Olga reports that Jason has dropped the ball. What effect does this have on your belief that the battery is low? What type of reasoning is being done?

Problem 5

Come up with your own problem involving reasoning with evidence and uncertainty. Write down a text description of the problem, then model it using a Bayesian network. Make the problem sufficiently complex that your network has at least 8 nodes and is multiply-connected (i.e., not a tree or a polytree).

1. Show the beliefs for each node in the network before any evidence is added.
2. Which nodes are d-separated with no evidence added?
3. Which nodes in your network would be considered evidence (or observation) nodes? Which might be considered the query nodes? (Obviously this depends on the domain and how you might use the network.)
4. Show how the beliefs change in a form of diagnostic reasoning when evidence about at least one of the domain variables is added. Which nodes are d-separated with this evidence added?
5. Show how the beliefs change in a form of predictive reasoning when evidence about at least one of the domain variables is added. Which nodes are d-separated with this evidence added?
6. Show how the beliefs change through "explaining away" when particular combinations of evidence are added.
7. Show how the beliefs change when you change the priors for a root node (rather than adding evidence).

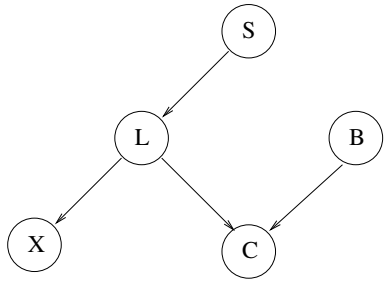
Conditional Independence

Problem 6

Consider the following Bayesian network for another version of the medical diagnosis example, where *B=Bronchitis*, *S=Smoker*, *C=Cough*, *X=Positive X-ray* and *L=Lung cancer* and all nodes are Booleans.

[¶] Variation of Exercise 19.6 in Nilsson, N.J. *Artificial Intelligence: A New Synthesis*, Copyright (1998). With permission from Elsevier.

© 2004 by Chapman & Hall/CRC Press LLC



List the pairs of nodes that are conditionally independent in the following situations:

1. There is no evidence for any of the nodes.
2. The cancer node is set to true (and there is no other evidence).
3. The smoker node is set to true (and there is no other evidence).
4. The cough node is set to true (and there is no other evidence).

Variable Ordering

Problem 7

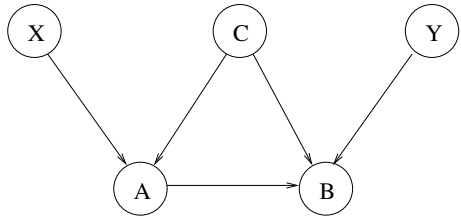
Consider the Bayesian network given for the previous problem.

1. What variable ordering(s) could have been used to produce the above network using the network construction algorithm (Algorithm 2.1)?
2. Given different variable orderings, what network structure would result from this algorithm? Use only pen and paper for now! Compare the number of parameters required by the CPTs for each network.

d-separation

Problem 8

Consider the following graph.



1. Find all the sets of nodes that d-separate X and Y (not including either X or Y in such sets).
2. Try to come up with a real-world scenario that might be modeled with such a network structure.

© 2004 by Chapman & Hall/CRC Press LLC

Problem 9

Design an internal representation for a Bayesian network structure; that is, a representation for the nodes and arcs of a Bayesian network (but not necessarily the parameters — prior probabilities and conditional probability tables). Implement a function which generates such a data structure from the Bayesian network described by a Netica `dne` input file. Use this function in the subsequent problems. (Sample `dne` files are available from the book Web site.)

Problem 10

Implement the network construction algorithm (Algorithm 2.1). Your program should take as input an ordered list of variables and prompt for additional input from the keyboard about the conditional independence of variables as required. It should generate a Bayesian network in the internal representation designed above. It should also print the network in some human-readable form.

Problem 11

Given as input the internal Bayesian network structure N (in the representation you have designed above), write a function which returns all undirected paths (Definition 2.1) between two sets **X** and **Y** of nodes in N.

Test your algorithm on various networks, including at least

- The d-separation network example from Problem 8, `dsepEg.dne`
- `Cancer_Neapolitan.dne`
- `ALARM.dne`

Summarize the results of these experiments.

Problem 12

Given the internal Bayesian network structure N, implement a **d-separation oracle** which, for any three sets of nodes input to it, **X**, **Y**, and **Z**, returns:

- **true** if $X \perp\!\!\!\perp Y | \mathbf{Z}$ (i.e., **Z** d-separates **X** and **Y** in N);
- **false** if $\neg(X \perp\!\!\!\perp Y | \mathbf{Z})$ (i.e., **Z** does not d-separate **X** and **Y** in N);
- some diagnostic (a value other than **true** or **false**) if an error in N is encountered.

Run your algorithm on a set of test networks, including at least the three network specified for Problem 11. Summarize the results of these experiments.

Problem 13

Modify your network construction algorithm from Problem 9 above to use the d-separation oracle from the last problem, instead of input from the user. Your new

© 2004 by Chapman & Hall/CRC Press LLC

algorithm should produce exactly the same network as that used by the oracle whenever the variable ordering provided it is compatible with the oracle's network. Experiment with different variable orderings. Is it possible to generate a network which is simpler than the oracle's network?

© 2004 by Chapman & Hall/CRC Press LLC

3

Inference in Bayesian Networks

3.1 Introduction

The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of query nodes, given values for some evidence nodes. This task is called **belief updating** or **probabilistic inference**. Inference in Bayesian networks is very flexible, as evidence can be entered about any node while beliefs in any other nodes are updated.

In this chapter we will cover the major classes of inference algorithms — exact and approximate — that have been developed over the past 20 years. As we will see, different algorithms are suited to different network structures and performance requirements. Networks that are simple chains merely require repeated application of Bayes' Theorem. Inference in simple tree structures can be done using local computations and message passing between nodes. When pairs of nodes in the BN are connected by multiple paths the inference algorithms become more complex. For some networks, exact inference becomes computationally infeasible, in which case approximate inference algorithms must be used. In general, both exact and approximate inference are theoretically computationally complex (specifically, NP hard). In practice, the speed of inference depends on factors such as the structure of the network, including how highly connected it is, how many undirected loops there are and the locations of evidence and query nodes.

Many inference algorithms have not seen the light of day beyond the research environment that produced them. Good exact and approximate inference algorithms are implemented in BN software, so knowledge engineers do not have to. Hence, our main focus is to characterize the main algorithms' computational performance to both enhance understanding of BN modeling and help the knowledge engineer assess which algorithm is best suited to the application. It is important that the belief updating is not merely a "black-box" process, as there are knowledge engineering issues that can only be resolved through an understanding of the inference process.

We will conclude the chapter with a discussion of how to use Bayesian networks for **causal modeling**, that is for reasoning about the effect of active interventions in the causal process being represented by the network. Such reasoning is important for hypothetical or counterfactual reasoning and for planning and control applications. Unfortunately, current BN tools do not explicitly support causal modeling; however, they can be used for such reasoning and we will describe how to do so.

© 2004 by Chapman & Hall/CRC Press LLC

3.2 Exact inference in chains

3.2.1 Two node network

We begin with the very simplest case, a two node network $X \rightarrow Y$.

If there is evidence about the parent node, say $X = x$, then the posterior probability (or belief) for Y, which we denote $Bel(Y)$, can be read straight from the value in CPT, $P(Y|X = x)$.

If there is evidence about the child node, say $Y = y$, then the inference task of updating the belief for X is done using a simple application of Bayes' Theorem.

$$\begin{aligned} Bel(X = x) &= P(X = x|Y = y) \\ &= \frac{P(Y = y|X = x)P(X = x)}{P(Y = y)} \\ &= \alpha P(x)\lambda(x) \end{aligned}$$

where

$$\alpha = \frac{1}{P(Y = y)}$$

$P(x)$ is the prior and $\lambda(x) = P(Y = y|X = x)$ is the **likelihood**. Note that we don't need to know the prior for the evidence. Since the beliefs for all the values of X must sum to one (due to the Total Probability Theorem 1.2), we can compute α, as a **normalizing constant**.

Example: $Flu \rightarrow HighTemp$

Suppose that we have this very simple model of flu causing a high temperature, with prior $P(Flu = T) = 0.05$, and CPT values $P(HighTemp = T|Flu = T) = 0.9$, $P(HighTemp = T|Flu = F) = 0.2$. If a person has a high temperature (i.e., the evidence available is $HighTemp = T$), the computation for this diagnostic reasoning is as follows.

$$\begin{aligned} Bel(Flu = T) &= \alpha P(Flu = T)\lambda(Flu = T) \\ &= \alpha \times 0.05 \times 0.9 \\ &= \alpha 0.045 \\ Bel(Flu = F) &= \alpha P(Flu = F)\lambda(Flu = F) \\ &= \alpha \times 0.95 \times 0.2 \\ &= \alpha 0.19 \end{aligned}$$

We can compute α via

$$Bel(Flu = T) + Bel(Flu = F) = 1 = \alpha 0.045 + \alpha 0.19$$

© 2004 by Chapman & Hall/CRC Press LLC

giving

$$\alpha = \frac{1}{0.19 + 0.045}$$

This allows us to finish the belief update:

$$Bel(Flu = T) = \frac{0.045}{0.19 + 0.045} = 0.8085$$
$$Bel(Flu = F) = \frac{0.19}{0.19 + 0.045} = 0.1915$$

3.2.2 Three node chain

We can apply the same method when we have three nodes in a chain, $X \rightarrow Y \rightarrow Z$.

If we have evidence about the root node, $X = x$, updating in the same direction as the arcs involves the simple application of the chain rule (Theorem 1.3), using the independencies implied in the network.

$$Bel(Z) = P(Z|X = x) = \sum_{Y=y} P(Z|Y)P(Y|X = x)$$

If we have evidence about the leaf node, $Z = z$, the diagnostic inference to obtain $Bel(X)$ is done with the application of Bayes' Theorem and the chain rule.

$$\begin{aligned} Bel(X = x) &= P(X = x|Z = z) \\ &= \frac{P(Z = z|X = x)P(X = x)}{P(Z = z)} \\ &= \frac{\sum_{Y=y} P(Z = z|Y = y, X = x)P(Y = y|X = x)P(X = x)}{P(Z = z)} \\ &= \frac{\sum_{Y=y} P(Z = z|Y = y)P(Y = y|X = x)P(X = x)}{P(Z = z)} \quad (Z \perp\!\!\!\perp X|Y) \\ &= \alpha P(x)\lambda(x) \end{aligned}$$

where

$$\lambda(x) = P(Z = z|X = x) = \sum_{Y=y} P(Z = z|Y = y)P(Y = y|X = x)$$

Example: $Flu \rightarrow HighTemp \rightarrow HighTherm$

Here our flu example is extended by having an observation node $HighTherm$ representing the reading given by a thermometer. The possible inaccuracy in the thermometer reading is represented by the following CPT entries:

- $P(HighTherm = T|HighTemp = T) = 0.95$ (so 5% chance of a false negative reading)
- $P(HighTherm = T|HighTemp = F) = 0.15$ (15% chance of a false positive reading).

© 2004 by Chapman & Hall/CRC Press LLC

Suppose that a high thermometer reading is taken, i.e., the evidence is $HighTherm = T$.

$$\begin{aligned}
\lambda(Flu = T) &= P(HighTherm = T|HighTemp = T)P(HighTemp = T|Flu = T) \\
&\quad + P(HighTherm = T|HighTemp = F)P(HighTemp = F|Flu = T) \\
&= 0.95 \times 0.9 + 0.15 \times 0.1 \\
&= 0.855 + 0.015 = 0.87 \\
Bel(Flu = T) &= \alpha P(Flu = T)\lambda(Flu = T) \\
&= \alpha 0.05 \times 0.87 = \alpha 0.0435 \\
\lambda(Flu = F) &= P(HighTherm = T|HighTemp = T)P(HighTemp = T|Flu = F) \\
&\quad + P(HighTherm = T|HighTemp = F)P(HighTemp = F|Flu = F) \\
&= 0.95 \times 0.2 + 0.15 \times 0.8 \\
&= 0.19 + 0.12 = 0.31 \\
Bel(Flu = F) &= \alpha P(Flu = F)\lambda(Flu = F) \\
&= \alpha 0.95 \times 0.31 = \alpha 0.2945
\end{aligned}$$

So,

$$\alpha = \frac{1}{0.0435 + 0.2945}$$

hence

$$\begin{aligned}
Bel(Flu = T) &= \frac{0.0435}{0.0435 + 0.2945} = 0.1287 \\
Bel(Flu = F) &= \frac{0.2945}{0.0435 + 0.2945} = 0.8713 \ (\ = 1\ -\ Bel(Flu = T))
\end{aligned}$$

Clearly, inference in chains is straightforward, using application of Bayes' Theorem and the conditional independence assumptions represented in the chain. We will see that these are also the fundamental components of inference algorithms for more complex structures.

3.3 Exact inference in polytrees

Next we will look at how inference can be performed when the network is a simple structure called a **polytree** (or "**forest**"). Polytrees have at most one path between any pair of nodes; hence they are also referred to as **singly-connected** networks.

Assume X is the query node, and there is some set of evidence nodes $\mathbf{E}$ (not including X). The task is to update $Bel(X)$ by computing $P(X|\mathbf{E})$.

Figure 3.1 shows a diagram of a node X in a generic polytree, with all its connections to parents (the U_i), children (the Y_j), and the children's other parents (the Z_{ij}). The local belief updating for X must incorporate evidence from all other parts of the network. From this diagram we can see that evidence can be divided into:

© 2004 by Chapman & Hall/CRC Press LLC

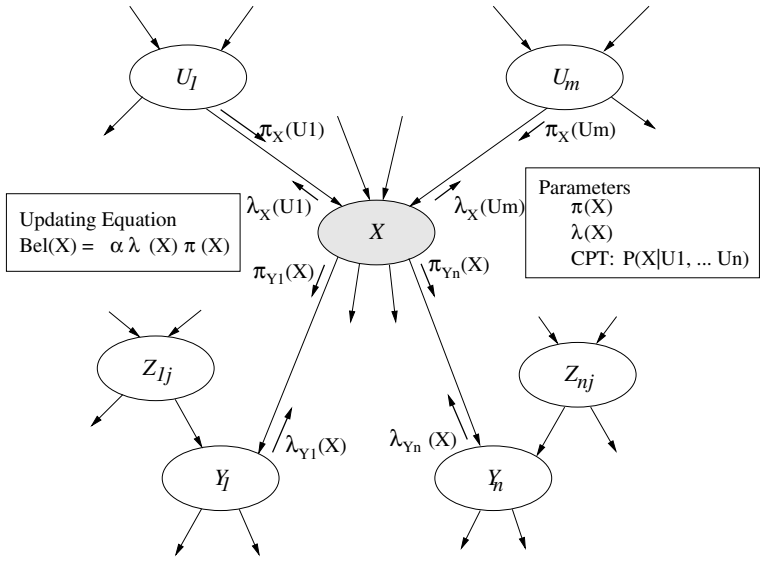


FIGURE 3.1
A generic polytree showing how local belief updating of node X is achieved through incorporation of evidence through its parents (the U_i) and children (the Y_j). Also shown are the message passing parameters and messages.

- The **predictive support** for X, from evidence nodes connected to X through its parents, $U_1, \ldots U_m$; and

- The **diagnostic support** for X, from evidence nodes connected to X through its children $Y_1, \ldots Y_n$.

3.3.1 Kim and Pearl's message passing algorithm

A "bare bones" description of Kim and Pearl's message passing algorithm appears below as Algorithm 3.1. The derivation of the major steps is beyond the scope of this text; suffice it to say, it involves the repeated application of Bayes' Theorem and use of the conditional independencies encoded in the network structure. Those details can be found elsewhere (e.g., [217, 237, 200]). Instead, we will present the major features of the algorithm, illustrating them by working through a simple example.

The parameters and messages used in this algorithm are also shown in Figure 3.1. The basic idea is that at each iteration of the algorithm, $Bel(X)$ is updated locally using three types of parameters — $\lambda(X)$, $\pi(X)$ and the CPTs (equation (3.1)). $\lambda(X)$ and $\pi(X)$ are computed using the π and λ messages received from X's parents and children respectively. π and λ messages are also sent out from X so that its neighbors can perform updates. Let's look at some of the computations more closely.

© 2004 by Chapman & Hall/CRC Press LLC

ALGORITHM 3.1

Kim and Pearl's Message Passing Algorithm

This algorithm requires the following three types of parameters to be maintained.

- *The current strength of the predictive support π contributed by each incoming link $U_i \rightarrow X$:*

  $\pi_X(U_i) = P(U_i|E_{U_i \setminus X})$

  where $E_{U_i \setminus X}$ is all evidence connected to U_i except via X.
- *The current strength of the diagnostic support λ contributed by each outgoing link $X \rightarrow Y_j$:*

  $\lambda_{Y_j}(X) = P(E_{Y_j \setminus X}|X)$

  where $E_{Y_j \setminus X}$ is all evidence connected to Y_j through its parents except via X.
- *The fixed CPT $P(X|U_i, \ldots, U_n)$ (relating X to its immediate parents).*

These parameters are used to do local belief updating in the following three steps, which can be done in any order.

(Note: in this algorithm, x_i means the ith state of node X, while $u_1 \ldots u_n$ is used to represent an instantiation of the parents of X, $U_1 \ldots U_n$, in the situations where there is a summation of all possible instantiations.)

1. **Belief updating**.

   Belief updating for a node X is activated by messages arriving from either children or parent nodes, indicating changes in their belief parameters.

   When node X is activated, inspect $\pi_X(U_i)$ (messages from parents), $\lambda_{Y_j}(X)$ (messages from children). Apply with

   $$Bel(x_i) = \alpha \lambda(x_i) \pi(x_i) \tag{3.1}$$

   where,

   $$\lambda(x_i) = \begin{cases} 1 & \text{if evidence is } X = x_i \\ 0 & \text{if evidence is for another } x_j \\ \prod_j \lambda_{Y_j}(x_i) & \text{otherwise} \end{cases} \tag{3.2}$$

   $$\pi(x_i) = \sum_{u_1, \ldots, u_n} P(x_i|u_1, \ldots, u_n) \prod_i \pi_X(u_i) \tag{3.3}$$

   and α is a normalizing constant rendering $\sum_{x_i} Bel(X = x_i) = 1$.
2. **Bottom-up propagation**.

   Node X computes new λ messages to send to its parents.

   $$\lambda_X(u_i) = \sum_{x_i} \lambda(x_i) \sum_{u_k : k \neq i} P(x_i|u_1, \ldots, u_n) \prod_{k \neq i} \pi_X(u_k) \tag{3.4}$$

© 2004 by Chapman & Hall/CRC Press LLC

3. **Top-down propagation**.

Node X computes new π messages to send to its children.

$$\pi_{Y_j}(x_i) = \begin{cases} 1 & \textit{if evidence value } x_i \textit{ is entered} \\ 0 & \textit{if evidence is for another value } x_j \\ \alpha[\prod_{k \neq j} \lambda_{Y_k}(x_i)] \sum_{u_1,\ldots,u_n} P(x_i|u_1,\ldots,u_n) \prod_i \pi_X(u_i) & \\ = \frac{\alpha Bel(x_i)}{\lambda_{Y_j}(x_i)} & \end{cases} \tag{3.5}$$

First, equation (3.2) shows how to compute the $\lambda(x_i)$ parameter. Evidence is entered through this parameter, so it is 1 if x_i is the evidence value, 0 if the evidence is for some other value x_j, and is the product of all the λ messages received from its children if there is no evidence entered for X. The $\pi(x_i)$ parameter (3.3) is the product of the CPT and the π messages from parents.

The λ message to one parent combines (i) information that has come from children via λ messages and been summarized in the $\lambda(X)$ parameter, (ii) the values in the CPT and (iii) any π messages that have been received from any other parents.

The $\pi_{Y_j}(x_i)$ message down to child Y_j is 1 if x_i is the evidence value and 0 if the evidence is for some other value x_j. If no evidence is entered for X, then it combines (i) information from children other than Y_j, (ii) the CPT and (iii) the π messages it has received from its parents.

The algorithm requires the following initializations (i.e., before any evidence is entered).

- Set all λ values, λ messages and π messages to 1.
- Root nodes: If node W has no parents, set $\pi(W)$ to the prior, $P(W)$.

The message passing algorithm can be used to compute the beliefs for all nodes in the network, even before any evidence is available.

When specific evidence $W = w_i$ is obtained, given that node W can take values $\{w_1, w_2, \ldots, w_n\}$,

- Set $\lambda(W)$ = (0, 0, …, 0, 1, 0, …, 0) with the 1 at the ith position.

This π/λ notation used for the messages is that introduced by Kim and Pearl and can appear confusing at first. Note that the format for both types of messages is $\pi_{Child}(Parent)$ and $\lambda_{Child}(Parent)$. So,

- π messages are sent in the direction of the arc, from parent to child, hence the notation is $\pi_{Receiver}(Sender)$;
- λ messages are sent from child to parent, against the direction of the arc, hence the notation is $\lambda_{Sender}(Receiver)$.

Note also that π plays the role of prior and λ the likelihood in Bayes' Theorem.

© 2004 by Chapman & Hall/CRC Press LLC

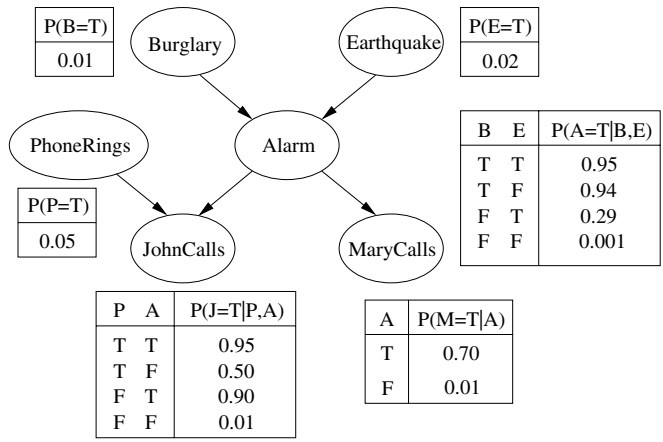


FIGURE 3.2
Extended earthquake BN.

3.3.2 Message passing example

Here, we extend the BN given in Figure 2.6 for Pearl's earthquake problem, by adding a node *PhoneRings* to represent explicitly the phone ringing that John sometimes confuses with the alarm. This extended BN is shown in Figure 3.2, with the parameters and messages used in the message passing algorithm shown in Figure 3.3.

We will work through the message passing updating for the extended earthquake problem, first without evidence, then with evidence entered for node M. The data propagation messages are shown in Figure 3.4. The updating and propagation sequencing presented here are not necessarily those that a particular algorithm would produce, but rather the most efficient sequencing to do the belief updating in the minimum number of steps.

First, before any evidence is entered, various parameters are initialized: $\pi(B)$, $\pi(E)$, $\pi(P)$, from the priors, and $\lambda(J)$, $\lambda(M)$ with (1,1) as leaf nodes without evidence.

During this propagation before evidence, all of the diagnostic messages (the λ messages) will be the unit vector. We can see from the updating equations that we only multiply by these messages, so multiplying by the unit vector will not change any other parameters or messages. So we will not show the unit vector λ message propagation at this stage.

Using the belief updating equation (3.1), $Bel(B)$, $Bel(E)$ and $Bel(P)$ are computed, while sending new messages $\pi_A(B)$, $\pi_A(E)$ and $\pi_J(P)$ are computed using (3.5) and sent out.

Node A has received all its π messages from its parents, so it can update $Bel(A)$ and compute its own π messages to pass down to its children J and M. At this stage, we *could* update $Bel(J)$, as it has just received a π message from P; however, it has yet to receive a π message from A, so we won't do that this cycle.

© 2004 by Chapman & Hall/CRC Press LLC

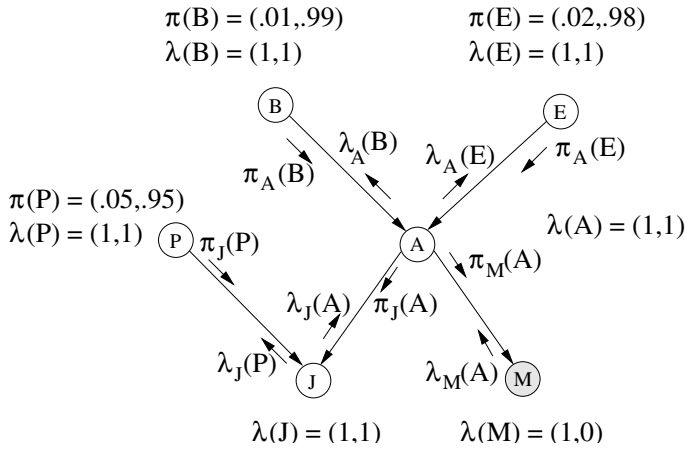


FIGURE 3.3
Extended earthquake BN showing parameter initialization and π and λ messages for Kim and Pearl's message passing algorithm.

After the second message propagation phase, $Bel(J)$ and $Bel(M)$ can be computed, as all π messages have been received from their parents.

Next, we look at how evidence $M = T$, entered by setting $\lambda(M) = (1, 0)$, can be propagated through the network. First, the message $\lambda_M(A)$ is computed and sent up to A. $\lambda(A)$ and in turn $Bel(A)$ are then recomputed, and new messages sent to A's parents via $\lambda_A(B)$ and $\lambda_A(E)$, and to its other child J via $\pi_J(A)$.

These new messages allow $Bel(B)$, $Bel(E)$ and $Bel(J)$ to be recomputed, and the final message $\lambda_J(P)$ computed and sent from J to P. The last computation is the updating of $Bel(P)$. Note that the minimum number of propagation steps for this evidence example is three, since that is the distance of the furthest node P from the evidence node M.

3.3.3 Algorithm features

All the computations in the message passing algorithm are local: the belief updating and new outgoing messages are all computed using incoming messages and the parameters. While this algorithm is efficient in a sense because of this locality property, and it lends itself to parallel, distributed implementations, we can see that there is a summation over all joint instantiations of the parent nodes, which is exponential in the number of parents. Thus, the algorithm is computationally infeasible when there are too many parents. And the longer the paths from the evidence node or nodes, the more cycles of data propagation must be performed to update all other nodes.

Note also that in the presentation of this algorithm, we have followed Pearl's presentation [217] and normalized all the messages. This is a computational overhead that is not strictly necessary, as all the normalizing can be done when computing the

© 2004 by Chapman & Hall/CRC Press LLC

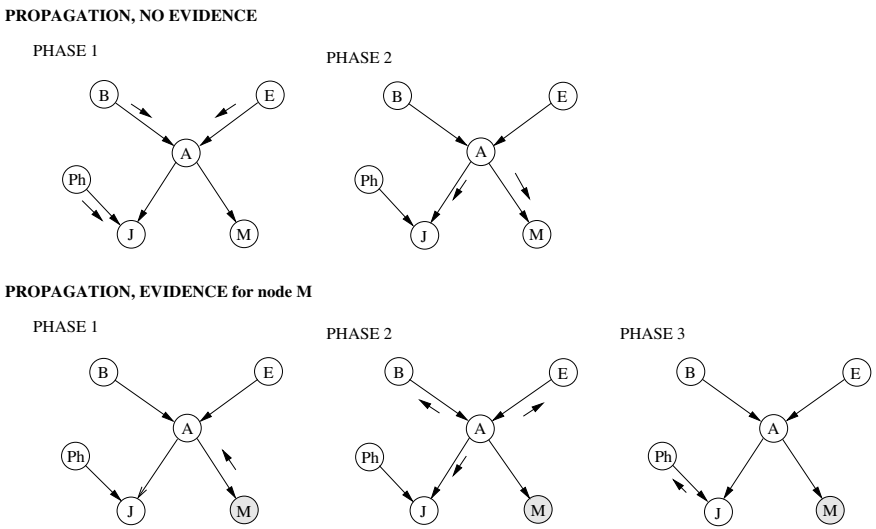


FIGURE 3.4
Message passing algorithm propagation stages for without evidence (above) and with evidence for node M (below).

marginals (i.e., when computing $Bel(x_i)$). The normalization constants, α, are the same for all the marginals, being the inverse of the probability P(**E**), the computation of which is often useful for other purposes.

3.4 Inference with uncertain evidence

Thus far we have assumed that any evidence is a direct observation of the value of a variable that will result in the belief for a node being set to 1 for that value and 0 for all other values. This is the specific evidence described in §2.3.2, which is entered in the message passing algorithm as a vector with a 1 in the position of the evidence value and 0 in the other positions. Once specific evidence is entered for a node, the belief for that node is "clamped" and doesn't change no matter what further information becomes available.

However, the inference algorithms should also be able to handle evidence that has uncertainty associated with it. In our earthquake example, suppose that after you get a call from your neighbor Mary saying she has heard your alarm going off, a colleague who is in your office at the time says he thinks he heard earlier on the radio that there was a minor earthquake in your area, but he is only 80% sure.

We introduced this notion very briefly in §2.3.2, where we mentioned that this sort of evidence is called "virtual" evidence or "likelihood" evidence. We deferred further

© 2004 by Chapman & Hall/CRC Press LLC

explanation of these terms until here, as they relate to how inference is performed.

Some of the major BN software packages (e.g., Netica and Hugin) provide the facility for adding likelihood evidence, as well as specific and negative evidence. In fact, we describe how to enter likelihood evidence in both Netica and Hugin in Figure 3.15. In our opinion, the explanations of this sort of evidence in both the literature and available software documentation are confusing and incomplete. It is important for people using this feature in the software to understand how likelihood evidence affects the inference and also how to work out the numbers to enter.

First, there is the issue of how to interpret uncertainty about an observation. The uncertain information could be represented by adopting it as the new distribution over the variable in question. This would mean for the earthquake node somehow setting $Bel(Earthquake = T) = 0.8$. However, we certainly do not want to "clamp" this belief, since this probabilistic judgement should be integrated with any further independent information relevant to the presence of an earthquake (e.g., the evidence about Mary calling).

3.4.1 Using a virtual node

Let's look at incorporating an uncertain observation for the simplest case, a single Boolean node *X*, with a uniform prior, that is P(X=T)=P(X=F)=0.5. We add a **virtual node** V, which takes values $\{T, F\}$,* as a child of *X*, as shown in Figure 3.5. The uncertainty in the observation of X is represented by the CPT; for an 80% sure observation this gives $P(V = T|X = T) = 0.8$ and $P(V = T|X = F) = 0.2$†. Now specific evidence is entered that $V = T$. We can use Bayes' Theorem to perform the inference, as follows.

$$\begin{aligned} Bel(X = T) &= \alpha P(V = T|X = T)P(X = T) \\ &= \alpha 0.8 \times 0.5 \\ Bel(X = F) &= \alpha P(V = T|X = T)P(X = F) \\ &= \alpha 0.2 \times 0.5 \end{aligned}$$

Since $Bel(X = T) + Bel(X = F) = 1$, this gives us $\alpha = 2$, and hence $Bel(X = T) = 0.8$ and $Bel(X = F) = 0.2$, as desired.

It is important to note here that due to the normalization, it is not the likelihoods for $P(V = T|X = T)$ and $P(V = T|X = F)$ that determine the new belief, but rather the **ratio**:

$$P(V = T|X = T) : P(V = T|X = F)$$

For example, we would get the same answers for $P(V = T|X) = (0.4, 0.1)$ or $P(V = T|X) = (0.5, 0.125)$.

*Note that V takes these values *irrespective* of the values of the observed node.

†Note that the semantics of this CPT are not well specified. For example, what is the meaning of the CPT entries for $P(V = F|X)$?

© 2004 by Chapman & Hall/CRC Press LLC

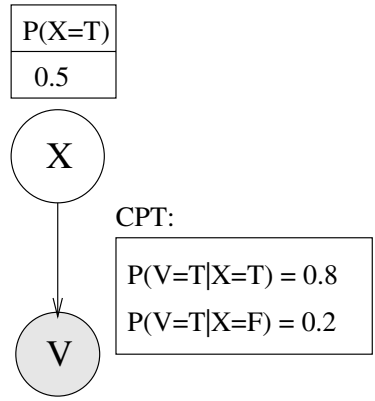


FIGURE 3.5
Virtual evidence handled through virtual node V for single node X, with uncertainty in CPT.

So far, we have seen that with the use of a "virtual" node we can represent uncertainty in an observation by a likelihood ratio. For example, my colleague attributing 80% credibility to there having been an earthquake means that he is four times more likely to think an earthquake occurred if that was in fact the case than he is of mistakenly thinking it occurred.

But we have considered only the situation of a node without parents, with uniform priors. If the priors are *not* uniform, then simply mapping the uncertainty into a likelihood ratio as described does not give a new belief that correctly represents the specified observational uncertainty. For example, in our earthquake example, if $P(E) = (0.02, 0.98)$, then

$$\begin{aligned} Bel(E = T) &= \alpha P(V = T|E = T)P(E = T) \\ &= \alpha 0.8 \times 0.02 \\ Bel(E = F) &= \alpha P(V = T|E = T)P(E = F) \\ &= \alpha 0.2 \times 0.98 \end{aligned}$$

which gives $\alpha \approx 4.72$, and $Bel(E = T) = 0.075$ and $Bel(E = F) = 0.925$ so the posterior belief in *Earthquake* is only 7.5%. However, if $P(E) = (0.9, 0.1)$, then

$$\begin{aligned} Bel(E = T) &= \alpha P(V = T|E = T)P(E = T) \\ &= \alpha 0.8 \times 0.9 \\ Bel(E = F) &= \alpha P(V = T|E = T)P(E = F) \\ &= \alpha 0.2 \times 0.1 \end{aligned}$$

which gives us $Bel(E = T) = 0.973$ and $Bel(E = F) = 0.027$.

© 2004 by Chapman & Hall/CRC Press LLC

It is clear that directly mapping the observational uncertainty into a likelihood ratio only results in the identical belief for the node in question (as intended in Jeffrey conditionalization) when the priors are uniform. When the priors are non-uniform, a likelihood ratio of 4:1 will increase the belief, but not necessarily to the intended degree.

Some people feel this is an advantage of this representation, supposedly indicating that the observation is an external "opinion" from someone whose priors are unknown. However, if we wish to represent unknown priors, we have no business inferring a posterior at all!

If we *really* want the uncertain evidence to shift the beliefs $Bel(X = T) = P(X = T|V) = 0.8$, then we can still use a virtual node and likelihood ratio approach, but we need to compute the ratio properly. Let's return to our earthquake example. Recalling Definition 1.6, $P(E|V) = 0.8$ means $O(E = T|V) = 4$. Since (by odds-likelihood Bayes' theorem 1.5)

$$O(E = T|V) = \frac{P(V = T|E = T)}{P(V = T|E = F)} O(E = T)$$

and, as given in the example,

$$O(E = T) = 0.02/0.98$$

we get the likelihood ratio

$$\frac{P(V = T|E = T)}{P(V = T|E = F)} = \frac{0.8}{0.2} \times \frac{0.98}{0.02} = 196$$

This ratio is *much* higher than the one used with uniform priors, which is necessary in order to shift the belief in *Earthquake* from its very low prior of 0.02 to 0.8.

3.4.2 Virtual nodes in the message passing algorithm

When implementing a message passing inference algorithm, we don't actually need to add the V node. Instead, the virtual node is connected by a virtual link as a child to the node it is "observing." In the message passing algorithm, these links only carry information one way, from the virtual node to the observed variable. For our earthquake example, the virtual node V represents the virtual evidence on E. There are no parameters $\lambda(V)$, but instead it sends a $\lambda_V(E)$ message to E. The virtual node and the message it sends to E is shown in Figure 3.6.

© 2004 by Chapman & Hall/CRC Press LLC

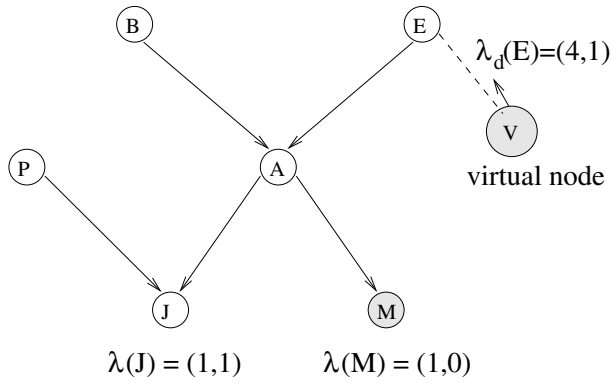


FIGURE 3.6
Virtual evidence handled through virtual node V for earthquake node E, with λ message in message passing algorithm.

3.5 Exact inference in multiply-connected networks

In the most general case, the BN structure is a (directed acyclic) graph, rather than simply a tree. This means that at least two nodes are connected by more than one path in the underlying undirected graph. Such a network is **multiply-connected** and occurs when some variable can influence another through more than one causal mechanism.

The metastatic cancer network shown in Figure 3.7 is an example. The two causes of C (S and B) share a common parent, M. For this BN, there is an undirected loop around nodes M, B, C and S. In such networks, the message passing algorithm for polytrees presented in the previous section does not work. Intuitively, the reason is that with more than one path between two nodes, the same piece of evidence about one will reach the other through two paths and be counted twice. There are many ways of dealing with this problem in an exact manner; we shall present the most popular of these.

3.5.1 Clustering methods

Clustering inference algorithms transform the BN into a probabilistically equivalent polytree by merging nodes, removing the multiple paths between the two nodes along which evidence may travel. In the cancer example, this transformation can be done by creating a new node, say Z, that combines nodes B and S, as shown in Figure 3.8. The new node has four possible values, $\{TT, TF, FT, FF\}$, corresponding to all possible instantiations of B and S. The CPTs for the transformed network are also shown in Figure 3.8. They are computed from the CPTs of the original graph as

© 2004 by Chapman & Hall/CRC Press LLC

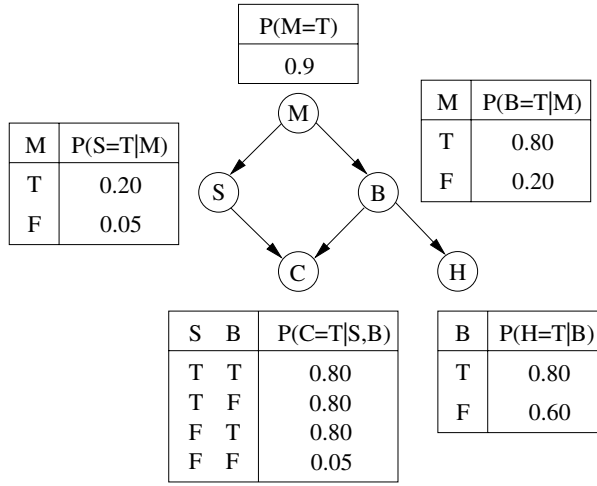


FIGURE 3.7
Example of a multiply-connected network: metastatic cancer BN (§2.5.2).

follows.

$$\begin{aligned} P(Z|M) &= P(S,B|M) \text{ by definition of Z} \\ &= P(S|M)P(B|M) \text{ since S and B are independent given M} \end{aligned}$$

Similarly,

$$\begin{aligned} P(H|Z) &= P(H|S,B) = P(H|B) \text{ since H is independent of S given B} \\ P(C|Z) &= P(C|S,B) \text{ by definition of Z} \end{aligned}$$

It is always possible to transform a multiply-connected network into a polytree. In the extreme case, all the non-leaf nodes can be merged into a single compound node, as shown in Figure 3.9(a) where all the nodes D_i are merged into a single super-node. The result is a simple tree. But other transformations are also possible, as shown in Figure 3.9(b) and (c), where two different polytrees are produced by different clusterings. It is better to have smaller clusters, since the CPT size for the cluster grows exponentially in the number of nodes merged into it. However the more highly connected the original network, the larger the clusters required.

Exact clustering algorithms perform inference in two stages.

1. Transform the network into a polytree
2. Perform belief updating on that polytree

The transformation in Step 1 may be slow, since a large number of new CPT values may need to be computed. It may also require too much memory if the original network is highly connected. However, the transformation only needs to be done once, unless the structure or the parameters of the original network are changed.

© 2004 by Chapman & Hall/CRC Press LLC

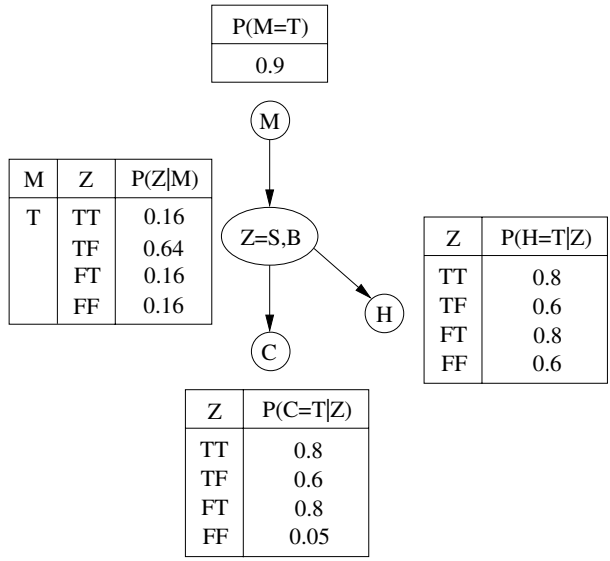


FIGURE 3.8
Result of ad hoc clustering of the metastatic cancer BN.

The belief updating in the new polytree is usually quite fast, as an efficient polytree message passing algorithm may now be applied. The caveat does need to be added, however, that since clustered nodes have increased complexity, this can slow down updating as well.

3.5.2 Junction trees

The clustering we've seen thus far has been ad hoc. The junction tree algorithm provides a methodical and efficient method of clustering, versions of which are implemented in the main BN software packages (see §B.4).

ALGORITHM 3.2
The Junction Tree Clustering Algorithm

1. **Moralize:** *Connect all parents and remove arrows; this produces a so-called* **moral graph**.
2. **Triangulate:** *Add arcs so that every cycle of length > 3 has a chord (i.e., so there is a subcycle composed of exactly three of its nodes); this produces a* **triangulated graph**.
3. **Create new structure:** *Identify maximal cliques in the triangulated graph to become new compound nodes, then connect to form the so-called* **junction tree**.
4. **Create separators:** *Each arc on the junction tree has an attached* **separator**, *which consists of the intersection of adjacent nodes.*

© 2004 by Chapman & Hall/CRC Press LLC

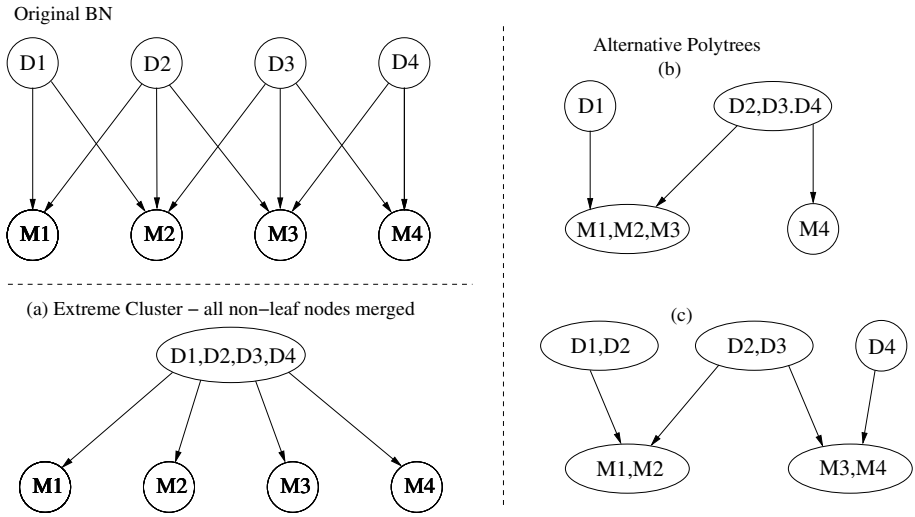


FIGURE 3.9
The original multiply-connected BN can be clustered into: (a) a tree; different polytrees (b) and (c).

5. **Compute new parameters:** *Each node and separator in the junction tree has an associated table over the configurations of its constituent variables. These are all a table of ones to start with.*

   For *each node* X *in the original network,*

   (a) Choose one node Y *in the junction tree that contains* X *and all of* X*'s parents,*
   (b) Multiply $P(X|Parents(X))$ *on* Y*'s table.*

6. **Belief updating:** *Evidence is added and propagated using a message passing algorithm.*

The moralizing step (Step 1) is so called because an arc is added if there is no direct connection between two nodes with a common child; it is "marrying" the parent nodes. For our metastatic cancer example, S and B have C as a common child; however, there is no direct connection between them, so this must be added. The resultant moral graph is shown in Figure 3.10(a).

As we have seen, the size of the CPT for a new compound node is the product of the number of states of the combined nodes, so the size of the CPT increases exponentially with the size of the compound node. Different triangulations (Step 2) produce different clusters. Although the problem of finding an optimal triangulation is NP complete, there are heuristics which give good results in practice. No arcs need to be added to the moral graph for the metastatic cancer example, as it is already triangulated.

© 2004 by Chapman & Hall/CRC Press LLC

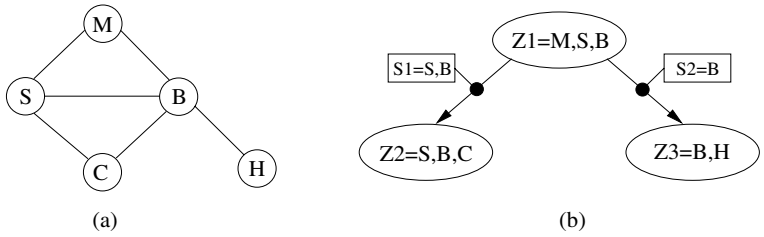


FIGURE 3.10
Applying the junction tree algorithm to the metastatic cancer BN gives (a) the moral graph and (b) the junction tree.

A **clique** of an undirected graph is defined as a set of nodes that are all pairwise linked; that is, for every pair of nodes in the set, there is an arc between them. A **maximal clique** is such a subgraph that cannot be increased by adding any node. In our example the maximal cliques can be read from the moral graph, giving three new compound nodes, $Z_1 = M, S, B$, $Z_2 = S, B, C$ and $Z_3 = B, H$ (Step 3). The junction tree, including the separators (Step 4), is shown in Figure 3.10(b).

Note that sometimes after the compound nodes have been constructed and connected, a junction *graph* results, instead of a junction tree. However, because of the triangulation step, it is always possible to remove links that aren't required in order to form a tree. These links aren't required because the same information can be passed along another path; see [129] for details.

The metastatic cancer network we've been using as an example is not sufficiently complex to illustrate all the steps in the junction tree algorithm. Figure 3.11 shows the transformation of a more complex network into a junction tree (from [129]).

Finally, Step 5 computes the new CPTs for the junction tree compound nodes. The result of Step 5 is that the product of all the node CPTs in the cluster tree is the product of all the CPTs in the original BN. Any cluster tree obtained using this algorithm is a representation of the same joint distribution, which is the product of all the cluster tables divided by the product of all the separator tables.

Adding evidence to the junction tree is simple. Suppose that there is evidence e about node X. If it is specific evidence that $X = x_i$, then we create an evidence vector with a 1 in the ith position, zeros in all the others. If it is a negative finding for state x_j, there is a zero in the jth position and ones in all the others. If it is virtual evidence, the vector is constructed as in §3.4. In all cases, the evidence vector is multiplied on the table of any node in the junction tree containing X.

Once the junction tree has been constructed (called "**compilation**" in most BN software, including Netica) and evidence entered, belief updating is performed using a message passing approach. The details can be found in [128]. The basic idea is that separators are used to receive and pass on messages, and the computations involve computing products of CPTs.

The cost of belief updating using this junction tree approach is primarily deter-

© 2004 by Chapman & Hall/CRC Press LLC

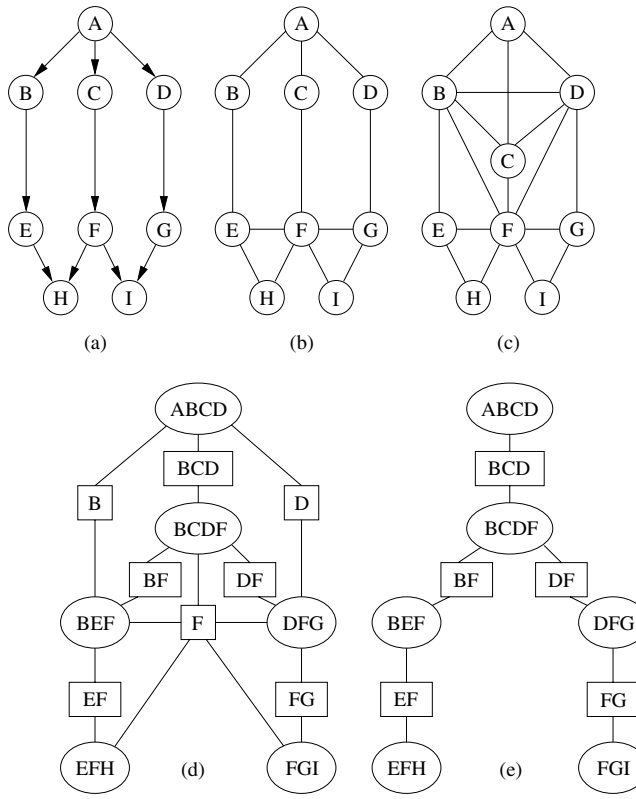


FIGURE 3.11

Example of junction tree algorithm application: (a) the original multiply-connected BN; (b) the moral graph; (c) the triangulated graph; (d) the junction graph; and (e) the junction tree. (From Jensen, F.V. *An Introduction to Bayesian Networks*. UCL Press, London, 1996. With permission.)

mined by the sizes of the state space of the compound nodes in the junction tree. It is possible to estimate the computational cost of performing belief updating through the **junction tree cost** [139, 70]. Suppose the network has been transformed into a junction tree with new compound nodes $C_1, \ldots, C_i, \ldots, C_n$, The junction tree cost is defined as:

$$\sum_{C_i \in \{C_1,\ldots,C_n\}} \left(k_i \prod_{X \in C_i} |\Omega_X| \right) \tag{3.6}$$

where k_i is the number of separation sets involving clique C_i, i.e., the total number of parents and children of C_i in the junction tree. So the junction-tree cost is the product of the size of all the constituent nodes in the cluster and the number of children and parents, summed over all the compound nodes in the junction tree. The junction tree

© 2004 by Chapman & Hall/CRC Press LLC

cost provides a measure that allows us to compare different junction trees, obtained from different triangulations.

While the size of the compound nodes in the junction tree can be prohibitive, in terms of memory as well as computational cost, it is often the case that many of the table entries are zeros. Compression techniques can be used to store the tables more efficiently, without these zero entries[‡].

3.6 Approximate inference with stochastic simulation

In general, exact inference in belief networks is computationally complex, or more precisely, NP hard [52]. In practice, for most small to medium sized networks, up to three dozen nodes or so, the current best exact algorithms — using clustering — are good enough. For larger networks, or networks that are densely connected, approximate algorithms must be used.

One approach to approximate inference for multiply-connected networks is **stochastic simulation**. Stochastic simulation uses the network to generate a large number of cases from the network distribution. The posterior probability of a target node is estimated using these cases. By the Law of Large Numbers from statistics, as more cases are generated, the estimate converges on the exact probability.

As with exact inference, there is a computational complexity issue: approximating to within an arbitrary tolerance is also NP hard [64]. However, in practice, if the evidence being conditioned upon is not too unlikely, these approximate approaches converge fairly quickly.

Numerous other approximation methods have been developed, which rely on modifying the representation, rather than on simulation. Coverage of these methods is beyond the scope of this text (see Bibliographic Notes §3.10 for pointers).

3.6.1 Logic sampling

The simplest sampling algorithm is that of logic sampling (LS) [106] (Algorithm 3.3). This generates a case by randomly selecting values for each node, weighted by the probability of that value occurring. The nodes are traversed from the root nodes down to the leaves, so at each step the weighting probability is either the prior or the CPT entry for the sampled parent values. When all the nodes have been visited, we have a "case," i.e., an **instantiation** of all the nodes in the BN. To estimate $P(X|E)$ with a sample value $P'(X|E)$, we compute the ratio of cases where both X and E are true to the number of cases where just E is true. So after the generation of each case, these combinations are counted, as appropriate.

[‡] The Hugin software uses such a compression method.

© 2004 by Chapman & Hall/CRC Press LLC

ALGORITHM 3.3

Logic Sampling (LS) Algorithm

Aim: *to compute* $P'(X|E = e)$ *as an estimate of the posterior probability for node* X *given evidence* $E = e$.

Initialization

- *For each value* x_i *for node* X
  Create a count variable Count(x_i*,*e*)*
- *Create a count variable Count(*e*)*
- *Initialize all count variables to 0*

For each round of simulation

1. *For all root nodes*
   Randomly choose a value for it, weighting the choice by the priors.
2. *Loop*
   Choose values randomly for children, using the conditional probabilities given the known values of the parents.

   Until all the leaves are reached
3. *Update run counts:*
   If the case includes $E = e$
   Count(e*)* $\leftarrow$ *Count(*e*) + 1*

   If this case includes both $X = x_i$ *and* $E = e$
   Count(x_i*,*e*)* $\leftarrow$ *Count(*x_i*,*e*) + 1*

Current estimate for the posterior probability

$$P'(X = x_i|E = e) = \frac{Count(x_i, E = e)}{Count(E = e)}$$

Let us work through one round of simulation for the metastatic cancer example.

- The only root node for this network is node M, which has prior $P(M = T) = 0.2$. The random number generator produces a value between 0 and 1; any number > 0.2 means the value F is selected. Suppose that is the case.
- Next, the values for children S and B must be chosen, using the CPT entries $P(S = T|M = F)$ and $P(B = T|M = F)$. Suppose that values $S = T$ and $B = F$ are chosen randomly.
- Finally, the values for C and H must be chosen weighted with the probabilities $P(M = |S = T, B = F)$ and $P(H|S = T, B = F)$. Suppose that values $M = F$ and $H = T$ are selected.
- Then the full "case" for this simulation round is the combination of values: $C = F, S = T, B = F, M = F$ and $H = T$.

© 2004 by Chapman & Hall/CRC Press LLC

- If we were trying to update the beliefs for a person having metastatic cancer or not (i.e., $Bel(M)$) given they had a severe headache, (i.e., evidence E that $H = T$), then this case would add one to the count variable Count($H = T$), one to Count($M = F$,$H = T$), but not to Count($M = T$,$H = T$).

The LS algorithm is easily generalized to more than one query node. The main problem with the algorithm is that when the evidence E is unlikely, most of the cases have to be discarded, as they don't contribute to the run counts.

3.6.2 Likelihood weighting

A modification to the LS algorithm called likelihood weighting (LW) [87, 248] (Algorithm 3.4) overcomes the problem with unlikely evidence, always employing the sampled value for each evidence node. However, the same straightforward counting would result in posteriors that did not reflect the actual BN model. So, instead of adding "1" to the run count, the CPTs for the evidence node (or nodes) are used to determine how likely that evidence combination is, and that fractional likelihood is the number added to the run count.

ALGORITHM 3.4
Likelihood Weighting (LW) Algorithm
Aim: *to compute $P'(X|E = e)$, an approximation to the posterior probability for node X given evidence $\mathbf{E} = e$, consisting of $\{E_1 = e_1, \ldots, E_n = e_n\}$.*

Initialization

- *For each value x_i for node X*
  Create a count variable Count(x_i,e)
- *Create a count variable Count(e)*
- *Initialize all count variables to 0*

For each round of simulation

1. *For all root nodes*
   If a root is an evidence node, E_j
   choose the evidence value, e_j
   likelihood($E_j = e_j$) ← P($E_j = e_j$)
   Else
   Choose a value for the root node, weighting the choice by the priors.
2. *Loop*
   If a child is an evidence node, E_j
   choose the evidence value, e_j
   likelihood($E_j = e_j$) = P($E_j = e_j$|chosen parent values)
   Else
   Choose values randomly for children, using the conditional probabilities given the known values of the parents.

Until all the leaves are reached

© 2004 by Chapman & Hall/CRC Press LLC

3. *Update run counts:*
   If the case includes $E = e$
      Count(e) $\leftarrow$ *Count(e)* + $\prod_j$ *likelihood*$(E_j = e_j)$
   If this case includes both $X = x_i$ *and* $E = e$
      Count(x_i*,e)* $\leftarrow$ *Count(*x_i*,e)* + $\prod_j$ *likelihood*$(E_j = e_j)$

Current estimate for the posterior probability

$$P'(X = x_i|E = e) = \frac{Count(x_i, E = e)}{Count(E = e)}$$

Let's look at how this works for another metastatic cancer example, where the evidence is $B = T$, and we are interested in probability of the patient going into a coma. So, we want to compute an estimate for $P(C = T|B = T)$.

- Choose a value for M with prior $P(M) = 0.2$. Assume we choose $M = F$.
- Next we choose a value for S from distribution $P(S|M = F) = 0.20$. Assume $S = T$ is chosen.
- Look at B. This is an evidence node that has been set to T and $P(B = T|M = F) = 0.05$. So this run counts as 0.05 of a complete run.
- Choose a value for C randomly with $P(C|S, B) = 0.80$. Assume $C = T$.
- So, we have completed a run with likelihood 0.05 that reports $C = T$ given $B = T$. Hence, both Count($C = T$,$B = T$) and Count($B = T$) are incremented.

3.6.3 Markov Chain Monte Carlo (MCMC)

Both the logic sampling and likelihood weighting sampling algorithms generate each sample individually, starting from scratch. MCMC on the other hand generates a sample by making a random change to the previous sample. It does this by randomly sampling a value for one of the non-evidence nodes X_i, conditioned on the current value of the nodes in its Markov blanket, which consists of the parents, children and children's parents (see §2.2.2).

The technical details of why MCMC returns consistent estimates for the posterior probabilities are beyond the scope of this text (see [238] for details). Note that different uses of MDMC are presented elsewhere in this text, namely Gibbs sampling (for parameter learning) in §7.3.2.1 and Metropolis search (for structure learning) in §8.6.2.

3.6.4 Using virtual evidence

There are two straightforward alternatives for using virtual evidence with sampling inference algorithms[§].

[§]Thanks to Bob Welch and Kevin Murphy for these suggestions.

© 2004 by Chapman & Hall/CRC Press LLC

1. Use a virtual node: add an explicit virtual node *V* to the network, as described in §3.4, and run the algorithm with evidence *V=T*.
2. In the likelihood weighting algorithm, we already weight each sample by the likelihood. We can set $likelihood(E_j)$ to the normalized likelihood ratio.

3.6.5 Assessing approximate inference algorithms

In order to assess the performance of a particular approximate inference algorithm, and to compare algorithms, we need to have a measure for the quality of the solution at any particular time. One possible measure is the **Kullback-Leibler divergence** between a true distribution P and the estimated distribution P' of a node with states i, given by[¶]:

Definition 3.1 Kullback-Leibler divergence

$$KL(P, P') = \sum_i P(i) \log \frac{P(i)}{P'(i)}$$

Note that when P and P' are the same, the KL divergence is zero (which is proven in §10.8). When $P(i)$ is zero, the convention is to take the summand to have a zero value. And, KL divergence is undefined when $P'(i) = 0$; standardly, it is taken as infinity (unless also $P(i) = 0$, in which case the summand is 0).

Alternatively, we can put this measure in terms of the updated belief for query node X.

$$KL(Bel(X), Bel'(X)) = \sum_{x_i} Bel(X = x_i) \log \frac{Bel(X = x_i)}{Bel'(X = x_i)} \qquad (3.7)$$

where $Bel(X)$ is computed by an exact algorithm and $Bel'(X)$ by the approximate algorithm. Of course, this measure can only be applied when the network is such that the exact posterior *can* be computed.

When there is more than one query node, we should use the marginal KL divergence over all the query nodes. For example, if X and Y are query nodes, and Z the evidence, we should use $KL(P(X, Y|Z), P'(X, Y|Z))$. Often the average or the sum of the KL divergences for the individual query nodes are used to estimate the error measure, which is not exact. Problem 3.11 involves plotting the KL divergence to compare the performance of approximate inference algorithms.

An example of this use of KL divergence is shown in Figure 3.12. These graphs show the results of an algorithm comparison experiment [205]. The test network contained 99 nodes and 131 arcs, and the LS and LW algorithms were compared for two cases:

[¶] Although "Kullback-Leibler distance" is the commonly employed word, since the KL measure is asymmetric — measuring the difference from the point of view of one or the other of the distributions — it is no true distance.

© 2004 by Chapman & Hall/CRC Press LLC

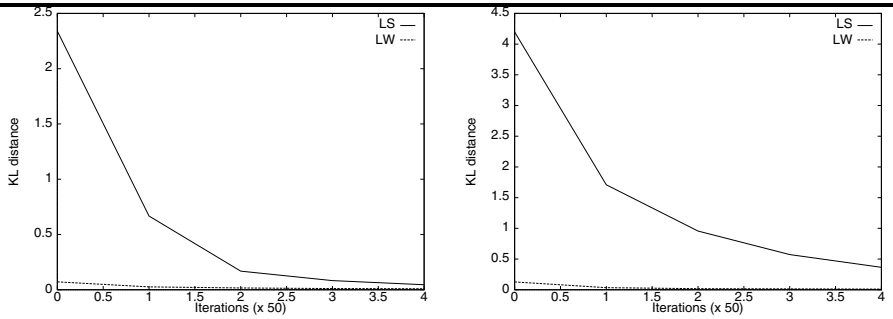


FIGURE 3.12
Comparison of the logic sampling and likelihood-weighting approximate inference algorithms.

- Experiment 1: evidence added for 1 root node, while query nodes were all 35 leaf nodes.
- Experiment 2: evidence added for 1 leaf node, while query nodes were all 29 root nodes (29).

As well as confirming the faster convergence of LW compared to LS, these and other results show that stochastic simulation methods perform better when evidence is nearer to root nodes [205]. In many real domains when the task is one of diagnosis, evidence tends to be near leaves, resulting in poorer performance of the stochastic simulation algorithms.

3.7 Other computations

In addition to the standard BN inference described in this chapter to data — computing the posterior probability for one or more query nodes — other computations are also of interest and provided by some BN software.

3.7.1 Belief revision

It is sometimes the case that rather than updating beliefs given evidence, we are more interested in the most probable values for a set of query nodes, given the evidence. This is sometimes called **belief revision** [217, Chapter 5]. The general case of finding a most probable instantiation of a set of n variables is called **MAP** (maximum aposteriori probability). MAP involves finding the assignment for the n variables that maximizes $P(X_1 = x1, \ldots, X_n = x_n|\mathbf{E})$. Finding MAPs was first shown to be

© 2004 by Chapman & Hall/CRC Press LLC

NP hard [254], and more recently NP complete [213]; approximating MAPs is also NP hard [1].

A special case of MAP is finding an instantiation of *all* the non-evidence nodes, also known as computing a **most probable explanation (MPE)**. The "explanation" of the evidence is a complete assignment of all the non-evidence nodes, $\{Y_1 = y1, \ldots, Y_n = y_m\}$, and computing the MPE means finding the assignment that maximizes $P(Y_1 = y1, \ldots, Y_n = y_n|\mathbf{E})$. MPE can be calculated efficiently with a similar method to probability updating (see [128] for details). Most BN software packages have a feature for calculating MPE but not MAP.

3.7.2 Probability of evidence

When performing belief updating, it is usually the case that the probability of the evidence, P(E), is available as a by-product of the inference procedure. For example, in polytree updating, the normalizing constant α is just $1/P(E)$. Clearly, there is a problem should $P(E)$ be zero, indicating that this combination of values is **impossible** in the domain. If that impossible combination of values is entered as evidence, the inference algorithm must detect and flag it.

The BN user must decide between the following alternatives.

1. It is indeed the case that the evidence is impossible in their domain, and therefore the data are incorrect, due to errors in either gathering or entering the data.

2. The evidence should not be impossible, and the BN incorrectly represents the domain.

This notion of possible incoherence in data has been extended from impossible evidence to unlikely combinations of evidence. A **conflict measure** has been proposed to detect possible incoherence in evidence [130, 145]. The basic idea is that correct findings from a coherent case covered by the model support each other and therefore would be expected to be positively correlated. Suppose we have a set of evidence $\mathbf{E} = \{E_1 = e_1, \ldots, E_m = e_m\}$. A conflict measure on $\mathbf{E}$ is

$$C(\mathbf{E}) = \log \frac{P(E_1 = e_1), \ldots, P(E_m = e_m)}{P(E)}$$

$C(\mathbf{E})$ being positive indicates that the evidence may be conflicting. The higher the conflict measure, the greater the discrepancy between the BN model and the evidence. This discrepancy may be due to errors in the data or it just may be a rare case. If the conflict is due to flawed data, it is possible to trace the conflicts.

© 2004 by Chapman & Hall/CRC Press LLC

3.8 Causal inference

There is no consensus in the community of Bayesian network researchers about the proper understanding of the relation between causality and Bayesian networks. The majority opinion is that there is nothing special about a **causal interpretation**, that is, one which asserts that corresponding to each (non-redundant) direct arc in the network not only is there a probabilistic dependency but also a causal dependency. As we saw in Chapter 2, after all, by reordering the variables and applying the network construction algorithm we can get the arcs turned around! Yet, clearly, *both* networks cannot be causal.

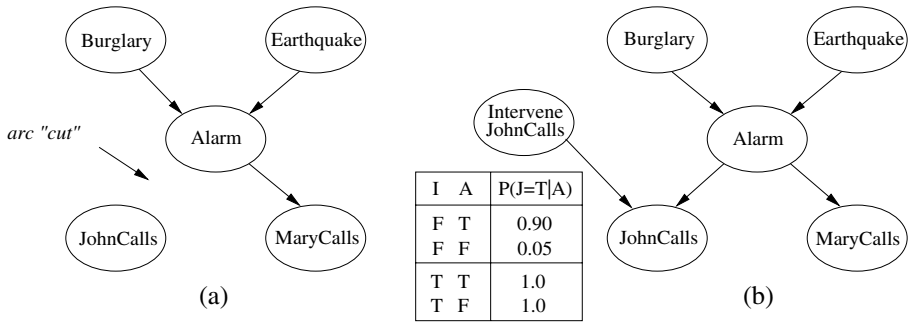
We take the minority point of view, however (one, incidentally, shared by Pearl [218] and Neapolitan [199]), that causal structure is what *underlies* all useful Bayesian networks. Certainly not all Bayesian networks are causal, but if they represent a real-world probability distribution, then some causal model is their source.

Regardless of how that debate falls out, however, it is important to consider how to do inferences with Bayesian networks that *are* causal. If we have a causal model, then we can perform inferences which are not available with a non-causal BN. This ability is important, for there is a large range of potential applications for particularly causal inferences, such as process control, manufacturing and decision support for medical intervention. For example, we may need to reason about what will happen to the quality of a manufactured product if we adopt a cheaper supplier for one of its parts. Non-causal Bayesian networks, and causal Bayesian networks using ordinary propagation, are currently used to answer just such questions; but this practice is wrong. Although the Bayesian network tools do not explicitly support causal reasoning, we will nevertheless now explain how to do it properly.

Consider again Pearl's earthquake network of Figure 2.6. That network is intended to represent a causal structure: each link makes a specific causal claim. Since it is a Bayesian network (causal or not), if we observe that *JohnCalls* is true, then this will raise the probability of *MaryCalls* being true, as we know. However, if we *intervene*, somehow forcing John to call, this probability raising inference will no longer be valid. Why? Because the reason an observation raises the probability of Mary calling is that there is a common cause for both, the Alarm; so one provides evidence for the other. However, under intervention we have effectively cut off the connection between the Alarm and John's calling. The belief propagation (message passing) from JohnCalls to Alarm and then down to MaryCalls is all wrong under **causal intervention**.

Judea Pearl, in his recent book *Causality* [218], suggests that we understand the "effectively cut off" above quite literally, and model causal intervention in a variable X simply by (temporarily) cutting all arcs from $Parents(X)$ to X. If you do that with the earthquake example (see Figure 3.13(a)), then, of course, you will find that forcing John to call will tell us nothing about earthquakes, burglaries, the Alarm or Mary — which is quite correct. This is the simplest way to model causal interventions and often will do the job.

© 2004 by Chapman & Hall/CRC Press LLC



I	A	P(J=T|A)
F	T	0.90
F	F	0.05
T	T	1.0
T	F	1.0

FIGURE 3.13

Modeling causal interventions: (a) Pearl's cut method; (b) augmented model (CPT unchanged when no intervention).

There is, however, a more general approach to causal inference. Suppose that the causal intervention itself is only probabilistic. In the John calling case, we can imagine doing something which will simply *guarantee* John's cooperation, like pulling out a gun. But suppose that we have no such guarantees. Say, we are considering a middle-aged patient at genetic risk of cardiovascular disease. We would like to model life and health outcomes assuming we persuade the patient to give up *Smoking*. If we simply cut the connection between *Smoking* and its parents (assuming the *Smoking* is caused!), then we are assuming our act of persuasion has a probability one of success. We might more realistically wish to model the effectiveness of persuasion with a different probability. We can do that by making an **augmented model**, by adding a new parent of *Smoking*, say *Intervene-on-Smoking*. We can then instrument the CPT for *Smoking* to include whatever probabilistic impact the (attempted) intervention has. Of course, if the interventions are fully effective (i.e., with probability one), this can be put into the CPT, and the result will be equivalent to Pearl's cut method (see Figure 3.13(b)). But with the full CPT available, any kind of intervention — including one which interacts with other parents — may be represented. Subsequent propagation with the new intervention node "observed" (but not its intervened-upon effect) will provide correct causal inference, given that we have a causal model of the process in question to begin with.

Adding intervention variables will unfortunately alter the original probability distribution over the unaugmented set of variables. That distribution can be recovered by setting the priors over the new variables to zero. Current Bayesian network tools, lacking support for explicit causal modeling, will not then allow those variables to be instantiated, because of the idea that this constitutes an impossible observation; an idea that is here misplaced. As a practical matter, using current tools, one could maintain two Bayesian networks, one with and one without intervention variables, in order to simplify moving between reasoning with and without interventions[‖].

[‖] This point arose in conversation with Richard Neapolitan.

© 2004 by Chapman & Hall/CRC Press LLC

3.9 Summary

Reasoning with Bayesian networks is done by updating beliefs — that is, computing the posterior probability distributions — given new information, called evidence. This is called probabilistic inference. While both exact and approximate inference is theoretically computationally complex, a range of exact and approximate inference algorithms have been developed that work well in practice. The basic idea is that new evidence has to be propagated to other parts of the network; for simple tree structures an efficient message passing algorithm based on local computations is used. When the network structure is more complex, specifically when there are multiple paths between nodes, additional computation is required. The best exact method for such multiply-connected networks is the junction tree algorithm, which transforms the network into a tree structure before performing propagation. When the network gets too large, or is highly connected, even the junction tree approach becomes computationally infeasible, in which case the main approaches to performing approximate inference are based on stochastic simulation. In addition to the standard belief updating, other computations of interest include the most probable explanation and the probability of the evidence. Finally, Bayesian networks can be augmented for causal modeling, that is for reasoning about the effect of causal interventions.

3.10 Bibliographic notes

Pearl [215] developed the message passing method for inference in simple trees. Kim [144] extended it to polytrees. The polytree message passing algorithm given here follows that of Pearl [217], using some of Russell's notation [237].

Two versions of junction tree clustering were developed in the late 1980s. One version by Shafer and Shenoy [252] (described in [128]) suggested an elimination method resulting in a message passing scheme for their so-called join-tree structure, a term taken from the relational data base literature. The other method was initially developed by Lauritzen and Spiegelhalter [169] as a two stage method based on the running intersection property. This was soon refined to a message passing scheme in a junction tree [124, 127, 125] described in this chapter**.

The junction tree cost given here is an estimate, produced by Kanazawa [139, 70], of the complexity of the junction tree method.

Another approach to exact inference is cutset conditioning [216, 113], where the network is transformed into multiple, simpler polytrees, rather than the single, more

** Thanks to Finn Jensen for providing a summary chronology of junction tree clustering.

© 2004 by Chapman & Hall/CRC Press LLC

complex polytree produced by clustering. This approach has not developed as a serious contender to the clustering methods implemented in BN software.

Another exact inference algorithm implemented in some current BN software is variable elimination for updating the belief of a single query node (and a variant, bucket elimination), based on product and marginalization operators; a clear exposition is given in [238]. Note that some software (e.g., JavaBayes) also refers to "bucket tree elimination," a somewhat confusing name for what is essentially a junction tree approach.

The logic sampling method was developed by Henrion [106], while likelihood weighting was developed in [87, 248]. Other approximation methods based on stochastic sampling not covered in this book include Gibbs sampling [128], self-importance sampling and heuristic-importance sampling [248], adaptive importance sampling [43], and backward sampling [88].

There have been a number of other approximate inference methods proposed. These include state space abstraction [297], localized partial evaluation [76], weak arc removal [148] and using a mutual information measure to guide approximate evaluation [133]. It has also been shown that applying Pearl's polytree algorithm to general networks, as suggested by Pearl [217], — so-called **loopy propagation** — can be a both fast and accurate approximate method [197]. To our knowledge, none of these methods have been implemented in widely available BN software. Hugin implements an approximation scheme that involves setting very small probabilities in the junction tree tables to zero, which in turns allows more compression to take place.

Cooper [52] showed that the general problem of inference in belief networks is NP hard, while Dagum and Luby [64] showed the problem of approximating the new beliefs to within an arbitrary tolerance is also NP hard.

3.11 Problems

Message passing

Problem 1

Consider (again — see Problem 2.8, Chapter 2) the belief network for another version of the medical diagnosis example, where *B=Bronchitis*, *S=Smoker*, *C=Cough*, *X=Positive X-ray* and *L=Lung cancer* and all nodes are Booleans. Suppose that the prior for a patient being a smoker is 0.25, and the prior for the patient having bronchitis (during winter in Melbourne!) is 0.05.

© 2004 by Chapman & Hall/CRC Press LLC

A Quick Guide to Using Hugin

Installation: Web Site www.hugin.com. Download Hugin Lite, which is available for MS Windows (95 / 98 / NT4 / 2000 / XP), Solaris Sparc, Solaris X86 and Linux. This gives you `HuginLite63.exe`, a self-extracting zip archive. Double-clicking will start the extraction process.

Network Files: BNs are stored in `.net` files, with icon . Hugin comes with a `samples` folder of example networks. To open an existing network, select , or select `File`→`Open` menu option, or double-click on the file.

Compilation: Once a Hugin BN has been opened, before you can see the initial beliefs or add evidence, you must first compile it (which they call "switch to run mode"): click on , or select `Network`→`Run`(in edit mode), or `Recompile` (in run mode) menu option.

This causes another window to appear on the left side of the display (called the Node Pane List), showing the network name, and all the node names. You can display/hide the states and beliefs in several ways. You can select a particular node by clicking on the '+' by the node name, or all nodes with `View`→`Expand Node List`, or using icon . Unselecting is done similarly with '-', or `View`→`Collapse Node List`, or using icon .

Selecting a node means all its states will be displayed, together with a bar and numbers showing the beliefs. Note that Hugin beliefs are given as percentages out of 100, not as direct probabilities (i.e., not numbers between 0 and 1).

Editing/Creating a BN: You can only change a BN when you are in "edit" mode, which you can enter by selecting the edit mode icon , or selecting `Network`→`Edit`. Double-clicking on a node will bring up a window showing node features, or use icon .

- Add a node by selecting either (for discrete node) or (for continuous node), `Edit`→`Discrete Chance Tool` or `Edit`→`Continuous Chance Tool`. In each case, you then "drag-and-drop" with the mouse.
- Add an arc by selecting either , or `Edit`→`Link Tool`, then left-click first on the parent node, then the child node.
- Click on the icon to split the window horizontally between a Tables Pane (above), showing the CPT of the currently selected node, and the network structure (below).

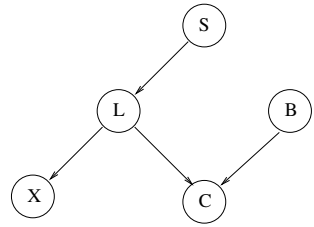
Saving a BN: Select , or the `File`→`Save` menu option. Note that the Hugin Lite demonstration version limits you to networks with up to 50 nodes; for larger networks, you need to buy a license.

Junction trees: To change the triangulation method select `Network`→`Network Properties`→`Compilation`, then turn on "`Specify Triangulation Method`." To view, select the `Show Junction Tree` option.

FIGURE 3.14

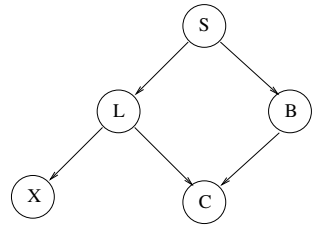
A quick guide to using Hugin.

© 2004 by Chapman & Hall/CRC Press LLC



Suppose that evidence is obtained that a patient has a positive X-ray result and a polytree message passing algorithm was to be used to perform belief updating.

1. Write down the π and λ values for the following nodes: *S*, *B*, *C*, *X*.
2. Show the 3 stages of data propagation by the message passing algorithm in terms of the π and λ messages.
3. Suppose that a doctor considered that smoking was a contributing factor towards getting bronchitis as well as cancer. The new network structure reflecting this model is as follows.



Why is the polytree message passing algorithm unsuitable for performing belief updating on this network?

Virtual / Likelihood evidence

Problem 2

A description of how to enter so-called likelihood evidence in both Netica and Hugin software is given in Figure 3.15. In §3.4 we only looked at an example where we added virtual evidence for a root node; however it can be added for any node in the network. Consider the cancer example from Chapter 2 (supplied as Netica file `cancer.dne`). Suppose that the radiologist who has taken and analyzed the X-ray in this cancer example is uncertain. He thinks that the X-ray looks positive, but is only 80% sure.

- Add this evidence as a likelihood ratio of 4:1.
- Work out the likelihood ratio that needs to be used to produce a new belief of $Bel(\textit{X-ray}) = 0.8$
- Add this likelihood ratio as evidence to confirm your calculations.
- Add an explicit virtual node *V* to represent the uncertainty in the observation. Confirm that adding specific evidence for *V* gives the same results as adding the likelihood evidence.

© 2004 by Chapman & Hall/CRC Press LLC

Adding virtual evidence

Suppose you want to add uncertain evidence for node X with values $\{x_1, \ldots, x_n\}$ and that you have worked out that the w that the likelihood ratio vector is $(r_1 : r_2 : \ldots : r_n)$ (as described in §3.4).

Netica

Right click on node and select the `likelihood` option. This will bring up a series of windows, the first of which asks you to provide a probability (default is 1) for $P(V|X = x_1)$.

Here you should enter the probability r_1. Note that this *is* a probability, unlike everywhere else in Netica, where probabilities are given as values out of 100. You are asked for a series of these probabilities, one for each possible value of X.

If you try to re-enter likelihood evidence for a node, you will be asked if you want to discard previous evidence. If you do not, both the old and new evidence will be incorporated, equivalent to using multiple virtual nodes for each piece of evidence.

Hugin

To add evidence in Hugin, you have the following options.

- With a node selected, select `Network→Enter Likelihood`.
- Right click on belief bar/belief/state entry, and then choose the `Enter Likelihood` option.
- Click on the icon .

This brings up a window, showing each state with a horizontal bar whose length represents the evidence with a value from 0 to 1. The default for all states is 1.

- To set specific evidence, set a single value x_i to 1.0, and all the others to 0.
- To set negative evidence, set a single value x_j to 0, and leave all the others 1.
- To set likelihood evidence, set each value x_k corresponding to the r_k ratio that you have previously determined.

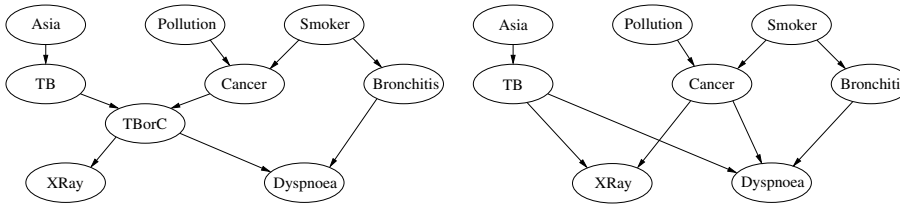
FIGURE 3.15
Adding virtual evidence in Netica and Hugin.

© 2004 by Chapman & Hall/CRC Press LLC

Clustering

Problem 3

Consider the BNs for the "Asia" problem described in §2.5.3.



1. Use the Jensen junction tree algorithm (Algorithm 3.2) to construct a junction tree from both these networks, drawing (by hand) the resultant junction trees.
2. Load these networks (`asia1.dne` and `asia2.dne` in the on-line material) into Netica.

   (a) Compile the networks using standard Netica compilation.

   (b) Inspect the junction tree by selecting `Report→Junction tree` menu option, and note the elimination ordering (used in Netica's junction tree algorithm) by selecting `Report→Elimination Ordering` menu option. How do the junction trees produced by Netica compare to the ones you computed using Algorithm 3.2? What are the junction tree costs of each?

   (c) Re-compile the networks using the `Network→Compile Optimizing` menu option. Inspect again the junction trees and the elimination orderings. How much do they differ?

3. Now load the Hugin versions of these networks (`asia.net` and `asia2.net` in the on-line material), and compile them with varying settings of the triangulation heuristic. (See **A Quick Guide to Using Hugin Expert** in Figure 3.14.)

   (a) Clique Size

   (b) Clique Weight

   (c) Fill-in Size

   (d) Fill-in Weight

   (e) Optimal Triangulation

   How do the resultant junction trees differ in structure and corresponding junction tree cost

   (a) From each other?

   (b) From the those you obtained executing the algorithm by hand?

   (c) From those obtained from Netica's standard and optimized compilation?

© 2004 by Chapman & Hall/CRC Press LLC

Approximate inference

Problem 4

The on-line material for this text provides a version of both the Logic Sampling algorithm (Algorithm 3.3) and the Likelihood Weighting algorithm (Algorithm 3.4).

Take an example BN (either provided with the online material, or that you've developed for the problems set in Chapter 2) and do the following.

1. Run the BN software to obtain the exact inference result.
2. Run the LS Algorithm, printing out the approximate beliefs every 10 iterations and stopping when a certain level of convergence has been achieved.
3. Do the same for the LW algorithm.
4. As we have seen, the Kullback-Leibler divergence (§3.6.5) can be used to measure the error in the beliefs obtained using an approximate inference algorithm. Compute and plot the KL error over time for both the LS and LW algorithm.
5. Investigate what effect the following changes may have on the error for the LW algorithm.

    - Vary the priors between (i) more uniform and (ii) more extreme.
    - Vary the location of the evidence (i) root, (ii) intermediate and (iii) leaf.
    - Set evidence that is more or less likely.

Problem 5

As mentioned in the Bibliographic Notes, Hugin implements an approximation scheme that involves setting very small probabilities in the junction tree tables to zero, which in turns allows more compression to take place.

To turn on the approximation scheme, select `Network→Network Properties→Compilation`, then check the approximate optimization box.

As for the previous problem, select an example BN and perform an investigation of the effect of varying the approximation threshold on the belief updating, again using the KL divergence measure.

Causal reasoning

Problem 6

Take Pearl's earthquake example. Suppose there is an intervention on *JohnCalls*.

1. Use ordinary observation and updating. What is *Bel(Burglary)*?
2. Use Pearl's cut-link method. What is *Bel(Burglary)*? this case?
3. Use the augmented model with 0.9 effectiveness. What is *Bel(Burglary)*?

Now add an independent observation for *MaryCalls=T*. Parts 4, 5 and 6 of this problem involve repeating the parts 1, 2 and 3 for this new situation.

© 2004 by Chapman & Hall/CRC Press LLC

4

Decision Networks

4.1 Introduction

By now we know how to use Bayesian networks to represent uncertainty and do probabilistic inference. In this chapter we extend them to support decision making. Adding an explicit representation of both the actions under consideration and the value or **utility** of the resultant outcomes gives us **decision networks** (also called **influence diagrams** [115]). Bayesian decision networks combine probabilistic reasoning with utilities, helping us to make decisions that maximize the **expected utility**, as described in §1.5.3.

We will begin with utilities and then describe how they are represented together with probabilities in decision networks. Then we present the algorithm for evaluating a decision network to make individual decisions, illustrating with several examples. We are also interested in determining the best sequences of decisions or actions, that is to say, with **planning**. First, we will use a decision network for a "test-then-act" combination of decisions. Then, we introduce **dynamic Bayesian networks** for explicitly modeling how the world changes over time. This allows us to generalize decision networks to **dynamic decision networks**, which explicitly model sequential decision making or planning under uncertainty.

4.2 Utilities

When deciding upon an action, we need to consider our **preferences** between the different possible outcomes of the available actions. As already introduced in §1.5.3, **utility theory** provides a way to represent and reason with preferences. A **utility function** quantifies preferences, reflecting the "usefulness" (or "desirability") of the outcomes, by mapping them to the real numbers.

Such a mapping allows us to combine utility theory with probability theory. In particular, it allows us to calculate which action is expected to deliver the most value (or utility) given any available evidence E in its **expected utility**:

$$EU(A|E) = \sum_i P(O_i|E, A)\, U(O_i|A) \tag{4.1}$$

© 2004 by Chapman & Hall/CRC Press LLC

where

- *E* is the available evidence,
- *A* is a non-deterministic action with possible outcome states O_i,
- $U(O_i|A)$ is the utility of each of the outcome states, given that action A is taken,
- $P(O_i|E, A)$ is the conditional probability distribution over the possible outcome states, given that evidence E is observed and action A taken.

The **principle of maximum expected utility** asserts that an essential part of the nature of rational agents is to choose that action which maximizes the expected utility. In the following section we will see how to extend Bayesian networks to model such rational decision making.

First, however, we will point out a few pitfalls in constructing utility functions. The simplest, and very common, way of generating utilities, especially for business modeling, is to equate utilities with money. Indeed, in the case of business modeling this may well be satisfactory. Perhaps the most obvious caveat is that there is a time value associated with money, so if some outcome delivers money in the future, it should be discounted in comparison with outcomes delivering money in the present (with the discount being equal to standard interest/discount rates). A slightly more subtle point is that even discounted money is rarely just identical to utility. In fact, the relation between money and utility is represented in Figure 4.1, with the dashed line representing the abnormal case where money and utility are identical. The solid line represents the more usual case, which can readily be understood at the extremities: after losing enough money, one is bankrupt, so losing more does not matter; after earning enough money, one is retired, so earning more does not matter. In short, the marginal value of money declines, and this needs to be kept in mind when generating utility functions.

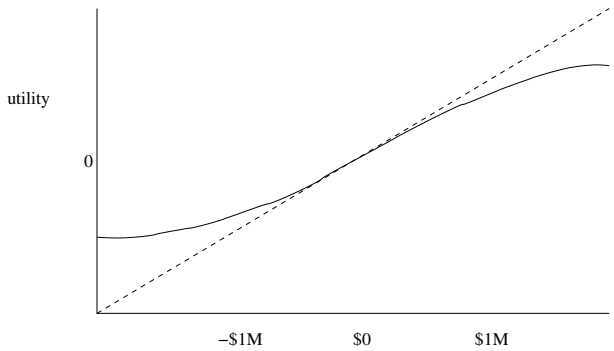


FIGURE 4.1
The utility of money.

© 2004 by Chapman & Hall/CRC Press LLC

Over and above such considerations are others that are possibly less rational. Thus, cognitive psychologists have discovered that most people are **risk averse**, meaning that they will forgo a significant gain in expected value in order to reduce their uncertainty about the future. For example, many people will prefer to keep $10 in their hand, rather than buy a lottery ticket with an expected value of $20 when the probability of losing the $10 is, say, 0.999. Of course, many others are **risk prone** and will happily part with $1 day after day in the hopes of becoming a millionaire, even when the expected value of the gamble is -95 cents. Such behavior may again be explained, in part, through differences between utility and money. However, there does appear to be residual risk aversion and risk proneness which resists such explanation, that is, which remains even when matters are carefully recast in terms of utilities alone [18].

In generating utilities the choice of unit is arbitrary, as utility functions differing only in scale result in the same decisions. The range of utilities can be set by establishing a scale from the best possible outcome U(O_{best}), to some neutral outcome, down to the worst case U(O_{worst}). We discuss how to assess utilities further in §9.3.7.

4.3 Decision network basics

A decision network is an extension of Bayesian networks that is able to represent the main considerations involved in decision making: the state of the world, the decisions or actions under consideration, the states that may result from an action and the utility of those resultant states.

4.3.1 Node types

A decision network consists of three types of nodes, as shown in Figure 4.2.

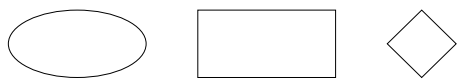


FIGURE 4.2
Decision network node types.

Chance nodes: These have an oval shape and represent random variables, exactly as in Bayesian networks. Each has an associated conditional probability table (CPT), giving the probability of the variable having a particular value given a

© 2004 by Chapman & Hall/CRC Press LLC

combination of values of its parents. Their parent nodes can be decision nodes as well as other chance nodes.

Decision nodes: These have a rectangular shape and represent the decision being made at a particular point in time. The values of a decision node are the actions that the decision maker must choose between. A decision node can have chance nodes as a special kind of parent, indicating that evidence about the parent nodes will be available at the time of decision (see §4.3.4). A decision network representing a single decision has only one decision node, representing an isolated decision. When the network models a sequence of decisions (see §4.4), decision nodes can have other decision nodes as parents, representing the order of decisions.

Utility nodes: These have a diamond shape and represent the agent's utility function. They are also called **value nodes**. The parents of a utility node are the variables describing the outcome state that directly affect the utility and may include decision nodes. Each utility node has an associated **utility table** with one entry for each possible instantiation of its parents, perhaps including an action taken. When there are multiple utility nodes, the overall utility is the sum of the individual utilities.

We will now see how these node types can be used to model a decision problem with the following simple example.

4.3.2 Football team example

Clare's football team, Melbourne, is going to play her friend John's team, Carlton. John offers Clare a friendly bet: whoever's team loses will buy the wine next time they go out for dinner. They never spend more than $15 on wine when they eat out. When deciding whether to accept this bet, Clare will have to assess her team's chances of winning (which will vary according to the weather on the day). She also knows that she will be happy if her team wins and miserable if her team loses, regardless of the bet.

A decision network for this problem is shown in Figure 4.3. This network has one chance node *Result* representing whether Clare's team wins or loses (values $\{$*melb_wins, melb_loses*$\}$), and a second chance node *Weather* which represents whether or not it rains during the match (values $\{$*rain*, *dry*$\}$). It has a binary decision node $AcceptBet$ representing whether or not she bets and a utility node U that measures the decision maker's level of satisfaction.

The priors for the *Weather* reflect the typical match conditions at this time of year. The CPT for *Result* shows that Clare expects her team to have a greater chance of winning if it doesn't rain (as she thinks they are the more skillful team).

There are arcs from *Result* and *AcceptBet* to *U*, capturing the idea that Clare's satisfaction will depend on a combination of the eventual match winner and the betting decision. Her preferences are made explicit in the utility table. The numbers given indicate that the best outcome is when her team wins and she accepted the bet (utility = 40) while the next best outcome is her team wins but she didn't bet on the result

© 2004 by Chapman & Hall/CRC Press LLC

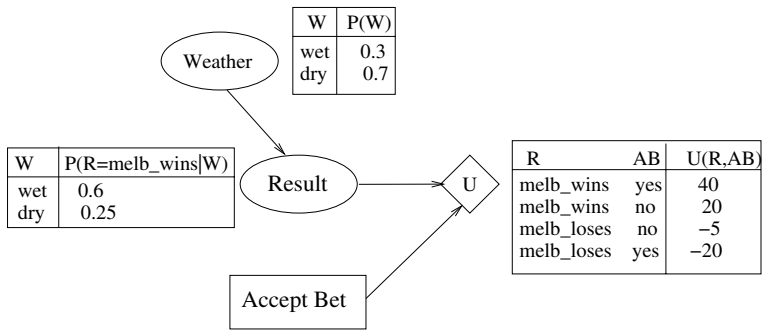


FIGURE 4.3
A decision network for the football team example.

(utility = 20). When her team loses but she didn't bet, Clare isn't happy (utility = -5) but the worst outcome is when she has to buy the dinner wine also (utility = -20). Clearly, Clare's preferences reflect more than the money at risk. Note also that in this problem the decision node doesn't affect any of the variables being modeled, i.e., there is no explicit outcome variable.

4.3.3 Evaluating decision networks

To evaluate a decision network with a single decision node:

ALGORITHM 4.1
Decision Network Evaluation Algorithm (Single decision)

1. *Add any available evidence.*
2. *For each action value in the decision node:*
   (a) *Set the decision node to that value;*
   (b) *Calculate the posterior probabilities for the parent nodes of the utility node, as for Bayesian networks, using a standard inference algorithm;*
   (c) *Calculate the resulting expected utility for the action.*
3. *Return the action with the highest expected utility.*

For the football example, with no evidence, it is easy to see that the expected utility in each case is the sum of the products of probability and utility for the different cases. With no evidence added, the probability of Melbourne winning is

$$\begin{aligned} P(R = melb\_wins) &= P(W = w) \times P(R = melb\_wins|W = w) + P(W = d) \\ &= \times P(R = melb\_wins|W = d) \end{aligned}$$

and the losing probability is $1 - P(R = melb\_wins)$. So the expected utility is:

© 2004 by Chapman & Hall/CRC Press LLC

$$
\begin{aligned}
EU(AB = yes) &= P(R = melb\_wins) \times U(R = melb\_wins | AB = yes) \\
&\quad + P(R = melb\_loses) \times U(R = melb\_loses | AB = yes) \\
&= (0.3 \times 0.6 + 0.7 \times 0.25) \times 40 + (0.3 \times 0.4 + 0.7 \times 0.75) \times -20 \\
&= 0.355 \times 40 + 0.645 \times -20 = 14.2 - 12.9 = 1.3 \\
EU(AB = no) &= P(R = melb\_wins) \times U(R = melb\_wins | AB = no) \\
&\quad + P(R = melb\_loses) \times U(R = melb\_loses | AB = no) \\
&= (0.3 \times 0.6 + 0.7 \times 0.25) \times 20 + (0.3 \times 0.4 + 0.7 \times 0.75) \times -5 \\
&= 0.355 \times 20 + 0.645 \times -5 = 7.1 - 3.225 = 3.875
\end{aligned}
$$

Note that the probability of the outcomes Clare is interested in (i.e., her team winning or losing) is independent of the betting decision. With no other information available, Clare's decision is to not accept the bet.

4.3.4 Information links

As we mentioned when introducing the types of nodes, there may be arcs from chance nodes to decision nodes — these are called **information links** [128, p. 139]. These links are not involved in the basic network evaluation process and have no associated parameters. Instead, they indicate when a chance node needs to be observed before the decision *D* is made — but after any decisions prior to *D*. With an information link in place, network evaluation can be extended to calculate explicitly what decision should be made, *given* the different values for that chance node. In other words, a table of optimal actions conditional upon the different relevant states of affairs can be computed, called a **decision table**. To calculate the table, the basic network evaluation algorithm is extended with another loop, as shown in Algorithm 4.2. We will also refer to this conditional decision table as a **policy**.

ALGORITHM 4.2
Decision Table Algorithm (Single decision node, with information links)

1. *Add any available evidence.*
2. *For each combination of values of the parents of the decision node:*
   (a) *For each action value in the decision node:*
      i. *Set the decision node to that value;*
      ii. *Calculate the posterior probabilities for the parent nodes of the utility node, as for Bayesian networks, using a standard inference algorithm;*
      iii. *Calculate the resulting expected utility for the action.*
   (b) *Record the action with the highest expected utility in the decision table.*
3. *Return the decision table.*

© 2004 by Chapman & Hall/CRC Press LLC

To illustrate the use of information links, suppose that in the football team example, Clare was only going to decide whether to accept the bet or not after she heard the weather forecast. The network can be extended with a *Forecast* node, representing the current weather forecast for the match day, which has possible values {*sunny*, *cloudy* or *rainy*}. *Forecast* is a child of *Weather*. There should be an information link from *Forecast* to *AcceptBet*, shown using dashes in Figure 4.4, indicating that Clare will know the forecast when she makes her decision. Assuming the same CPTs and utility table, the extended network evaluation computes a decision table, for the decision node *given* the forecast, also shown in Figure 4.4. Note that most BN software does not display the expected utilities*. If we want them, we must evaluate the network for each evidence case; these results are given in Table 4.1 (highest expected utility in each case in **bold**).

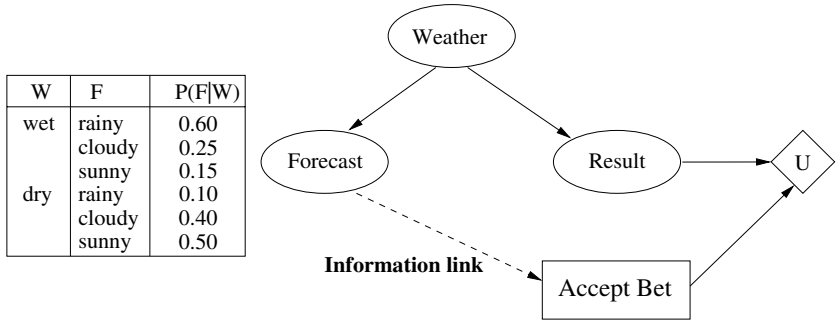


FIGURE 4.4

The football team decision network extended with the *Forecast* node and an information link, with the decision table for *AcceptBet* computed during network evaluation.

TABLE 4.1

Decisions calculated for football team, given the new evidence node *Forecast*

F	$Bel(W = wet)$	$Bel(R = melb\_wins)$	EU(AB=yes)	EU(AB=no)
rainy	0.720	0.502	**10.12**	7.55
cloudy	0.211	0.324	-0.56	**3.10**
sunny	0.114	0.290	-2.61	**2.25**

*GeNIe is one exception that we are aware of.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

4.3.5 Fever example

Suppose that you know that a fever can be caused by the flu. You can use a thermometer, which is fairly reliable, to test whether or not you have a fever. Suppose you also know that if you take aspirin it will almost certainly lower a fever to normal. Some people (about 5% of the population) have a negative reaction to aspirin. You'll be happy to get rid of your fever, as long as you don't suffer an adverse reaction if you take aspirin. (This is a variation of an example in [128].)

A decision network for this example is shown in Figure 4.5. The *Flu* node (the cause) is a parent of the *Fever* (an effect). That symptom can be measured by a thermometer, whose reading *Therm* may be somewhat unreliable. The decision is represented by the decision node *Take Aspirin*. If the aspirin is taken, it is likely to get rid of the fever. This change over time is represented by a second node *FeverLater*. Note that the aspirin has no effect on the flu and, indeed, that we are not modeling the possibility that the flu goes away. The adverse reaction to taking aspirin is represented by *Reaction*. The utility node, *U*, shows that the utilities depend on whether or not the fever is reduced and whether the person has an adverse reaction. The decision table computed for this network is given in Table 4.2 (highest expected utility in each case in **bold**). Note the observing a high temperature changes the decision to taking the aspirin, but further information about having a reaction reverses that decision.

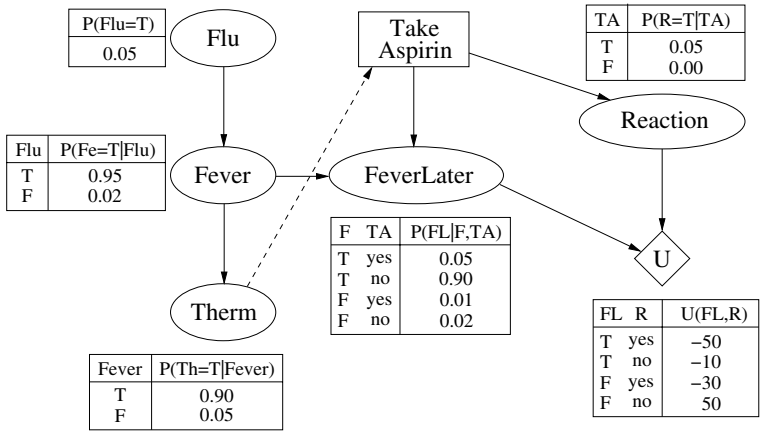


FIGURE 4.5
A decision network for the fever example.

4.3.6 Types of actions

There are two main types of actions in decision problems, intervening and non-intervening. **Non-intervening actions** do not have a direct effect on the chance

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

variables being modeled, as in Figure 4.6(a). Again, in the football team example, making a bet doesn't have any effect on the states of the world being modeled.

TABLE 4.2
Decisions calculated for the fever problem given different values for *Therm* and *Reaction*

Evidence	*Bel(FeverLater=T)*	EU(TA=yes)	EU(TA=no)	Decision
None	0.046	45.27	**45.29**	no
Therm=F	0.525	45.41	**48.41**	no
Therm=T	0.273	**44.1**	19.13	yes
Therm=T & *Reaction=T*	0.273	-30.32	**0**	no

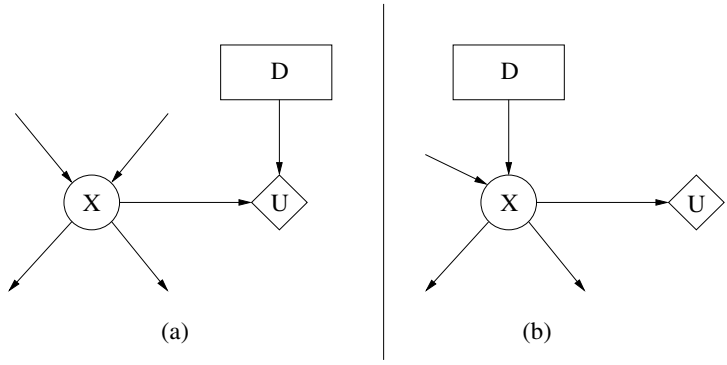


FIGURE 4.6
Generic decision networks for (a) non-intervening and (b) intervening actions.

Intervening actions do have direct effects on the world, as in Figure 4.6(b). In the fever example, deciding to take aspirin will affect the later fever situation. Of course in all decision making, the underlying assumption is that the decision will affect the utility, either directly or indirectly; otherwise, there would be no decision to make.

The use of the term "intervention" here is apt: since the decision impacts upon the real world, it is a form of causal intervention as previously discussed in §3.8. Indeed, one can perform the causal modeling discussed there with the standard Bayesian network tools by attaching a parent decision node to the chance node that one wishes to intervene upon. We would prefer these tools to keep decision making and causal modeling distinct, at least in the human-computer interface. One reason is that causal intervention is generally (if not necessarily) associated with a single chance node, whereas the impact of decisions is often more wide-ranging. In any case, the user's

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

intent is normally quite different. Decision making is all about optimizing a utility-driven decision, whereas causal modeling is about explaining and predicting a causal process under external perturbation.

4.4 Sequential decision making

Thus far, we have considered only single decision problems. Often however a decision maker has to select a sequence of actions, or a **plan**.

4.4.1 Test-action combination

A simple example of a sequence of decisions is when the decision maker has the option of running a test, or more generally making an observation, that will provide useful information before deciding what further action to take.

In the football decision problem used in the previous section, Clare might have a choice as to whether to obtain the weather forecast (perhaps by calling the weather bureau). In the lung cancer example (see §2.2), the physician must decide whether to order an X-ray, before deciding on a treatment option.

This type of decision problem has two stages:

1. The decision whether to run a test or make an observation
2. The selection of a final action

The test/observe decision comes *before* the action decision. And in many cases, the test or observation has an associated cost itself, either monetary, or in terms of discomfort and other physical effects, or both.

A decision network showing the general structure for these test-act decision sequences is shown in Figure 4.7. There are now two decision nodes, *Test*, with values $\{yes, no\}$, and *Action*, with as many values as there are options available. The temporal ordering of the decisions is represented by a **precedence link**, shown as a dotted line.

If the decision is made to run the test, evidence will be obtained for the observation node *Obs*, *before* the *Action* decision is made; hence there is an information link from *Obs* to *Action*. The question then arises as to the meaning of this information link if the decision is made *not* to run the test. This situation is handled by adding an additional state, *unknown*, to the *Obs* node and setting the CPT for *Obs*:

$$P(Obs = unknown|Test = no) = 1$$
$$P(Obs = unknown|Test = yes) = 0$$

In this generic network, there are arcs from the *Action* node to both the chance node *X* and the utility node *U*, indicating intervening actions with a direct associated

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

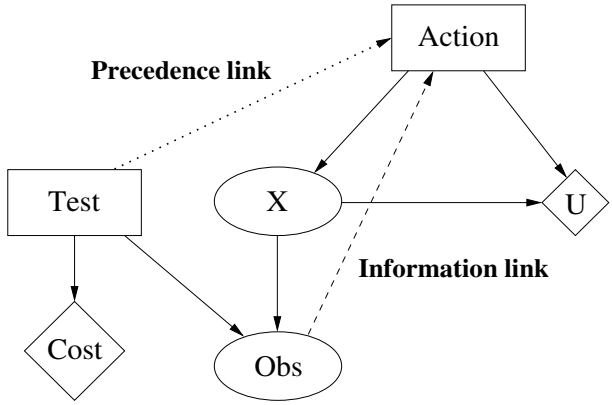


FIGURE 4.7
Decision network for a test-action sequence of decisions.

cost. However, either of these arcs may be omitted, representing a non-intervening action or one with no direct cost, respectively.

There is an implicit assumption of **no-forgetting** in the semantics of a decision network. The decision maker remembers the past observations and decisions, indicated explicitly by the information and precedence links.

Algorithm 4.3 shows how to use the "Test-Action" decision network for sequential decision making. We will now look at an decision problem modeled with such a network and work through the calculations involved in the network evaluation.

ALGORITHM 4.3
Using a "Test-Action" Decision Network

1. *Evaluate decision network with any available evidence (other than for the Test result).*
   Returns Test decision.
2. *Enter Test decision as evidence.*
3. *If Test decision is 'yes'*
   Run test, get result;
   Enter test result as evidence to network.
   Else
   Enter result 'unknown' as evidence to network.
4. *Evaluate decision network.*
   Returns Action decision.

4.4.2 Real estate investment example

Paul is thinking about buying a house as an investment. While it looks fine externally, he knows that there may be structural and other problems with the house that aren't immediately obvious. He estimates that there is a 70% chance that the house is really

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

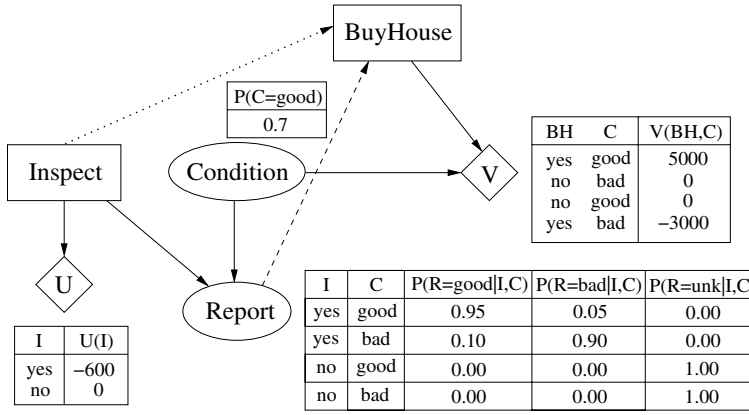


FIGURE 4.8
Decision network for the real estate investment example.

in good condition, with a 30% chance that it could be a real dud. Paul plans to resell the house after doing some renovations. He estimates that if the house really is in good condition (i.e., structurally sound), he should make a $5,000 profit, but if it isn't, he will lose about $3,000 on the investment. Paul knows that he can get a building surveyor to do a full inspection for $600. He also knows that the inspection report may not be completely accurate. Paul has to decide whether it is worth it to have the building inspection done, and then he will decide whether or not to buy the house.

A decision network for this "test-act" decision problem is shown in Figure 4.8. Paul has two decisions to make: whether to do have an inspection done and whether to buy the house. These are represented by the *Inspect* and *BuyHouse* decision nodes, both {*yes,no*} decisions. The condition of the house is represented by the *Condition* chance node, with values {*good, bad*}. The outcome of the inspection is given by node *Report*, with values {*good, bad, unknown*}. The cost of the inspection is represented by utility node *U*, and the profits after renovations (not including the inspection costs) by a second utility node *V*. The structure of this network is exactly that of the general network shown in Figure 4.7.

When Paul decides whether to have an inspection done, he doesn't have any information about the chance nodes, so there are no information links entering the *Inspect* decision node. When he decides whether or not to buy, he will know the outcome of that decision (either a *good* or *bad* assessment, or it will be *unknown*), hence the information link from *Report* to *BuyHouse*. The temporal ordering of his decisions, first about the inspection, and then whether to buy, is represented by the precedence link from *Inspect* to *BuyHouse*. Note that there is a directed path from *Inspect* to *BuyHouse* (via *Report*) so even if there was no explicit precedence link added by the knowledge engineer for this problem, the precedence could be inferred from the rest

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

of the network structure[†].

Given the decision network model for the real-estate investment problem, let's see how it can be evaluated to give the expected utilities and hence make decisions.

4.4.3 Evaluation using a decision tree model

In order to show the evaluation of the decision network, we will use a **decision tree** representation. The nonleaf nodes in a decision tree are either decision nodes or chance nodes and the leaves are utility nodes. The nodes are represented using the same shapes as in decision networks. From each decision node, there is a labeled link for each alternative decision, and from each chance node, there is a labeled link for each possible value of that node. A decision tree for the real estate investment problem is shown in Figure 4.9.

To understand a decision tree, we start with the root node, which in this case is the first decision node, whether or not to inspect the house. Taking a directed path down the tree, the meaning of the link labels are:

- From a decision node, it indicates which decision is made
- From a chance node, it indicates which value has been observed

At any point on a path traversal, there is the same assumption of "no-forgetting," meaning the decison maker knows all the link labels from the root to the current position. Each link from a chance node has a probability attached to it, which is the probability of the variable having that value *given* the values of all the link labels to date. That is, it is a conditional probability. Each leaf node has a utility attached to it, which is the utility given the values of all the link labels on its path from the root.

In our real estate problem, the initial decision is whether to inspect (decision node *I*), the result of the inspection report, if undertaken, is represented by chance node *R*, the buying decision by *BH* and the house condition by *C*. The utilities in the leaves are combinations of the utilities in the *U* and *V* nodes in our decision network. Note that in order to capture exactly the decision network, we should probably include the report node in the "Don't Inspect" branch, but since only the "unknown" branch would have a non-zero probability, we omit it. Note that there is a lot of redundancy in this decision tree; the decision network is a much more compact representation.

A decision tree is evaluated as in Algorithm 4.4. Each possible alternative scenario (of decision and observation combinations) is represented by a path from the root to a leaf. The utility at that leaf node is the utility that would be obtained if that particular scenario unfolded. Using the conditional probabilities, expected utilities associated with the chance nodes can be computed as a sum of products, while the expected utility for a decision assumes that the action returning the highest expected utility will be chosen (shown with **BOLD**, with thicker arcs, in Figure 4.9). These expected utilities are stored at each non-leaf node in the tree (shown in Figure 4.9 in *underlined italics*) as the algorithm works its way recursively back up to the root node.

[†] Some BN software, such as Netica and GeNIe, will add such precedence links automatically.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

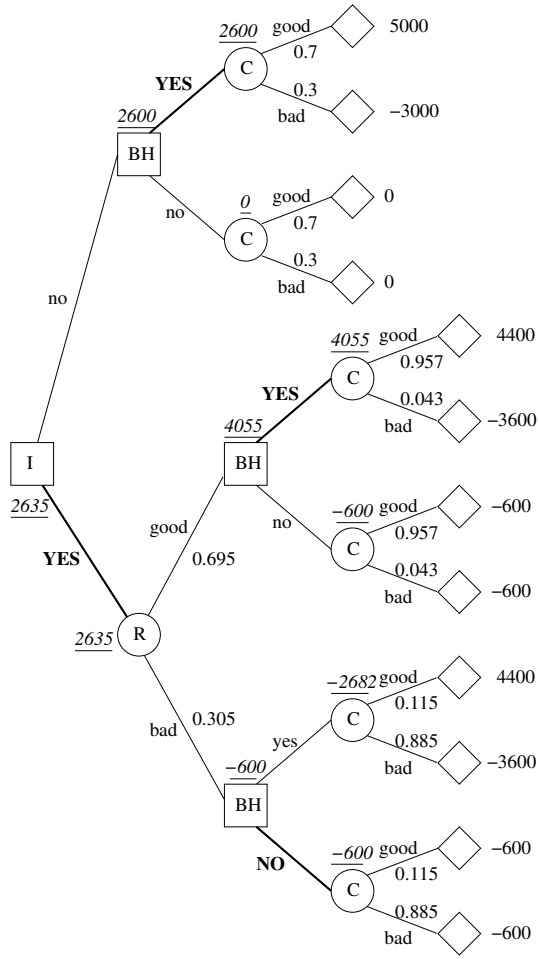


FIGURE 4.9
Decision tree evaluation for real estate investment example.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

ALGORITHM 4.4

Decision Tree Evaluation

1. *Starting with nodes that have only leaves (utility nodes) as children.*
2. *If the node X is a chance node, each outgoing link has a probability and each child has an associated utility. Use these to compute its expected utility*

$$EU(X) = \sum_{C \in Children(X)} U(C) \times P(C)$$

If the node is a decision node, each child has a utility or expected utility attached. Choose the decision whose child has the maximum expected utility and

$$EU(X) = max_{C \in Children(X)}(EU(C))$$

3. *Repeat recursively at each level in the tree, using the computed expected utility for each child.*
4. *The value for the root node is the maximal expected utility obtained if the expected utility is maximized at each decision.*

4.4.4 Value of information

The results of the decision tree evaluation are summarized in Table 4.3 (highest expected utility in each case in **bold**). We can see that the rational decision for Paul is to have the inspection done, as the expected utility (which is the expected profit in this case) is 2635 compared to 2600 if the inspection is not done. Paul's next decision, as to whether to buy the house, will depend on whether he receives a good or bad inspection report. If the report is bad, his decision will be not to buy, with expected utility -600 compared to -2682 for buying. If the report is good, he will go ahead and buy, with expected utility 4055 compared to the same -600 when not buying.

TABLE 4.3

Decisions calculated for the real estate investment problem

Evidence	*Bel(C=good)*	EU(I=yes)	EU(I=no)	Decision
None	0.70	**2635**	2600	I=yes
Given *I=no*		EU(BH=yes)	EU(BH=no)	
Report=unknown	0.70	**2600**	0	BH=yes
Given *I=yes*		EU(BH=yes)	EU(BH=no)	
Report=good	0.957	**4055**	-600	BH=yes
Report=bad	0.115	-2682	**-600**	BH=no

This is a situation where additional information may change a decision, as without the test Paul would just go ahead and buy the house. The decision of whether

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

to gather information is based on the value of the information. The difference in the expected utilities with and without the extra information is called the **expected benefit**. In general,

$$EB(Test) = EU(Test = yes) - EU(Test = no)$$

For our real estate example,

$$\begin{aligned} EB(Inspect) &= EU(Inspect = yes) - EU(Inspect = no) \\ &= 2635 - 2600 = 35 \end{aligned}$$

So, even if there is a cost associated with obtaining additional information (such as the inspection fee), if the expected benefit is greater than zero, the price is worth paying.

4.4.5 Direct evaluation of decision networks

This decision tree evaluation method conveys the underlying ideas of evaluating decision networks containing sequential decisions. We start with the final decision and calculate the expected utility for the various options, given the scenario that has been followed to get there, and choose the decision with the maximum expected utility. This is then repeated for the next-to-last decision, and so on, until the first decision is made.

However, expanding a decision network into a decision tree and using Algorithm 4.4 is very inefficient. Various methods have been developed for evaluating decision networks (see Bibliographic notes in §4.8). Many are similar to the inference algorithms described in Chapter 3, involving compilation into an intermediate structure, and propagation of probabilities *and* expected utilities (see [128, Chapter 7] for details). It is possible to avoid the repetitions of the same calculations that we saw using decision tree evaluation, by taking advantage of the network structure. However, all methods face the same complexity problem.

Thus far, we have only looked at sequential decision networks involving just two decisions. There is no theoretical limit to how many sequential decisions can be made. However, this leads us to the general problem of **planning under uncertainty**, which once again is an exponential search problem, as the number of possible plans is the product of the number of actions considered for each plan step.

4.5 Dynamic Bayesian networks

Bayesian and decision networks model relationships between variables at a particular point in time or during a specific time interval. Although a causal relationship represented by an arc implies a temporal relationship, BNs do not explicitly model

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

temporal relationships between variables. And the only way to model the relationship between the current value of a variable, and its past or future value, is by adding another variable with a different name. We saw an example of this with the fever example earlier in §4.3.5, with the use of the *FeverLater* node. In the decision networks we have seen so far, there is an ad hoc modeling of time, through the use of information and precedence links. When making a sequence of decisions that will span a period of time, it is also important to model how the world *changes* during that time. More generally, it is important to be able to represent and reason about changes over time explicitly when performing such tasks as diagnosis, monitoring, prediction and decision making/planning.

In this section we introduce a generalization of Bayesian networks, called **dynamic Bayesian networks** (DBNs)[‡], that explicitly model change over time. In the following section we will extend these DBNs with decision and utility nodes, to give **dynamic decision networks**, which are a general model for sequential decision making or planning under uncertainty.

4.5.1 Nodes, structure and CPTs

Suppose that the domain consists of a set of n random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, each of which is represented by a node in a Bayesian network. When constructing a DBN for modeling changes over time, we include one node for each X_i for each time step. If the current time step is represented by t, the previous time step by $t-1$, and the next time step by $t+1$, then the corresponding DBN nodes will be:

- Current: $\{X_1^t, X_2^t, \ldots, X_n^t\}$
- Previous: $\{X_1^{t-1}, X_2^{t-1}, \ldots, X_n^{t-1}\}$
- Next: $\{X_1^{t+1}, X_2^{t+1}, \ldots, X_n^{t+1}\}$

Each time step is called a **time-slice**. The relationships between variables in a time-slice are represented by **intra-slice** arcs, $X_i^T \longrightarrow X_j^T$. Although it is not a requirement, the structure of a time-slice does not usually change over time. That is, the relationship between the variables X_1^T, $X_2^T \ldots, X_n^T$ is the same, regardless of the particular T.

The relationships between variables at successive time steps are represented by **inter-slice** arcs, also called **temporal arcs**, including relationships between (i) the same variable over time, $X_i^T \longrightarrow X_i^{T+1}$, and (ii) different variables over time, $X_i^T \longrightarrow X_j^{T+1}$.

In most cases, the value of a variable at one time affects its value at the next, so the $X_i^T \longrightarrow X_i^{T+1}$ arcs are nearly always present. In general, the value of any node at one time can affect the value of any other node at the next time step. Of course, a fully temporally connected network structure would lead to complexity problems, but there is usually more structure in the underlying process being modeled.

[‡]Also called **dynamic belief networks** [237, 206], **probabilistic temporal networks** [69, 70] and **dynamic causal probabilistic networks** [147].

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

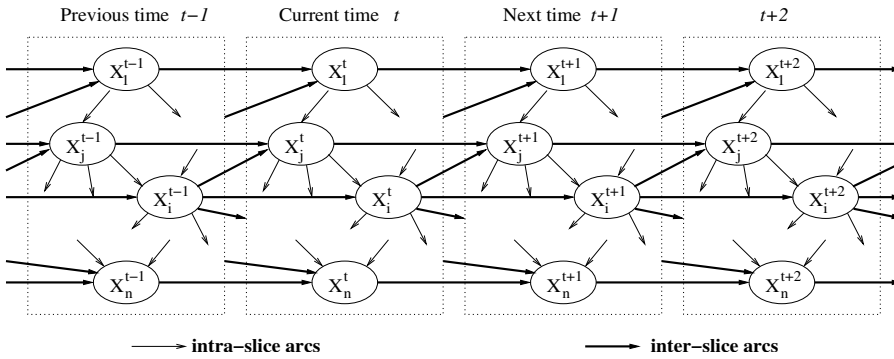


FIGURE 4.10
General structure of a Dynamic Bayesian Network.

Figure 4.10 shows a generic DBN structure, with a sequence of the same static BNs connected with inter-slice arcs (shown with thicker arcs). Note that there are no arcs that span more than a single time step. This is another example of the **Markov assumption** (see §2.2.4), that the state of the world at a particular time depends only on the previous state and any action taken in it.

The relationships between variables, both intra-slice and inter-slice, are quantified by the conditional probability distribution associated with each node.

In general, for node X_i^T with intra-slice parents $Y_1^T, \ldots, Y_m^T$ and inter-slice parents X_i^{T-1} and $Z_1^{T-1}, \ldots, Z_r^{T-1}$, the CPT is

$$P(X_i^T | Y_1^T, \ldots, Y_m^T, X_i^{T-1}, Z_1^{T-1}, \ldots, Z_r^{T-1}).$$

Given the usual restriction that the networks for each time slice are exactly the same and that the changes over time also remain the same (i.e., both the structure and the CPTs are unchanging), a DBN can be specified very compactly. The specification must include:

- Node names
- Intra-slice arcs
- Temporal (inter-slice) arcs
- CPTs for the first time slice t_0 (when there are no parents from a previous time)
- CPTs for $t + 1$ slice (when parents may be from t or $t + 1$ time-slices).

Some BN software packages provide a facility to specify a DBN compactly (see §B.4).

Figure 4.11 shows how we can use a DBN to represent change over time explicitly in the fever example. The patient's flu status may change over time, as there is some chance the patient will get better, hence the $Flu^t \longrightarrow Flu^{t+1}$ arc. Taking aspirin at time t, indicated by A^t, may produce a reaction at the next time

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

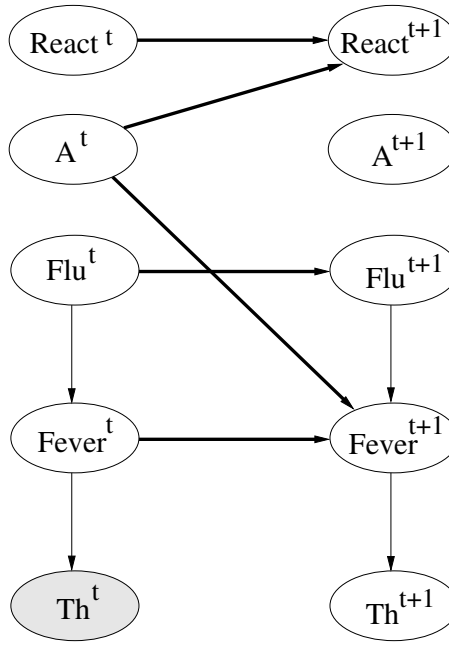


FIGURE 4.11
DBN for the fever example. (Shaded node indicates evidence added.)

step ($A^t \longrightarrow React^{t+1}$) and it may reduce the fever ($A^t \longrightarrow Fever^{t+1}$), although the subsequent fever status depends on the earlier fever status (represented by $Fever^t \longrightarrow Fever^{t+1}$). A person's reaction to aspirin is consistent, hence the arc $React^t \longrightarrow React^{t+1}$.

4.5.2 Reasoning

Given evidence about a set of nodes, $\mathbf{E}_{\{1,t\}}$, from the first time slice up to and including the current time-slice t, we can perform belief updating on the full DBN, using standard BN inference algorithms. This means obtaining new posterior distributions for all the non-evidence nodes, including nodes in the $t+1$ and later time-slices. This updating into the future is called **probabilistic projection**.

However, this type of DBN gets very large, very quickly, especially if the interval between time slices is short. To cope, in most cases the DBN is not extended far into the future. Instead, a fixed size, sliding "window" of time slices is maintained. As the reasoning process moves forward with time, one older time slice is dropped off the DBN, while another is added.

Figure 4.12 shows the progress of a simple two time-slice DBN. This structure can be considered a generic DBN, consisting of state node X, its corresponding observation node Obs, where evidence is added, and an action node A that will

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

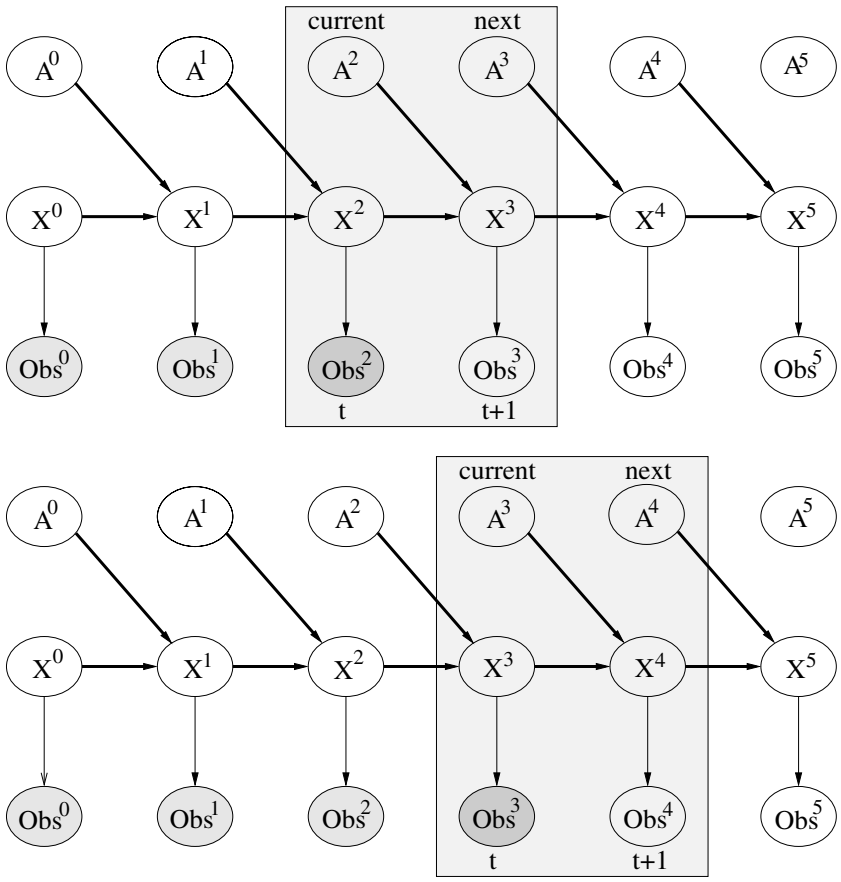


FIGURE 4.12
DBN maintained as a sliding "window" of two time-slices. (Shading indicates evidence node.)

affect the state node at the next time step[§].

This use of a fixed window means that every time we move the window along, the previous evidence received is no longer directly available. Instead, it is summarized taking the current belief for (root) nodes, and making these distributions the new priors. The DBN updating process is given in Algorithm 4.5. Note that the steps of this DBN updating algorithm are exactly those of a technique used in classical control theory, called a **Kalman Filter** [138] (see Bibliographic notes in §4.8).

[§]Note that this is a standard BN node representing a random variable, *not* a decision/action node from a decision network.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

ALGORITHM 4.5
DBN Updating Process

1. **Sliding:** *Move window along.*
2. **Prediction:**
   (a) *We already know* $Bel(X_{t-1}|\mathbf{E}_{\{1,t-1\}})$*, the estimated probability distribution over* X_{t-1}.
   (b) *Calculate the predicted beliefs,* $\widehat{Bel}(X_t|\mathbf{E}_{\{1,t-1\}})$,
3. **Rollup:**
   (a) *Remove time-slice* $t-1$.
   (b) *Use the predictions for the* t *slice as the new prior by setting* $P(X)$ *to* $\widehat{Bel}(X_t|\mathbf{E}_{\{1,t-1\}})$.
4. **Estimation:**
   (a) *Add new observations* $\mathbf{E}_t$.
   (b) *Calculate* $Bel(X_t|\mathbf{E}_{\{1,t\}})$*, the probability distribution over the current state.*
   (c) *Add the slice for* $t+1$.

4.5.3 Inference algorithms for DBNs

Exact clustering algorithms can be applied to DBNs, particularly if the inference is restricted to two time-slices¶. Unfortunately, there normally is a cluster containing all the nodes in a time slice with inter-slice connections, so the clusters become unwieldy. The intuitive reason for this is that even if there are no intra-slice arcs between two nodes, they often become correlated through common ancestors. A version of the junction tree algorithm has been developed for DBNs [149]. This method takes advantage of the DBN structure by creating a junction tree for each time slice and performing updating for time slices up to and including the current time-slice using inter-tree message passing. For probabilistic projection into future time-slices, a sampling method is effective as there is no evidence yet for these future time slices.

If the DBN clusters get too large, approximate algorithms using stochastic simulation (described in §3.6) are usually suitable. As we discussed in §3.6.5, stochastic simulation methods are more effective when the evidence is at the root nodes, while the typical DBN structure involves the modeling of one or more state variables, each of which is observed with a possibly noisy sensor. This structure is shown in Figure 4.12; we can see that in these models evidence is at the leaves.

¶This does not mean that the beliefs being maintained are exact, of course; since past evidence is being summarized, the beliefs are inexact.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

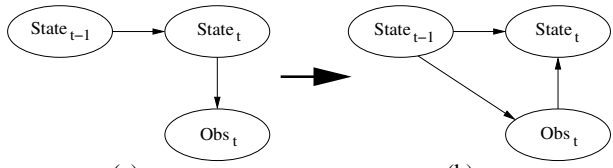


FIGURE 4.13
(a) Original network; (b) new structure after arc reversal process.

One solution to this problem with stochastic simulation is to reverse the arcs to evidence nodes, as proposed by Shachter [250], and then use the stochastic simulation algorithms. This **arc reversal** method (see §6.3.1.1) ensures that the evidence is at the root nodes, while maintaining the same overall joint probability distribution. It requires the addition of arcs to maintain the conditional independencies. A simple example is shown in Figure 4.13. The disadvantage is that we get a more complex structure that does not model the causal relationships.

Many other approximate algorithms for DBN inference have been proposed, including variations of stochastic simulation (e.g., [140]), filtering methods (e.g., [198]) and ignoring weak dependencies in the stochastic process (e.g., [31, 133, 148]). Unfortunately, most of these are not implemented in BN software packages (see §B.4).

4.6 Dynamic decision networks

Just as Bayesian networks can be extended with a temporal dimension to give DBNs, so can decision networks be extended to give **dynamic decision networks** (DDNs). Not only do they represent explicitly how the world changes over time, but they model general sequential decision making.

Figure 4.14 shows a generic DDN structure, for making a sequence of n decisions D^t, D^{t+1}, ..., D^{t+n}, from time t into the future. The temporal sequencing of the decision is represented by the precedence link (shown as a dotted line). The single chance node X determines the utility, while Obs is the observation node for which evidence will be added before each subsequent decision; this sequencing is represented by the information link (shown as a dashed line). Note that in Figure 4.14 the decision making will maximize the expected utility in n time steps. Another alternative would be to have a utility node at each time slice, in which case the decision making will maximize the cumulative expected utility from time t to $t + n$.

The DDN structure for the fever example is shown in Figure 4.15.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

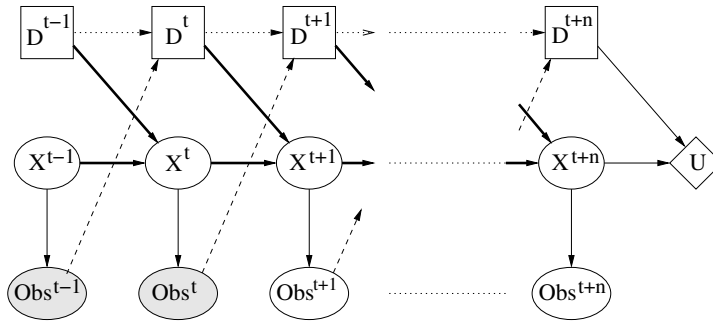


FIGURE 4.14
A generic DDN for a sequence of n decisions, to maximize expected utility at time $t + n$.

4.6.1 Mobile robot example

The robot's task is to detect and track a moving object, using sonar and vision sensor information, given a global map of the office floor environment. The robot must also continually reassess its own position (called localization) to avoid getting lost. At any point in time, the robot can make observations of its position with respect to nearby walls and corners and of the target's position with respect to the robot.

This deceptively simple, yet interesting, problem of a mobile robot that does both localization and tracking (a slightly simplified version of the one presented in [70]) can be modeled with a DDN as follows. The nodes S_T and S_R represent the locations of the target and the robot, respectively. The decision node is *M*, representing the robot's movement actions options. The nodes O_R and O_T represent the robot's observations of its own and the target's location, respectively. The overall utility is the weighted sum over time of the utility at each step, U^T, which is a measure of the distance between the robot and its target.

The DDN using these nodes is shown in Figure 4.16. The temporal arcs $S_T^t \rightarrow S_T^{t+1}$ indicate that the target's next position is related to its previous position; the actual model of possible movement is given implicitly in the CPT for S_T^{t+1}. The temporal arcs $S_R^t \rightarrow S_R^{t+1}$ and $M^t \rightarrow S_R^{t+1}$ indicate that the robot's own location depends on its previous position and the movement action taken. Both observation nodes have only intra-slice connections. The robot's observation of own position O_R^t depends only on its actual position ($S_R^t \rightarrow O_R^t$), while its ability to observe the position of the target will depend on both the target's actual location plus its own position (represented by $S_R^t \rightarrow O_T^t$ and $S_T^t \rightarrow O_T^t$), as it has to be close enough for its sensors to detect the target.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

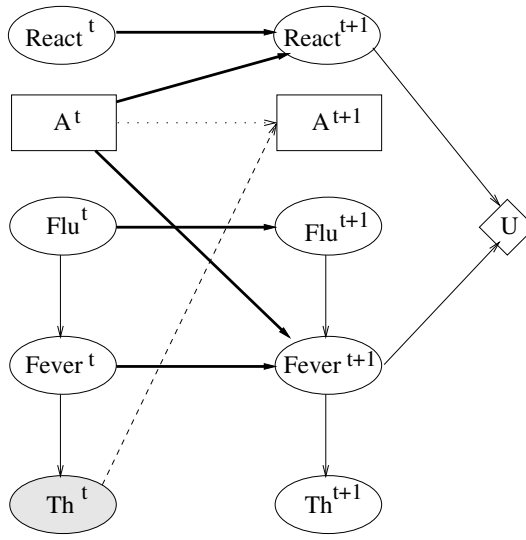


FIGURE 4.15
DDN structure for the fever example.

4.7 Summary

In order to make decisions, we must be able to take into account preferences between different outcomes. Utility theory provides a way to represent and reason with preferences. The combination of utility theory and probability theory gives a framework for decision making, where a rational being should make choices that maximize her or his expected utility. Extending Bayesian networks with decision nodes and utility nodes gives us decision networks (also called influence diagrams). These include an explicit representation of ordering between information and decisions, and decisions and decisions, and allow us to make isolated or sequential decisions. By adding a temporal dimension to Bayesian networks, we get dynamic Bayesian networks (DBNs), which allow us explicitly to model and reason about changes over time. These in turn can be extended with utility and decision nodes, to give dynamic decision networks (DDNs). DDNs run into complexity problems when used for general planning, in which case special purpose planning representations and algorithms may be more useful.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

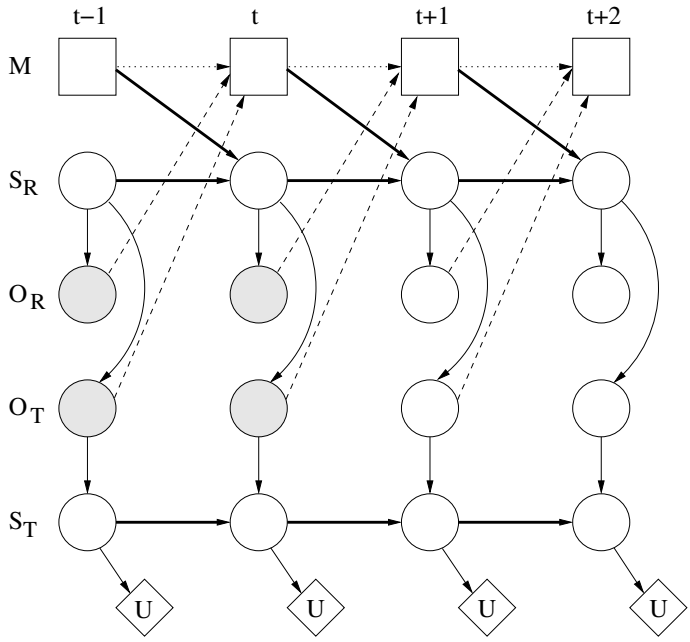


FIGURE 4.16

A dynamic decision network for the mobile robot example. (From Dean, T. and Wellman, M.P. *Planning and Control*. Morgan Kauffman Publishers, San Mateo, CA. 1991. With permission.)

4.8 Bibliographic notes

Influence diagrams were originally developed as a compact representation of decision trees for decision analysis. The basic concepts were developed by a group at SRI [191], and were formally introduced by Howard and Matheson [115].

Jensen in his recent text [128] gives a detailed presentation of one method for evaluation decision network, and summarizes other approaches. The term "DBN" was first coined by Dean and Kanazawa [69]; other early DBN research in the AI community was undertaken by Nicholson [206] and Kjærulff [147].

The chapter by Kevin Murphy on DBNs for Michael Jordan's forthcoming text book, *An Introduction to Probabilistic Graphical Models*, gives a very comprehensive survey of DBNs, their connections to hidden Markov models (HMMs) and state space models, and exact and approximate inference algorithms (including filtering approaches). Kjaerulff [149] describes the dHugin computation scheme for DBN inference, which includes window expansion and reduction, forward and back interjunction tree propagation and forward sampling for probabilistic projection.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

4.9 Problems

Utility

Problem 1

You have 20 pounds of cheese. We offer to make a gamble with you: tossing a fair coin, if it comes up heads, we'll give you 130 more pounds of cheese; if it comes up tails, we'll take your 20 pounds of cheese. What is the expected value of the gamble? Assuming that you maximize your expected utility and that you refuse this gamble, what can you infer about your utility function — that is, how many utiles (basic units of your utility) is a pound of cheese worth?

Problem 2

You have 20 utiles worth of chocolate. We offer to make a gamble with you: tossing some coin, if it comes up heads, we'll give you 130 more utiles worth of chocolate; if it comes up tails, we'll take your 20 utiles worth of chocolate. Assuming that you maximize your expected utility and that you refuse this gamble, what can you infer about the probability you give to the coin landing heads?

Modeling

Problem 3

Robert is trying to decide whether to study hard for the Bayesian Artificial Intelligence exam. He would be happy with a good mark (e.g., a High Distinction) for the subject, but he knows that his mark will depend not only on how hard he studies but also on how hard the exam is and how well he did in the Introduction to Artificial Intelligence subject (which indicates how well prepared he is for the subject).

Build a decision network to model this problem, using the following steps.

- Decide what chance nodes are required and what values they should take.
- This problem should only require a single decision node; what will it represent?
- Decide what the casual relationships are between the chance nodes and add directed arcs to reflect them.
- Decide what chance nodes Robert's decision may effect and add arcs to reflect that.
- What is Robert's utility function? What chance nodes (if any) will it depend on? Does it depend on the decision node? Will a single utility node be sufficient? Update the decision network to reflect these modeling decisions.
- Quantify the relationships in the network through adding numbers for the CPTs (of chance nodes) and the utility table for the utility node. Does the number of parameters required seem particularly large? If so, consider how you might reduce the number of parameters.

© 2004 by CRC Press LLC
© 2004 by Chapman & Hall/CRC Press LLC

- Once you have built your model, show the beliefs for the chance nodes and the expected utilities for the decisions before any evidence is added.
- Add evidence and see how the beliefs and the decision change, if at all.
- If you had an information link between the evidence node and the decision node, view the decision table.
- Perform your own decision tree evaluation for this problem and confirm the numbers you have obtained from the software.

Problem 4

Think of your own decision making problem involving reasoning with evidence and uncertainty. Write down a text description of the problem, then model it with a decision network using similar steps to the previous problem. Make the problem sufficiently complex that your network has at least 3 chance nodes (though a single decision node and a single utility node will be enough for a first modeling effort, unless you feel particularly adventurous).

Problem 5

Julia's manufacturing company has to decide whether to go ahead with the production of a new product. Her analysis indicates that the future profit will depend on a combination of the quality of the product and the market demand for it. Before the final decision is made, she has two other possible courses of action. One is to undertake further product development, to make the product quality better before it goes into production. The other is to do more market research to determine the likely demand. She could also choose to do both. Of course both product development and market research will cost money. And she has to be careful about delaying too long, as she knows that the first product of this kind that becomes available will corner the market.

Build a decision network to model this sequential decision problem.

1. Decide what chance nodes, decision nodes (hint: there should be three) and utility node(s) you need.
2. Add the main network links, then the information and precedence links. Note that you need to investigate two possible orderings for the pre-production decisions.
3. The problem description does not provide much in the way of quantitative information for parameterizing your model; so choose initial parameters that seem reasonable.
4. What is the *expected benefit* of additional marketing? And of further product development?

© 2004 by Chapman & Hall/CRC Press LLC

Dynamic Bayesian networks (DBNs)

Problem 6

Peter wants to do simple monitoring of his local weather. Suppose that the main weather types are: sunny, cloudy, showers, rain and snow. He knows that the weather changes from one day to the next in certain patterns, depending on the season. He also has access to the local weather forecast for the following day, and he has been monitoring this for long enough to know that the local forecasters get it right about 70% of the time (though slightly worse for cloudy and showery days).

Build a DBN to model this problem. You should use a BN software package that specifically supports DBNs (e.g., Netica, Hugin; see Appendix B).

Dynamic decision networks (DDNs)

Problem 7

Cate brought some shares for $1000. Each day, she has to decide whether to sell, or keep them. Her profit will depend on the price she gets for them. Each morning, she gets the previous day's closing price, and she rings her sister Megan and gets her opinion as to whether the price will go up or down today. She also knows that most of the time, the price of these shares only move within a certain range from one day to the next.

Build a dynamic decision network to model this problem.

© 2004 by Chapman & Hall/CRC Press LLC

5

Applications of Bayesian Networks

5.1 Introduction

In the previous three chapters, we have seen in detail how Bayesian and decision networks can be used to represent and reason with uncertainty. We have used simple illustrative examples throughout, which, together with some of the modeling problems set as exercises, give some indication of the type and scope of the problems to which Bayesian networks may be applied. In this chapter we provide more detailed examples of some BN applications which we have personally developed. By moving beyond elementary examples we can provide a better insight into what is involved in designing and implementing Bayesian models and the range of applications for which they can be useful.

We begin with a very brief survey of BN applications developed by others, especially medical applications, in order to show some of the variety of uses and BN models which are to be found in the literature. We then describe three of our own applications:

1. The first is a game playing application, specifically poker, a domain that is rich in uncertainty from a number of sources: physical randomization through shuffling, incomplete information through the opponent's hidden cards and limited knowledge about the opponent's strategies. In this application, the BN is used to compute the probability of winning, with a decision network combined with randomization strategies in a hybrid system to make betting decisions.

2. The second application is ambulation monitoring and fall detection, which illustrates the use of DBNs in medical monitoring and also includes typical sensor modeling within a Bayesian network.

3. Finally, we look at an argument generation application, where BNs for both normative domain modeling and user modeling are embedded in a wider architecture integrating the BNs with semantic networks and an attention focusing mechanism.

© 2004 by Chapman & Hall/CRC Press LLC

5.2 A brief survey of BN applications

5.2.1 Types of reasoning

As we discussed in Chapter 2, a key reasoning task using BNs is that of updating the posterior distribution of one or more variables, given evidence. Depending on what evidence is available, the probabilistic belief updating can be performing one of a number of tasks.

1. **Diagnosis**. Example: *Which illness do these symptoms indicate?*
2. **Monitoring/control**. Example: *Is the patient's glucose level stable, or does extra insulin need to be given?*
3. **Forward prediction**.

   Such predictions may be factual (that is, based on evidence) or hypothetical (such as predicting the effect of an intervention). For example, when considering the question *Will the patient survive the proposed operation?* a factual prediction might be

   > *Based on the results of the X-ray, MRI and blood tests, will the patient survive the proposed operation?*

   An example hypothetical prediction might be

   > *If the patient is given anticoagulant X, will the patient's chances of survival be improved?*

As we saw in the previous chapter, once we extend the network with decision and utility nodes, we can use it for decision making and planning. Examples include *Which treatment plan carries the least risk of failure?* and *Which sequence of actions will maximize the patient's quality of life?*

5.2.2 BN structures for medical problems

Medicine has undoubtedly been the most popular application area to date for Bayesian networks. As a complex domain where much knowledge is implicitly held by experienced medical practitioners, it has long been a target of expert systems. A connection between BN research and medical applications was established in the 1980s by researchers such as David Heckerman and Eric Horvitz who were in the Stanford Ph.D./M.D. program. The appeal of BNs for this application area lies in their ability to explicitly model causal interventions, to reason both diagnostically and predictively and the visual nature of the representation, which facilitates their use in explanation.

We have already looked at some simple medical reasoning examples in the preceding chapters. The simplest tree-structured network for diagnostic reasoning is the so-called naive Bayes model, as shown in Figure 5.1(a), where *Disease (D)* node

© 2004 by Chapman & Hall/CRC Press LLC

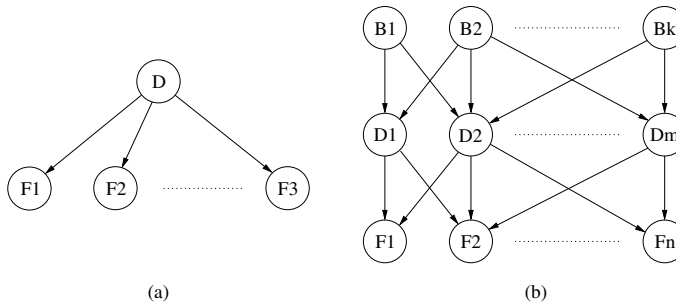


FIGURE 5.1
Generic BN structures for medical diagnosis: (a) naive Bayes model; (b) multiply-connected network.

has values for each of the diseases under consideration, while the F nodes represent "findings," which encompass both symptoms and test results. This network reflects two unrealistic assumptions: that the patient can have only a single disease and that the symptoms are independent of each other given the disease.

A more realistic, but more complex, model is the multiply-connected network of Figure 5.1(b). Here there is a Boolean node for each disease under consideration, while the B nodes represent background information such as the age and sex of the patient, whether they are a smoker, or have been exposed to pollution. However, in practice, this structure is likely to be too complex, requiring us to specify probabilistically the combined effect of every disease on each finding.

The network structure in Figure 5.1(b) is essentially that developed in the QMR-DT project [256, 190], a probabilistic version of the frame-based CPCS knowledge base for internal medicine. The QMR-DT network had this two-level structure. The problem of complexity was ameliorated by making it a binary noisy-or model (see §7.4.2). This was done by assuming the effect of a disease on its symptoms and test results is independent of other diseases and independent of other findings. One version of the QMR-DT network (described in [222]) had 448 nodes and 908 arcs, including 74 background nodes (which they called "predisposing factors") needing prior probabilities, while the remaining nodes required probabilities to be assessed for each of their values. In total more than 600 probabilities were estimated, a large but not an unreasonable number given the scope of the application. Performing exact inference on networks of this size is generally not feasible. Initial work on the application of likelihood weighting to this medical diagnosis problem is described in [255], while other approximate methods for QMR-DT are presented in [121].

The ALARM network for monitoring patients in intensive care [17], shown in Figure 5.2, is a sparsely connected BN consisting of 37 nodes and 42 arcs (the number of values for each node is shown next to the node name). This network is often used as a benchmark in the BN literature. Clearly, it does not map neatly into the generic medical diagnosis structure given above, and it does not provide a template for building other medical monitoring or diagnostic BNs.

© 2004 by Chapman & Hall/CRC Press LLC

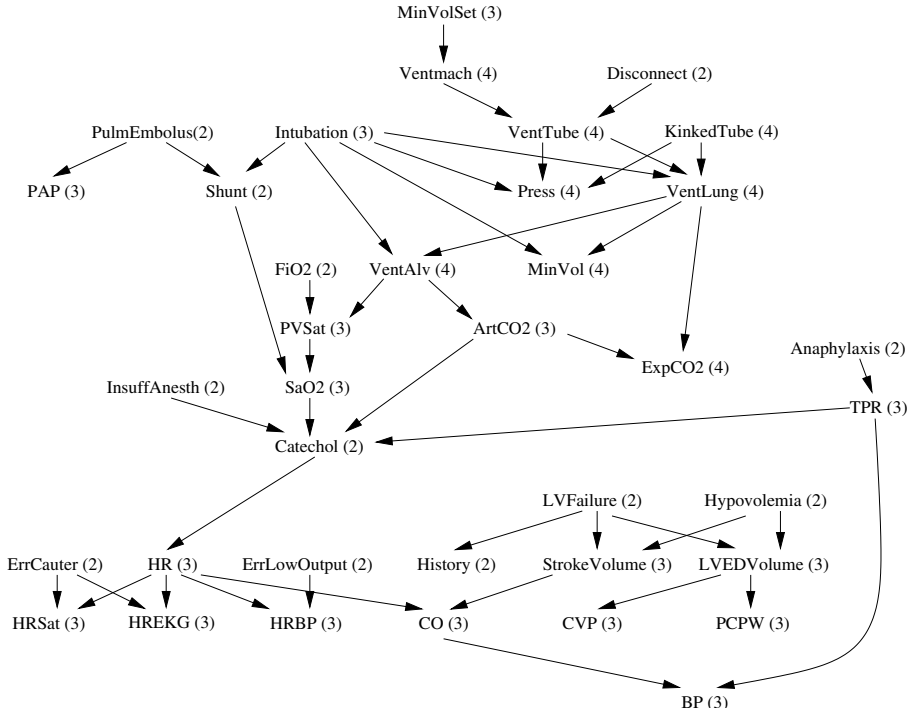


FIGURE 5.2
The ALARM BN for ICU monitoring.

5.2.3 Other medical applications

PATHFINDER is a diagnostic expert system for lymph-node diseases, with PATHFINDER IV containing a BN component [102]; it has been converted into a commercially available decision support system for surgical pathologists, INTELLIPATH. The ODIN research group at Aalborg University, Denmark, has been involved with the development of several medical applications. MUNIN [9] is a multiply connected BN of about 1000 nodes for diagnosing neuromuscular disorders. Dynamic BN medical applications include one for glucose prediction and insulin dose adjustment [8] and one for sleep apnea [64]. Van der Gaag and her group have worked on a BN for diagnosing oesophageal cancer [281]. BNs have also been used for mammography [37] and diagnosing liver disorder [212]. Finally, the PROMEDAS medical decision support tool [223] uses Bayesian networks, automatically compiling both the network and an interface from the underlying medical database (which currently covers the areas of endocrinology and lymphoma diagnostics).

© 2004 by Chapman & Hall/CRC Press LLC

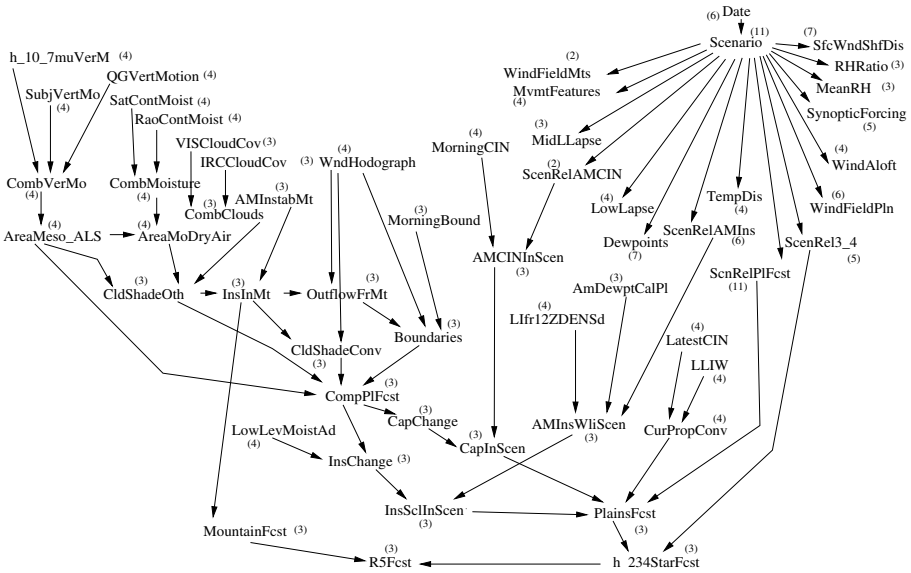


FIGURE 5.3
The Hailfinder BN.

5.2.4 Non-medical applications

Hailfinder is a well-known system for forecasting severe summer hail in northeastern Colorado [2]. The BN is shown in Figure 5.3. Again, the number of values for each node is shown next to the node name in parentheses. The GEMS (Generator Expert Monitoring System) [195] project is an example of a system where BNs succeeded when a rule-based system did not [162].

Vista [111] is a decision-theoretic system used at NASA Mission Control Center in Houston. It uses Bayesian networks to interpret live telemetry and provide advice on possible failures of the space shuttle's propulsion systems. Vista recommends actions of the highest expected utility, taking into account time criticality.

BNs have been used in a number of user modeling applications: software users in the Lumiere system [112], players in an adventure game [4] and various student models in intelligent tutoring systems [50, 49, 108, **?**, **?**] (see also §11.3).

Oil price forecasting is an application that has been modeled with an ordinary BN [3] and with a DBN [63]. Other early DBN applications included robot navigation and map learning [70] (see also §4.6.1), monitoring robot vehicles [207, 208] and traffic monitoring in both [225] and the BATmobile project for monitoring an automated vehicle traveling on a freeway [85]. A fragment of the BATmobile DBN is shown in Figure 5.4. The number of values for each node is shown in parentheses, the inter-slice arcs are shown in thicker solid lines, while the intra-slice arcs are thinner. The intraslice arcs for $t + 1$ are the same as for t and hence omitted. The observation nodes are shaded.

© 2004 by Chapman & Hall/CRC Press LLC

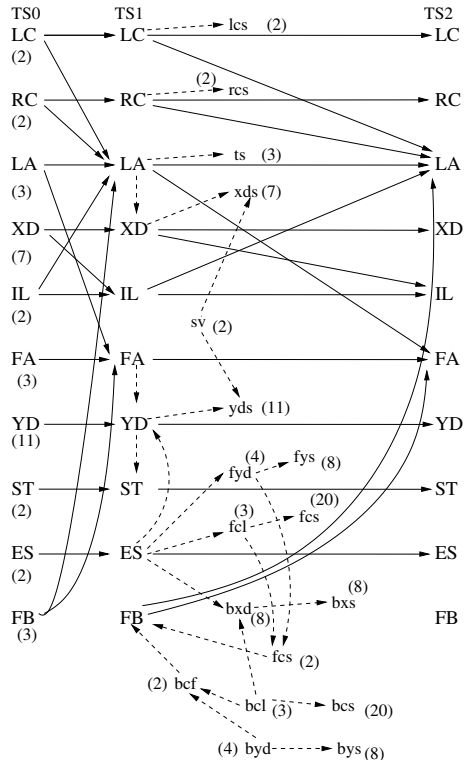


FIGURE 5.4
The BATmobile BN.

BNs have been used for biological applications such as modeling the biological processes of a water purification plant [131] and for deciding on the amount of fungicides to be used against attack of mildew in wheat [126]. More recently there has been much interest in using BNs for ecological applications [80, 163, 26, 183, 13].

5.3 Bayesian poker

We will now describe the application of Bayesian networks to a card game, five-card stud poker*. Poker is an ideal vehicle for testing automated reasoning under

*We have worked on this application occasionally since 1993. The version described here is an improved version of that presented in [160], with some structural changes and the use of utilities to make the betting decision. Details of the system evolution and evaluation are in Chapter 9, as an example of the knowledge engineering process.

© 2004 by Chapman & Hall/CRC Press LLC

uncertainty. It introduces uncertainty through physical randomization by shuffling and through incomplete information about opponents' hands. Another source of uncertainty is the limited knowledge of opponents, their tendencies to bluff, play conservatively, reveal weaknesses, etc. Poker being a game all about betting, it seems most apt to employ a Bayesian network to compute the odds.

5.3.1 Five-card stud poker

Poker is a non-deterministic **zero-sum game** with imperfect information. A game is zero-sum if the sum of the winnings across all players is zero, with the profit of one player being the loss of others. The long-term goal of all the players is to leave the table with more money than they had at the beginning. A poker session is played in a series of games with a standard deck of 52 playing cards. Each card is identified by its suit and rank. There are four suits: ♣ Clubs, ♢ Diamonds, ♡ Hearts and ♠ Spades. The thirteen card ranks are (in increasing order of importance): Deuce (2), Three (3), Four (4), Five (5), Six (6), Seven (7), Eight (8), Nine (9), Ten (T), Jack (J), Queen (Q), King (K) and Ace (A).

In five-card stud poker, after an **ante** (an initial fixed-size bet), players are dealt a sequence of five cards, the first down (hidden) and the remainder up (available for scrutiny by other players). Players bet after each upcard is dealt, in a clockwise fashion, beginning with the best hand showing. The first player(s) may `PASS` — make no bet, waiting for someone else to open the betting. Bets may be `CALLED` (matched) or `RAISED`, with up to three raises per round. Alternatively, a player facing a bet may `FOLD` her or his hand (i.e., drop out for this hand). After the final betting round, among the remaining players, the one with the strongest hand wins in a "showdown." The strength of poker hand types is strictly determined by the probability of the hand type appearing in a random selection of five cards (see Table 5.1). Two hands of the same type are ranked according to the value of the cards (without regard for suits); for example, a pair of Aces beats a pair of Kings.

TABLE 5.1
Poker hand types: weakest to strongest

Hand Type	Example	Probability
Busted	A♣ K♠ J♢ 10♢ 4♡	0.5015629
Pair	2♡ 2♢ J♠ 8♣ 4♡	0.4225703
Two Pair	5♡ 5♣ Q♠ Q♣ K♣	0.0475431
Three of a Kind	7♣ 7♡ 7♠ 3♡ 4♢	0.0211037
Straight (sequence)	3♠ 4♣ 5♡ 6♢ 7♠	0.0035492
Flush (same suit)	A♣ K♣ 7♣ 4♣ 2♣	0.0019693
Full House	7♠ 7♢ 7♣ 10♢ 10♣	0.0014405
Four of a Kind	3♡ 3♠ 3♢ 3♣ J♠	0.0002476
Straight Flush	3♠ 4♠ 5♠ 6♠ 7♠	0.0000134

© 2004 by Chapman & Hall/CRC Press LLC

The basic decision facing any poker player is to estimate one's winning chances accurately, taking into account how much money will be in the pot if a showdown is reached and how much it will cost to reach the showdown. Assessing the chance of winning is not simply a matter of the probability that the hand you have now, if dealt out to the full five cards, will end up stronger than your opponent's hand, if it is also dealt out. Such a pure combinatorial probability of winning is clearly of interest, but it ignores a great deal of information that good poker players rely upon. It ignores the "tells" some poker players have (e.g., facial tics, fidgeting); it also ignores current opponent betting behavior and the past association between betting behavior and hand strength. Our **Bayesian Poker Player** (BPP) doesn't have a robot's sensory apparatus, so it can't deal with tells, but it does account for current betting behavior and learns from the past relationship between opponents' behavior throughout the game and their hand strength at showdowns.

5.3.2 A decision network for poker

BPP uses a series of networks for decision making throughout the game.

5.3.2.1 Structure

The network shown in Figure 5.5 models the relationship between current hand type, final hand type, the behavior of the opponent and the betting action. BPP maintains a separate network for each of the four rounds of play (the betting rounds after two, three, four and five cards have been dealt). The number of cards involved in the current and observed hand types, and the conditional probability tables for them, vary for each round, although the network structure remains that of Figure 5.5.

The node *OPP_Final* represents the opponent's final hand type, while *BPP_Final* represents BPP's final hand type; that is, these represent the hand types they will have after all five cards are dealt. Whether or not BPP will win is the value of the Boolean variable *BPP_Win*; this will depend on the final hand types of both players. *BPP_Final* is an observed variable after the final card is dealt, whereas its opponent's final hand type is observed only after play ends in a showdown. Note that the two final hand nodes are *not* independent, as one player holding certain cards precludes the other player holding the same cards; for example, if one player has four-of-a-kind aces, the other player cannot.

At any given stage, BPP's current hand type is represented by the node *BPP_Current* (an observed variable), while *OPP_Current* represents its opponent's current hand type. Since BPP cannot observe its opponent's current hand type, this must be inferred from the information available: the opponent's upcard hand type, represented by node *OPP_Upcards*, and the opponent's actions, represented by node *OPP_Action*. Note that the existing structure makes the assumption that the opponent's action depends *only* on its current hand and does not model such things as the opponent's confidence or bluffing strategy.

Although the *BPP_Upcards* node is redundant, given *BPP_Current*, this node is included to allow BPP to work out its *opponents* estimate of winning (required for

© 2004 by Chapman & Hall/CRC Press LLC

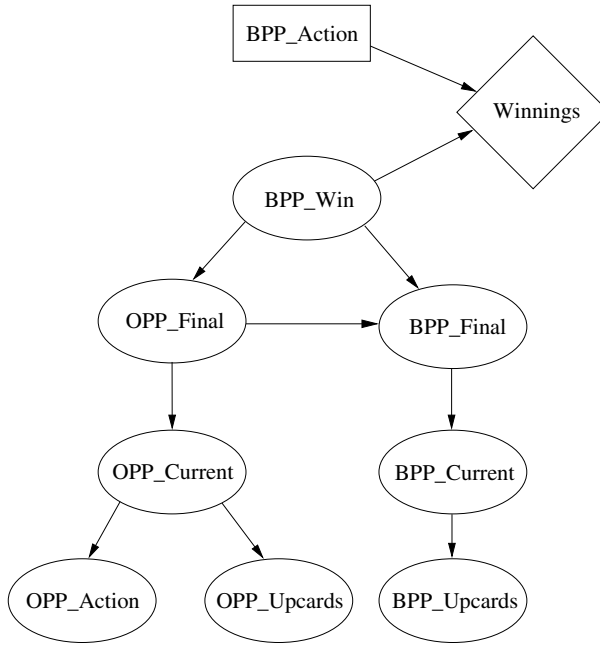


FIGURE 5.5
A decision network for poker.

bluffing, see §5.3.4). In this situation, *BPP_Upcards* becomes the observation node as the opponent only knows BPP's upcards.

5.3.2.2 Node values

The nodes representing hand types are given values which sort hands into strength categories. In principle, we could provide a distinct hand type to each distinct poker hand by strength, since there are finitely many of them. That finite number, however, is fairly large from the point of view of Bayesian network propagation; for example, there are already 156 differently valued Full Houses. BPP recognizes 24 types of hand, subdividing busted hands into busted-low (9 high or lower), busted-medium (10 or J high), busted-queen, busted-king and busted-ace, representing each paired hand separately, and with the 7 other hand types as listed in Table 5.1. The investigation of different refinements of hand types is described in §11.2. Note that until the final round, *BPP_Current*, *OPP_Current* and *OPP_Upcards* represent partial hand types (e.g., three cards to a flush, instead of a flush).

The nodes representing the opponent's actions have three possible values, *bet/raise, pass/call, fold*.

© 2004 by Chapman & Hall/CRC Press LLC

5.3.2.3 Conditional probability tables

There are four action probability tables $P_i(OPP\_Action|OPP\_Current)$, corresponding to the four rounds of betting. These report the conditional probabilities per round of the actions — folding, passing/calling or betting/raising — given the opponent's current hand type. BPP adjusts these probabilities over time, using the relative frequency of these behaviors per opponent. Since the rules of poker do not allow the observation of hidden cards unless the hand is held to showdown, these counts are made only for such hands, undoubtedly introducing some bias.

The four CPTs $P(OPP\_Upcards|OPP\_Current)$ give the conditional probabilities of the opponent having a given hand showing on the table when the current hand (including the hidden card) is of a certain type. The same parameters were used for $P(BPP\_Upcards|BPP\_Current)$. The remaining CPTs are the four giving the conditional probability for each type of partial hand given that the final hand will be of a particular kind, used for both *OPP_Current* and *BPP_Current*. These CPTs were estimated by dealing out 10,000,000 hands of poker.

5.3.2.4 Belief updating

Given evidence for *BPP_Current*, *OPP_Upcards* and *OPP_Action*, belief updating produces belief vectors for both players' final hand types and, most importantly, a posterior probability of BPP winning the game.

5.3.2.5 Decision node

Given an estimate of the probability of winning, it remains to make betting decisions. Recall that decision networks can be used to find the optimal decisions which will maximize an expected utility. For BPP, the decision node *BPP_Action* in Figure 5.5 represents the possible betting actions *bet/raise, pass/call, fold*, while the utility we wish to maximize is the amount of winnings BPP can accumulate.

5.3.2.6 The utility node

The utility node, *Winnings*, measures the dollar value BPP expects to make based on the possible combinations of the states of the parent nodes (*BPP_Win* and *BPP_Action*). For example, if BPP decided to fold with its next action, irrespective of whether or not it would have won at a showdown, the expected future winnings will be zero as there is no possibility of future loss or gain in the current game. On the other hand, if BPP had decided to bet and it were to win at a showdown, it would make a profit equal to the size of the final pot F_{bet}, minus any future contribution made on its behalf B_{bet}. If BPP bet and lost, it would make a loss equal to any future contribution it made towards the final pot, $-B_{bet}$. A similar situation occurs when BPP decides to pass, but with a differing expected total contribution B_{pass} and final pot F_{pass}. This information is represented in a utility table within the *Winnings* node, shown in Table 5.2.

The amount of winnings that can be made by BPP is dependent upon a number of factors, including the number of betting rounds remaining R, the size of the betting

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 5.2
Poker action/outcome utilities

BPP_Action	*BPP_Win*	Utility
Bet	Win	$F_{bet} - B_{bet}$
Bet	Lose	$-B_{bet}$
Pass	Win	$F_{pass} - B_{pass}$
Pass	Lose	$-B_{pass}$
Fold	Win	0
Fold	Lose	0

unit B and the current size of the pot C. The expected future contributions to the pot by both BPP and OPP must also be estimated (see Problem 5.8).

The decision network then uses the belief in winning at a showdown and the utilities for each (*BPP_Win*, *BPP_Action*) pair to calculate the expected winnings (EW) for each possible betting action. Folding is always considered to have zero EW, since regardless of the probability of winning, BPP cannot make any future loss or profit.

5.3.3 Betting with randomization

This decision network provides a "rational" betting decision, in that it determines the action that will maximize the expected utility if the showdown is reached. However, if a player invariably bets strongly given a strong hand and weakly given a weak hand, other players will quickly learn of this association; this will allow them to better assess their chances of winning and so to maximize their profits at the expense of the more predictable player. So BPP employs a mixed strategy that selects an action with some probability based on the EW of the action. This ensures that while most of the time BPP will bet strongly when holding a strong hand and fold on weak hands, it occasionally chooses a locally sub-optimal action, making it more difficult for an opponent to construct an accurate model of BPP's play.

Betting curves, such as that in Figure 5.6, are used to randomize betting actions. The horizontal axis shows the difference between the EW of folding and calling[†] (scaled by the bet size); the vertical axis is the probability with which one should fold. Note that when the difference is zero ($EW\,(call) - EW\,(fold) = 0$), BPP will fold randomly half of the time.

Once the action of folding has been rejected, a decision needs to be made between calling and raising. This is done analogously to deciding whether to fold, and is calculated using the difference between the EW of betting and calling.

[†]More exact would be to compute the differential EW between folding and not folding, the latter requiring a weighted average EW for *pass/call* and for *bet/raise*. We use the EW of calling as an approximation for the latter. Note also that we refer here to calling rather than passing or calling, since folding is not a serious option when there is no bet on the table, implying that if folding is an option, passing is not (one can only pass when there is no bet).

© 2004 by Chapman & Hall/CRC Press LLC

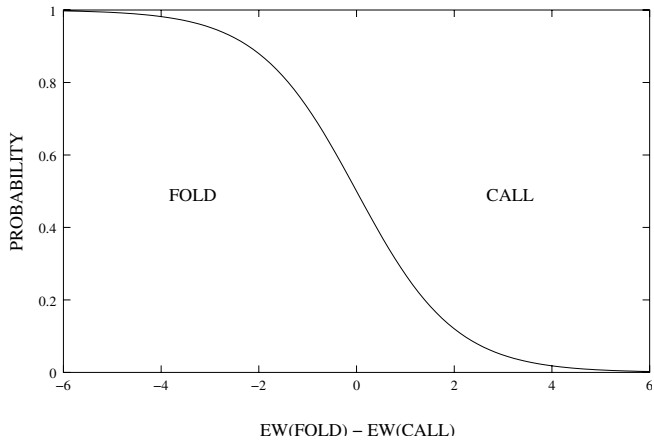


FIGURE 5.6
Betting curve for folding.

The betting curves were generated with exponential functions, with different parameters for each round of play. Ideal parameters will select the optimal balance between deterministic and randomized play by stretching or squeezing the curves along the horizontal axis. If the curves were stretched horizontally, totally random action selection could result, with the curves selecting either alternative with probability 0.5. On the other hand, if the curves were squeezed towards the center, a deterministic strategy would ensue, with the action with the greatest EW always being selected. The current parameters in use by BPP were obtained using a stochastic search of the parameter space when running against an earlier version of BPP.

5.3.4 Bluffing

Bluffing is the intentional misrepresentation of the strength of one's hand. You may over-represent that strength (what is commonly thought of as bluffing), in order to chase opponents with stronger hands out of the round. You may equally well underrepresent the strength of your hand ("sandbagging") in order to retain players with weaker hands and relieve them of spare cash. These are tactical purposes behind almost all (human) instances of bluffing. On the other hand, there is an important strategic purpose to bluffing, as von Neumann and Morgenstern pointed out, namely "to create uncertainty in [the] opponent's mind" [288, pp. 188-189]. In BPP this purpose is already partially fulfilled by the randomization introduced with the betting curves. However, that randomization occurs primarily at the margins of decision making, when one is maximally uncertain whether, say, calling or raising is optimal over the long run of similar situations. Bluffing is not restricted to such cases; the need is to disguise from the opponent what the situation is, whether or not the optimal response is known. Hence, bluffing is desirable for BPP as an action in addition to the use of randomizing betting curves.

© 2004 by Chapman & Hall/CRC Press LLC

The current version of BPP uses the notion of a "bluffing state." First, BPP works out what its opponent will believe is BPP's chance of winning, by performing belief updating given evidence for *BPP_Upcards*, *OPP_Upcards* and *OPP_Action*. Given this belief is non-zero, it is worth considering bluffing. In which case BPP has a low probability of entering the bluffing state in the last round of betting, whereupon it will continue to bluff (by over-representation) until the end of the round.

5.3.5 Experimental evaluation

BPP has been evaluated experimentally against two automated opponents:

1. A probabilistic player that estimates its winning probability for its current hand by taking a large sample of possible continuations of its own hand and its opponent's hand, then making its betting decision using the same method as BPP;
2. A simple rule-based opponent that incorporated plausible maxims for play (e.g., fold when your hand is already beaten by what's showing of your opponent's hand).

BPP was also tested against earlier versions of itself to determine the effect of different modeling choices (see §11.2). Finally, BPP has been tested against human opponents with some experience of poker who were invited to play via telnet. In all cases, we used BPP's cumulative winnings as the evaluation criterion. BPP performed significantly better than both the automated opponents, was on a par with average amateur humans, but lost fairly comprehensively to an expert human poker player. We are continuing occasional work on BPP. Currently, this includes converting it to play Texas Hold'em, which will allow us to test directly with the other significant computer poker project of Billings and company (see §5.7). Further discussion of BPP's limitations and our ongoing work is given in §11.2.

5.4 Ambulation monitoring and fall detection

Here we present our dynamic belief network (DBN) model for ambulation monitoring and fall detection, an interesting practical application of DBNs in medical monitoring, based on the version described in [203].

5.4.1 The domain

The domain task is to monitor the stepping patterns of elderly people and patients recovering from hospital. Actual falls need to be detected, causing an alarm to be raised. Also, irregular walking patterns, stumbles and near falls are to be identified. The monitoring is performed using two kinds of sensors: foot-switches, which report

© 2004 by Chapman & Hall/CRC Press LLC

steps, and a mercury sensor, which is triggered by a change in height, such as going from standing upright to lying horizontally, and so may indicate a fall. Timing data for the observations is also given.

Previous work in this domain performed fall detection with a simple state machine [66], developed in conjunction with expert medical practitioners. The state machine attempts to solve the fall detection problem with a set of *if-then-else* rules. This approach has a number of limitations. First, there is no representation of degrees of belief in the current state of the person's ambulation. Second, there is no distinction between actual states of the world and observations of them, and so there is no explicit representation of the uncertainty in the sensors [208]. Possible sensor errors include:

- **False positives**: the sensor wrongly indicates that an action (left, right, lowering action) has occurred (also called **clutter**, **noise** or **false alarms**).
- **False negatives**: an action occurred but the sensor was not triggered and no observation was made (also called **missed detection**).
- **Wrong timing data**: the sensor readings indicate the action which occurred; however the time interval reading is incorrect.

5.4.2 The DBN model

When developing our DBN model, a key difference from that state machine approach is that we focus on the causal relationships between domain variables, making a clear distinction between observations and actual states of the world. A DBN for the ambulation monitoring and fall detection problem is given in Figure 5.7. In the rest of this section, we describe the various features of this network in such a way as to provide an insight into the network development process.

5.4.2.1 Nodes and values

When considering how to represent a person's walking situation, possibilities include the person being stationary on both feet, on a step with either the left or right foot forward or having fallen and hence off his or her feet. F represents this, taking four possible values: $\{$*both, left, right, off*$\}$. The Boolean event node *Fall* indicates whether a fall has taken place between time slices. Fall warning and detection relies on an assessment of the person's walking pattern. The node S maintains the person's status and may take the possible values $\{$*ok, stumbling*$\}$. The action variable, A, may take the values $\{$*left, right, none*$\}$. The last value is necessary for the situation where a time slice is added because the mercury sensor has triggered (i.e., the person has fallen) but no step was taken or a foot switch false positive was registered.

There is an observation node for each of the two sensors. The foot switch observations are essentially observations on step actions, and are represented by AO, which contains the same values as the action node. The mercury sensor trigger is represented by the Boolean node M. The time between sensor observations is given by T. Given the problems with combining continuous and discrete variables (see §9.3.2.4), and the limitations of the sensor, node T takes discrete values representing tenths of

© 2004 by Chapman & Hall/CRC Press LLC

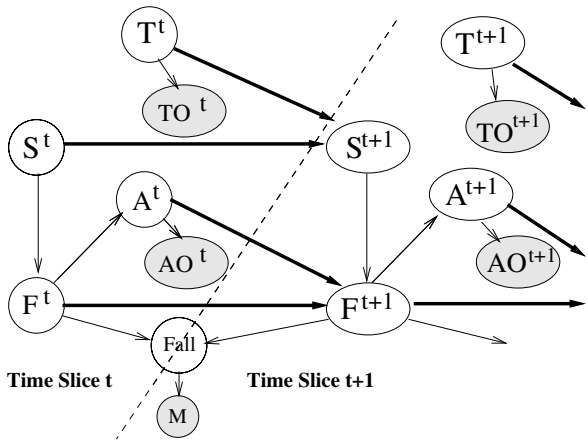


FIGURE 5.7
DBN for ambulation monitoring and fall detection.

seconds. While the fact that there is no obvious upper limit on the time between readings may seem to make it difficult to define the state space of the *T* node, recall that a monitoring DBN is extended to the next time slice when a sensor observation is made, say n tenths of a second later. If we ignored error in time data, we could add a *T* with a single value *n*. In order to represent the uncertainty in the sensor reading, we say it can take values within an interval around the sensor time reading that generates the addition of a new time slice to the DBN. If there is some knowledge of the patient's expected walking speed, values in this range can be added also. The time observation node, *TO*, has the same state space as *T*. For each new time slice a copy of each node is added. The possibility of adding further time slices is indicated by the dashed arcs.

5.4.2.2 Structure and CPTs

The CPTs for the state nodes *A*, *F*, *Fall* and *S* are given in Table 5.3. The model for walking is represented by the arcs from F^t to A^t and from F^t, A^t and S^t to F^{t+1}.

We assume that normal walking involves alternating left and right steps. Where the left and right are symmetric, only one combination is included in the table. We have priors for starting on both feet (r) or already being off the ground (s). By definition, if a person finishes on a particular foot, it rules out some actions; for example, if F^{t+1} = *left*, the action could not have been *right*. These zero conditional probability are omitted from the table. The CPT for F^{t+1} for the conditioning cases where S^{t+1} = *stumbling* is exactly the same as for *ok* except the p and q probability parameters will have lower values, representing the higher expectation of a fall. If there are any variations on walking patterns for an individual patient, for example if one leg was injured, the DBN can be customized by varying the probability parameters, s, r, p_i, q_i, u, v and w and removing the assumption that left and right are completely

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 5.3
Ambulation monitoring DBN CPTs

P(F_0=*left*||*right*|)	= (1-r-s)/2	
P(F_0=*both*|)	= r	
P(F_0=*off*|)	= s	
P(A=*left*|F=*right*)	= u	alternate feet
P(A=*right*|F=*right*)	= v	hopping
P(A=*none*|F=*right*)	= 1-u-v	stationary
P(A=\{*left*||*right*\}|F=*both*)	= $w/2$	start with left or right
P(A=*none*|F=*both*)	= 1-w	stationary
P(A=*none*|F=*off*)	= 1	can't walk when off feet
P(F^{t+1}=*left*|F^t=*right*,A^t=left,S^{t+1}=*ok*)	= p_1	successful alternate step
P(F^{t+1}=*both*|F^t=*right*,A^t=left,S^{t+1}=*ok*)	= q_1	half-step
P(F^{t+1}=*off*|F^t=*right*,A^t=left,S^{t+1}=*ok*)	= 1-p_1-q_1	fall prob
P(F^{t+1}=*left*|F^t=*left*,A^t=left,S^{t+1}=*ok*)	= p_2	successful hop
P(F^{t+1}=*both*|F^t=*left*,A^t=left,S^{t+1}=*ok*)	= q_2	half-hop
P(F^{t+1}=*off*|F^t=*left*,A^t=left,S^{t+1}=*ok*)	= 1-p_2-q_2	fall prob
P(F^{t+1}=*left*|F^t=*both*,A^t=left,S^{t+1}=*ok*)	= p_3	successful first step
P(F^{t+1}=*both*|F^t=*both*,A^t=left,S^{t+1}=*ok*)	= q_3	unsuccessful first step
P(F^{t+1}=*off*|F^t=*both*,A^t=left,S^{t+1}=*ok*)	= 1-p_3-q_3	fall prob
P(F^{t+1}=*left*|F^t=*left*,A^t=none,S^{t+1}=*ok*)	= p_4	
P(F^{t+1}=*off*|F^t=*left*,A^t=none,S^{t+1}=*ok*)	= 1-p_4	fall when on left foot
P(F^{t+1}=*right*|F^t=*right*,A^t=none,S^{t+1}=*ok*)	= p_5	
P(F^{t+1}=*off*|F^t=*right*,A^t=none,S^{t+1}=*ok*)	= 1-p_5	fall when on right foot
P(F^{t+1}=*both*|F^t=*both*,A^t=none,S^{t+1}=*ok*)	= p_6	
P(F^{t+1}=*off*|F^t=*both*,A^t=none,S^{t+1}=*ok*)	= 1-p_6	fall when on both feet
P(F^{t+1}=*off*|F^t=*off*,A^t=left,S^{t+1}=*any*)	= 1	no "get up" action
P(*Fall*=T | F^{t+1}=*off*,F^t=\{*left*||*right*||*both*\})	= 1	from upright to ground
P(*Fall*=F | F^{t+1}=*any*,F^t=*off*)	= 1	can't fall if on ground
P(S^{t+1}=*ok*|T^t=t)	= 1	if $t \geq$ y
P(S^{t+1}=*stumbling*|T^t=t)	= 1	if $t <$ y
P(M=*T*|*Fall*=*T*)	= pos_1	ok
P(M=*F*|*Fall*=*T*)	= 1-pos_1	missing
P(M=*F*|*Fall*=*F*)	= neg_1	ok
P(M=*T*|*Fall*=*F*)	= 1-neg_1	false alarm
P(AO=*left*|A=*left*)	= pos_2	ok
P(AO=*right*|A=*right*)	= pos_2	ok
P(AO=*right*|A=*left*)	= (1-pos_2)/2	wrong
P(AO=*left*|A=*right*)	= (1-pos_2)/2	wrong
P(AO=*none*|A=*left*)	= (1-pos_2)/2	missing
P(AO=*none*|A=*right*)	= (1-pos_2)/2	missing
P(AO=*none*|A=*none*)	= neg_2	ok
P(AO=*left*|A=*none*)	= (1-neg_2)/2	false alarm
P(AO=*right*|A=*none*)	= (1-neg_2)/2	false alarm
P(TO=x|T=x)	= pos_3	ok, $y \neq x$
P(TO=y|T=x)	= $1 - pos_3/m$-1,	ok, $y \neq x$

Parameter set used for case-based evaluation results: s = 0.0, r = 0.9, u = 0.7, v = 0.2, w = 0.1, p_1 = 0.6, q_1 = 0.3, p'_1 = 0.5, q'_1 = 0.4 p_2 = 0.6, q_2 = 0.3, p'_2 = 0.5, q'_2 = 0.4, p_3 = 0.6, q_3 = 0.3, p'_3 = 0.5, q'_3 = 0.4, p_4 = 0.95, p'_4 = 0.85, p_5 = 0.95, p'_5 = 0.85, p_6 = 0.9, p'_6 = 0.8, pos_1 = 0.9, pos_2 = 0.9, pos_3 = 0.9, neg_1 = 0.95, neg_2 = 0.95.

© 2004 by Chapman & Hall/CRC Press LLC

symmetric. The fall event node *Fall* has F^t and F^{t+1} as predecessors; a fall only occurs when the subject was on his or her feet to start with ($F^t \neq$ *off*), and finishes off their feet (F^{t+1} = *off*). Note that the fall event node does not really belong to either the t or $t+1$ time slice; it actually represents the **transition event** between the two time slice.

In this DBN model, the value of walking status node S is determined solely by the time between sensor readings. Also, the T node has no predecessors; its prior can be modified, based on sensor observations over time, to reflect an individual's ordinary walking speed. Note that adding an arc from T^t to T^{t+1} would allow a representation of the expectation that the walking pace should remain fairly constant.

The three observation nodes all have single parents, the variable they are sensing; their CPTs are shown in Table 5.3. Note that the confidence in an observation is given by some value based on a model of the sensor's performance and is empirically obtainable; pos is the sensitivity of the positive sensor data and neg is the specificity of the negative sensor data (or, 1-neg is the probability of ghost data). We make the default assumption that missing or wrong data are equally likely — this need not be the case and can be replaced by any alternative plausible values.

Obviously, this DBN is only one possible model for the ambulation monitoring and fall detection problem and has limitations; for example, it does not handle the cases where both foot switches provide data at the same time or the patient sits down. It also does not handle missing time data.

5.4.3 Case-based evaluation

Let us now look at the behavior of this DBN modeled with the set of parameters given in Table 5.3. For this evaluation, we entered a sequence of evidence that were considered a reasonable series of simulated observations from the sensors. Table 5.4 shows the new beliefs after every new piece of evidence was added. For reasons of space, we left out the initial S_0 node and the T_2 and TO_2 nodes from the model and do not give all the beliefs, especially if they are uniform or otherwise obvious. Probabilities have been rounded to 4 decimal places. The evidence sequence added and the effect on the beliefs was as follows.

No evidence added: All beliefs are based on the parameters. Belief in an immediate fall is small, bel($Fall_0$ = *T*)=0.1194, but chance of being off feet in 2 steps is higher, bel(F_0=*T*)=0.2238.

***TO*$_0$ set to *t*$_1$:** This increases the probability that the person is stumbling, that is, bel(S_1 = *stumbling*)=0.9, which in turn slightly increases the belief in a fall, bel($Fall_0$ = *T*) = 0.1828.

***AO*$_0$ set to *left*:** Foot switch information leads to a change in the belief in the initial starting state; bel(F_0=*right*) has increased from 0.05 to 0.2550, reflecting the model of alternate foot steps.

***M*$_0$ set to *F*:** The negative mercury trigger data makes it very unlikely that a fall occurred, bel($Fall_0$=*T*)=0.0203.

***TO*$_0$ set to *t*$_2$:** "Resetting" of the original timing data makes it less likely the person was stumbling, reducing the belief in a fall, bel($Fall_0$=*T*) = 0.0098.

© 2004 by Chapman & Hall/CRC Press LLC

***M_0* set to *T*:** However, resetting the mercury trigger data makes a fall most probable, bel($Fall_0$=*T*)=0.6285, although there is still the chance that the sensor has given a wrong reading.

***M_1* set to *F*, *TO_1* set to *t_4*, *AO_1* set to *none*:** No action, and no mercury trigger data, confirms the earlier fall, bel($Fall_0$=*T*)=0.7903, since if the person is already on the ground they won't take a left or right step.

5.4.4 An extended sensor model

The DBN described thus far provides a mechanism for handling (by implicitly rejecting) certain inconsistent data. It represents the underlying assumptions about the data uncertainty; however it does not provide an explanation of *why* the observed sensor data might be incorrect. We adapt an idea that has been used in other research areas, that of a moderating or invalidating condition[‡]. We can represent a defective sensor by the addition of a sensor status node *SS* or invalidating node [208]. This new node can take the possible values such as {*working, defective*}. Each sensor status node becomes a predecessor of the corresponding observation node, and there is a connection between sensor status nodes across time slices (see Figure 5.8).

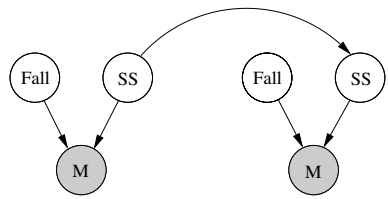


FIGURE 5.8
Extending DBN with sensor status node.

The priors for *SS* explicitly represent how likely it is that the sensor is working correctly; in many cases, this can be obtained from data. The CPTs for the *SS* node given in Table 5.5 show how we can model intermittent or persistent faults. The parameter X is related to the consistency of the fault. The degradation factor d is the probability that a sensor which has been working during the previous time interval has begun to respond defectively. It is based on a model of the expected degradation of the sensor and is a function of the time between sensor readings. It is worth noting that having only the two states doesn't allow us to distinguish between different fault manifestations, which is the source of the query for the CPT numbers when the sensor is defective (see Problem 5.8).

[‡]In the social sciences, the term "moderator" is used for an alternative variable that "moderates" the relationship between other variables [298]. A similar idea has been used in AI research [301, 5].

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 5.4
Changing beliefs for the ambulation monitoring DBN for an evidence sequence

Node	Value	Updated beliefs given particular evidence						
		None	TO_0=t_1	AO_0=*left*	M_0=F	TO_0=t_2	M_0=T	SET
T_0	t_1	0.25	0.9000	0.9000	0.8914	0.0305	0.0535	0.0616
	t_2	0.25	0.0333	0.0333	0.0361	0.9026	0.8812	0.8736
	t_3	0.25	0.0333	0.0333	0.0361	0.0334	0.0326	0.0323
	t_4	0.25	0.0333	0.0333	0.0361	0.0334	0.0326	0.0323
TO_0	t_1	0.25	**1.0**	**1.0**	**1.0**	0.0	0.0	0.0
	t_2	0.25	0.0	0.0	0.0	**1.0**	**1.0**	**1.0**
F_0	*left*	0.05	0.05	0.0870	0.0860	0.0856	0.0964	0.0911
	right	0.05	0.05	**0.2550**	0.2717	0.2515	0.2792	0.2767
	both	0.90	0.90	0.6581	0.6422	0.6628	0.6244	0.6322
	off	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A_0	*left*	0.09	0.09	0.6403	0.6483	0.6453	0.6047	0.5427
	right	0.09	0.09	0.0356	0.0360	0.0359	0.0336	0.0302
	none	0.82	0.82	0.3241	0.3156	0.3188	0.3617	0.4271
AO_0	*left*	0.1265	0.1265	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	right	0.1265	0.1265	0.0	0.0	0.0	0.0	0.0
	none	0.7470	0.7470	0.0	0.0	0.0	0.0	0.0
$Fall_0$	T	**0.1194**	**0.1828**	0.1645	**0.0203**	**0.0098**	**0.6285**	**0.7903**
	F	0.8806	0.8173	0.8355	0.9797	0.9902	0.3715	0.2096
M_0	T	0.1515	0.2053	0.1898	0.0	0.0	**1.0**	**1.0**
	F	0.8485	0.7947	0.8102	**1.0**	**1.0**	0.0	0.0
S_1	*ok*	0.75	0.1	0.1	0.1086	0.9695	0.9465	0.9383
	stum'g	0.25	**0.9**	0.9	0.8914	0.0305	0.0535	0.0617
F_1	*left*	0.0638	0.0425	0.2737	0.3208	0.5120	0.1921	0.0340
	right	0.0638	0.0425	0.0168	0.0197	0.0303	0.0114	0.0020
	both	0.7530	0.7322	0.5451	0.6391	0.4478	0.1680	0.1736
	off	0.1194	**0.1828**	0.1645	0.0203	0.0098	0.6285	0.7903
T_1	t_1	0.25	0.25	0.25	0.25	0.25	0.25	0.0326
	t_4	0.25	0.25	0.25	0.25	0.25	0.25	0.9006
TO_1	t_4	0.25	0.25	0.25	0.25	0.25	0.25	**1.0**
A_1	*left*	0.0950	0.0749	0.0938	0.1099	0.1461	0.0548	0.0035
	right	0.0950	0.0749	0.2222	0.2605	0.3869	0.1451	0.0092
	none	0.8090	0.8502	0.6841	0.6296	0.4670	0.8001	0.9872
AO_1	*left*	0.1308	0.1137	0.1297	0.1434	0.1741	0.0966	0.0
	right	0.1308	0.1137	0.2389	0.2714	0.3788	0.1734	0.0
	none	0.7383	0.7730	0.6315	0.5851	0.4671	0.7301	**1.0**
$Fall_1$	T	0.1044	0.0975	0.0959	0.1124	0.1099	0.0412	0.0024
	F	0.8956	0.9025	0.9041	0.8876	0.8901	0.9588	0.9976
M_1	T	0.1387	0.1329	0.1315	0.1455	0.1434	0.0850	0.0
	F	0.8612	0.8671	0.8685	0.8545	0.8566	0.9150	**1.0**
S_2	*ok*	0.75	0.75	0.75	0.75	0.75	0.75	0.9673
	stum'g	0.25	0.25	0.25	0.25	0.25	0.25	0.0327
F_2	*left*	0.0673	0.0531	0.0898	0.1053	0.1472	0.0552	0.0258
	right	0.0673	0.0531	0.1335	0.1565	0.2291	0.08594	0.0076
	both	0.6415	0.6136	0.5164	0.6055	0.5040	0.1891	0.1740
	off	**0.2238**	0.2802	0.2603	0.1327	0.1197	0.6698	0.7927

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 5.5
New CPTs with sensor status node added

P(*M=T*|*Fall=T,SS=work*)	= 1	ok
P(*M=F*|*Fall=F,SS=work*)	= 1	ok
P(*M=F*|*Fall=T,SS=def*)	= ???	missing
P(*M=T*|*Fall=F,SS=def*)	= ???	false alarm
P($SS_{t+1} = def | SS_t = work$)	= 1 - d	
P($SS_{t+1} = def | SS_t = def$)	= X	

5.5 A Nice Argument Generator (NAG)

Our Nice Argument Generator (NAG) applies Bayesian modeling techniques to a signal problem in artificial intelligence: how to design a computer system which can understand the natural language arguments presented to it and present its own good arguments in reply[§]. This attracted us as an interesting and difficult problem. Furthermore, there is good presumptive reason to believe that Bayesian methods would find a useful role in analyzing and generating arguments, since the considerable majority of live arguments in the wild (e.g., on newspaper opinion pages) are inductive and uncertain, unlike mathematical proofs, for example.

In order to argue well one must have a grasp of both the normative strength of the inferences that come into play and the effect that the proposed inferences will have on the audience. NAG is designed to embody both aspects of good argument — that is, to present arguments that are persuasive for an intended audience and also are as close to normatively correct as such persuasiveness allows, which are just the arguments we dub **nice**.

With such a nice system in mind, we incorporated two Bayesian network models within NAG:

- A **normative model**, for evaluating normative correctness
- A **user model**, for evaluating persuasive effect on the user

Currently, both networks are built by hand for each argumentative domain, using the BN software Netica. The normative model should ideally incorporate as many relevant items of knowledge as we can muster and the best understanding of their relationships. The user model should ideally reflect all that we might know of the specific audience that is inferentially relevant: their prior beliefs and their inferential tendencies and biases. We should gather such information about users by building an explicit profile of the user prior to initiating debate, as well as refining the network during debate. As a matter of mundane fact, after building an initial network for the user model, NAG is (thus far) limited to representing some of the simpler cognitive

[§]To be sure, the name of our program emphasizes the generation side of argumentation, neglecting the understanding side; but that was, of course, in view of finding an acronym.

© 2004 by Chapman & Hall/CRC Press LLC

heuristics and biases that cognitive psychologists have established to be widespread, such as the failure to use base rate information in inductive reasoning [158].

5.5.1 NAG architecture

Given a normative and a user model, some argumentative context and a goal proposition, NAG is supposed to produce a nice argument supporting the goal. In other words, NAG should produce an argument which will be effective in bringing the user to a degree of belief in the goal proposition within a target range, while simultaneously being "normative" — i.e., bringing the degree of belief in the goal in the normative model within its target. In general terms, NAG does this by building up a subgraph of both the user network and the normative network, called the **argument graph**, which it tests in both the normative and user models. The argument graph is expanded piecemeal until it satisfies the criteria in both models — or, again, until it is clear that the two targets will not be mutually satisfied, when NAG abandons the task.

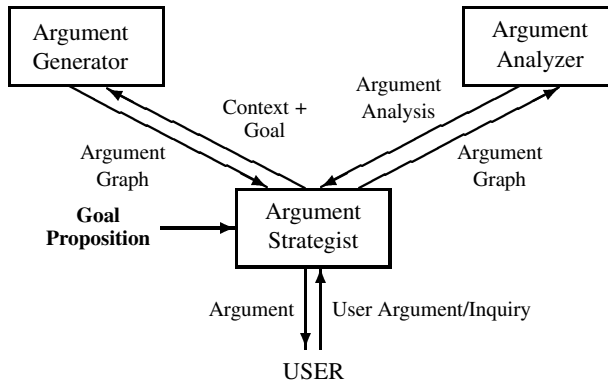


FIGURE 5.9
NAG architecture.

In somewhat more detail, NAG is composed of three modules as shown in Figure 5.9.

- The **Argument Strategist** governs the argumentation process. In the first instance it receives a goal proposition and context and then invokes the Generator to initiate the construction of an argument¶.
- The **Argument Generator** uses the argumentative context and the goal to construct an initial argument graph.

¶ The Strategist may instead receive a user argument, initiating an analysis and rebuttal mode of operation, which we will not describe here.

© 2004 by Chapman & Hall/CRC Press LLC

- The **Argument Analyzer** receives the argument graph from the Strategist and tests the effect of the argument on the goal proposition in both the user and the normative models, using Bayesian network propagation, with the propositions found in the context as the premises (the observed variables) and the goal as the query node.

Assuming that the Analyzer discovers that the goal is insufficiently supported in either the user or the normative model, the Strategist must employ the Generator in an attempt to expand (or contract) its argument graph. It uses guidance from the Analyzer to do this, such as information about which inferential steps are weak or strong in either model. If some of the premises are unhelpful, they may be trimmed from the argument graph, while new nodes that might connect the remaining premises to the goal may be added. The argument graph will then be returned to the Analyzer for the next round. This alternating invocation of the Generator and Analyzer continues until either some argument graph is generated which brings the original goal proposition into the target ranges for strength of belief, the Strategist is unable to fix a problem reported by the Analyzer, some operating constraint is violated which cannot be overcome (e.g., the overall complexity of the argument cannot be reduced to an acceptable level) or time runs out. Finally, the Strategist will report the argument to the user, if a suitable one has been produced.

5.5.2 Example: An asteroid strike

An example might help you get a feel for the process. Consider the Bayesian network shown in Figure 5.10, representing a goal proposition

> *A large iridium-rich asteroid struck Earth 65-million-years BC.* (G)

preceded by the preamble

> *Around 65-million-years BC the dinosaurs, large reptiles that dominated the Earth for many millions of years, became extinct.* (P2) *There was also a massive extinction of marine fauna.* (P3) *At about the same time, a world-wide layer of iridium was laid down on the Earth's crust.* (P1) *It may be that at the same time mass dinosaur migrations led to the spread of disease.* (P4)

The shaded nodes indicate those which are the potential premises in an argument (labeled Pn) and the goal proposition (G). In our example, this Bayesian network represents both the normative and the user models: they do not happen to differ structurally. During the iterative process of building the argument graph in Figure 5.11 the node P4, referring to an alternative explanation for the dinosaur extinction, is dropped because it is unhelpful; the node P3 is dropped because it is unnecessary. It takes two iterations to find the final argument graph, connecting the retained premises with the goal and leading to a posterior belief in the goal node within both target ranges. This argument will then be presented to the user. Note that the arcs in Figure 5.11 are all causal; the final argument proceeds from the existence of the effects to the existence of a cause.

© 2004 by Chapman & Hall/CRC Press LLC

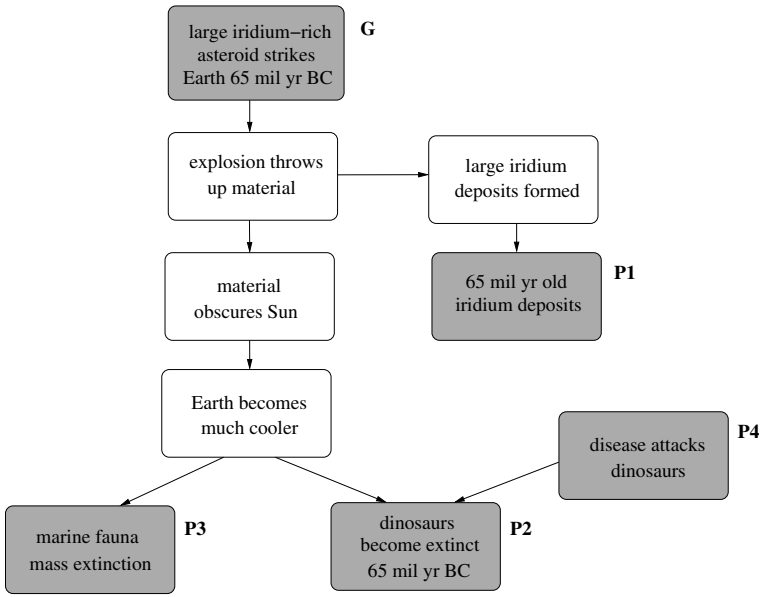


FIGURE 5.10
Asteroid Bayesian network.

5.5.3 The psychology of inference

A paradox of inference. A problem may have occurred to the gentle reader: if the premises of the argument are fixed by the argumentative context (excepting the omission of those which turn out to be unhelpful) and if the premises of the argument are identical with those nodes $\mathbf{O}$ which are observed (i.e., activated for Bayesian network propagation), then the subsequent addition and deletion of nodes in the argument graph is pointless, since Bayesian network propagation rules will guarantee that the goal node G will subsequently take $P(G|\mathbf{O})$ regardless of what ends up in the argument graph. Either the premises immediately and adequately support the goal or they do not, and no further twiddling with any "argument graph" is going to change that.

If this were a correct observation about NAG, then the idea of using NAG to generate any kind of argument would have to be abandoned. Arguments — normative, persuasive or otherwise — are not coextensive with the set of nodes (propositions) which have their conditional probabilities updated given new observations, since that set of nodes, on standard propagation algorithms, is the set of all nodes. What we do with NAG, however, is quite different: we model the psychology of inference. In particular, we model attentional processes during argumentation, and we propagate our Bayesian networks only to the extent that the nodes are being attended to.

© 2004 by Chapman & Hall/CRC Press LLC

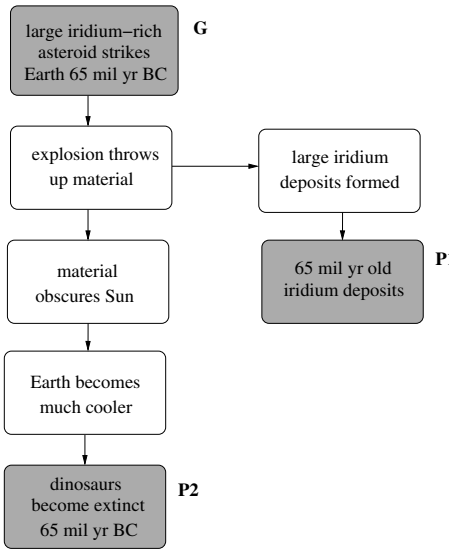


FIGURE 5.11
Asteroid argument graph.

Attention. The first fact about cognition, natural or artificial, is that, unlike the presumptions of some philosophers, no cognitive agent has access to infinite reserves of time or inferential power[‖]. We employ a constraining mechanism that is simple in concept, namely, attention. There is general agreement that attention serves to apply limited cognitive capacities to problem solving by regulating the flow of information to cognitive processes (e.g., Baars's functions of attention [12]). We implement such an attentional process through three features:

- An object's **salience**, the extent to which an object is prominent within the set of objects being processed by an agent at some time
- **Recency**, the time elapsed since the cognitive object was last "touched" by a cognitive process
- **Semantic distance**, the degree of semantic or associative relationship between objects

In order to assess semantic distance, we build semantic networks over the Bayesian networks in our user and normative models, as in Figure 5.12. The **semantic networks** represent the semantic relatedness of items directly, in terms of links between nodes and their association strengths. We take the context in which an argument occurs to provide the initial measure of salience: for example, if the user presents an argument to NAG, the propositions in the argument, and any in the preceding

[‖]See Cherniak's *Minimal Rationality* [44] for an interesting investigation of the difficulties with such philosophical commitments.

© 2004 by Chapman & Hall/CRC Press LLC

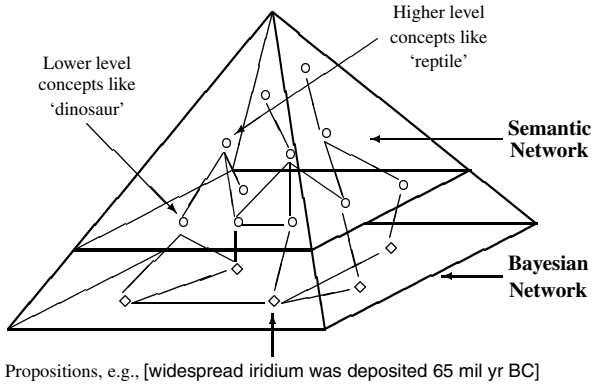


FIGURE 5.12
NAG combines semantic and Bayesian networks.

discussion, will be marked as salient. Again, when assessing the presentation of an argument, each proposition in the argument graph becomes salient in the order in which the presentation is being considered. Finally, we use activation [7], spreading from the salient objects (which are clamped at a fixed level of activation) through both the Bayesian and semantic networks, to determine the focus of attention. All items in the Bayesian networks which achieve a threshold activation level while the spreading activation process is iteratively applied will be brought into the span of attention.

This attentional process allows us to decide that some propositions need not be stated explicitly in an argument: if a proposition achieves sufficient activation, without itself being clamped during presentation, it need not be stated.

The attentional focus determines the limits of Bayesian network propagation. The argument graph identifies the premises of the argument — that is, which nodes are to be treated as observational during propagation — but the propagation itself is constrained by the span of attention, which commonly is larger than the argument graph alone. Such an attention-based model of cognition is inherently incomplete: the import of evidence may not be fully realized when that evidence is acquired and only absorbed over time and further argument, as is only to be expected for a limited cognitive agent. Hence, our probability propagation scheme is partial, and the paradox of inference evaded.

5.5.4 Example: The asteroid strike continues

The distinction between activation level and degree of belief needs to be emphasized. A node may have a high or low activation level with any degree of belief attached to it and vice versa. Clamping, for example, effectively means that the node is acting as an observation node during propagation; but observing means being a source in Bayesian network propagation and not necessarily being fully believed (see §2.3.2).

In our asteroid example, if the user model for the asteroid argument differed from the normative model by having a higher prior probability for the disease explanation

© 2004 by Chapman & Hall/CRC Press LLC

of saurian death (P4), it would still be activated to the same extent in the initial round, since it is part of the initial argument context. The argument graph of Figure 5.11 might well never be produced, however, because with the disease explanation of a key premise available, the goal node in the user model may not reach the target range with that argument. An argument graph corresponding to the complete Bayesian network of Figure 5.10 could nevertheless reach the target range in both Bayesian networks, since it provides additional evidential support for the asteroidal cause.

5.5.5 The future of argumentation

We have briefly described the NAG concept and architecture and given some idea of what it can do. What it *cannot* do at present is worth noting as well. It was designed around Bayesian networks in concept, so its primary means of interaction is via the networks themselves. Nevertheless, it can handle propositions presented to it in ordinary English, but only when the sentences are preprocessed in various ways. The Bayesian networks themselves, user and normative, must be prepared in advance and by hand. Ideally, of course, we would like to have machine learning software which could generate Bayesian networks for NAG, as well as other software which could generate useful networks from data bases and encyclopedia. As we noted earlier, we would also like to be able to validate and modify user models based upon user performance during argumentation. Extending NAG in these different ways is a long-term, difficult goal.

A more likely intermediate goal would be to employ NAG on the task of explaining Bayesian networks (see §10.4.1). It is often, and rightly, remarked that one of the advantages of Bayesian networks over many other AI representations (e.g., neural networks) is that the graphs, both nodes and arcs, have a relatively intuitive semantics. That helps end users to understand what the networks are "saying" and why they are saying it. Our practice suggests that this is quite correct. Despite that, end users still have difficulties understanding the networks and why, say, a target conditional probability has shifted in some particular way. We are planning to utilize the argumentation abilities of NAG in the task of answering such questions as *Why did this network predict that the patient will die under intervention X?* or *Why is intervention A being preferred to intervention B?*

5.6 Summary

In this chapter we have reviewed various BN structures for medical diagnosis applications and surveyed some of the early medical and other applications in the literature. In recent years there have been many new BN and DBN applications, and a full survey is beyond our scope and certainly would be out-of-date quickly. While it is a positive development for BN technology that more applications are being developed commercially, one result is that these are less likely to be published in the research literature or be included in public BN repositories.

© 2004 by Chapman & Hall/CRC Press LLC

We have also described in some detail the development of BN models for three quite different applications. First, we described a game playing application, poker, where the BN was used to estimate the probability of winning and to make betting decisions. Then we saw that a DBN model for ambulation monitoring and fall diagnosis can overcome the limitations of a state-machine approach. Given evidence from sensor observations, the DBN outputs beliefs about the current walking status and makes predictions regarding future falls. The model represents sensor error and is parameterized to allow customization to the individual being monitored. Finally, we looked at how BNs can be used simultaneously for normative domain modeling and user modeling in NAG, by integrating them with semantic networks using an attentional process.

At this stage, it is also worth noting that different evaluation methods were used in each of the example applications, including an experimental/empirical evaluation for the Bayesian poker player, and case-based evaluation for ambulation monitoring DBN. We take up the issue of evaluation in Chapter 10.

5.7 Bibliographic notes

There have been a number of attempts to collect information about successful BN applications and, in particular, systems in regular use, such as that of Russ Greiner (`http://excalibur.brc.uconn.edu/~baynet/fieldedSystems.html`) and Eugene Santos Jr. These and others have been used as a basis for our survey in §5.2.

An interesting case study of PATHFINDER's development is given in [237, p. 457]. The high level of interest in BN applications for medicine is indicated by the workshop *Bayesian Models in Medicine* at the 2001 European Conference on AI in Medicine (AIME'01). The journal *Artificial Intelligence in Medicine* may also be consulted. Husmeier et al. have a forthcoming text on probabilistic models for medical informatics and bioinformatics [119].

Findler [82] was the first to work on automated poker play, using a combination of a probabilistic assessment of hand strength with the collection of frequency data for opponent behavior to support the refinement of the models of opponent. Waterman [295] and Smith [259] used poker as a testbed for automatic learning methods, specifically the acquisition of problem-solving heuristics through experience. Koller and Pfeffer [152] have developed *Gala*, a system for automating game-theoretic analysis for two-player competitive imperfect information games, including simplified poker. Although they use comparatively efficient algorithms, the size of the game tree is too large for this approach to be applied to full poker. Most recently, Billings et al. [19, 20] have investigated the automation of the poker game Texas Hold'em with their program *Poki*. Our work on Bayesian networks for poker has largely appeared in unpublished honors theses (e.g., [132, 38]), but see also [160].

© 2004 by Chapman & Hall/CRC Press LLC

Ambulation monitoring and fall detection is an ongoing project, with variations, extensions and implementations described in [203, 186, 300].

NAG has been described in a sequence of research articles, including the first description of the architecture [305], detailed architectural descriptions (e.g., [306]), a treatment of issues in the presentation of arguments [228] and the representation of human cognitive error in the user model [136].

5.8 Problems

Problems using the example applications

Problem 1

Build the normative asteroid Bayesian network of Figure 5.10 and put in some likely prior probabilities and CPTs. Make a copy of this network, but increase the prior probability for P4; this will play the role of the user model for this problem. Operate both networks by observing P1, P2 and P3, reporting the posterior belief in P4 and G.

This is not the way that NAG operates. Explain how it differs. In your work has "explaining away" played a role? Does it in the way NAG dealt with this example in the text? Explain your answer.

Problem 2

What are the observation nodes when using the Poker network for estimating BPP's probability of winning? What are the observation nodes when using the network to assess the opponent's estimate of winning chances?

Problem 3

Determine a method of estimating the expected future contributions to the pot by both BPP and OPP, to be used in the utilities in the *Winnings* node. Explain your method together with any assumptions used.

Problem 4

All versions of the Bayesian Poker Player to date have used a succession of four distinct Bayesian networks, one per round. In a DDN model for poker, each round in the poker betting would correspond to a single time slice in the DDN, which would allow explicit representation of the interrelation between rounds of play.

1. How would the network structure within a single time slice differ from the DN shown in Figure 5.5?
2. What would the interslice connections be?
3. What information and precedence links should be included?

© 2004 by Chapman & Hall/CRC Press LLC

Problem 5

In §5.4.4, we saw how the use of a sensor status node can be used to model both intermittent and persistent faults in the sensor. In the text this node had only two states, *working* and *defective*, which does not represent the different kinds of faults that may occur, in terms of false negatives, false positives, and so on. Redesign the sensor status modeling to handle this level of detail.

New Applications

The BN, DBN and decision networks surveyed in this chapter have been developed over a period of months or years. It is obviously not feasible to set problems of a similar scope. The following problems are intended to take 6 to 8 hours and require domain knowledge to be obtained either from a domain expert or other sources.

Problem 6

Suppose that a dentist wants to use a Bayesian network to support the diagnosis and treatment of a patient's toothache. Build a BN model including at least 3 possible causes of toothache, any other symptoms of these causes and two diagnostic tests. Then extend the model to include possible treatment options, and a utility model that includes both monetary cost and less tangible features such as patient discomfort and health outcomes.

Problem 7

Design a heart attack Bayesian network which can be used to predict the impact of changes in lifestyle (e.g., smoking, exercise, weight control) on the probability of premature death due to heart attack. You will have to make various choices about the resolution of your variables, for example, whether to model years or decades of life lost due to heart attack. Such choices may be guided by what data you are able to find for parameterizing your network. In any case, do the best you can within the time frame for the problem. Then use your network to perform a few case studies. Estimate the expected number of years lost from heart attack. Then model one or more lifestyle changes and re-estimate.

Problem 8

Build a Bayesian network for diagnosing why a car on the road has broken down. First, identify the most important five or six causes for car breakdown (perhaps, lack of fuel or the battery being flat when attempting to restart). Identify some likely symptoms you can use to distinguish between these different causes. Next you need the numbers. Do the best you can to find appropriate numbers within the time frame of the assignment, whether from magazines, the Internet or a mechanic. Run some different scenarios with your model, reporting the results.

© 2004 by Chapman & Hall/CRC Press LLC

Problem 9

Design a Bayesian network bank loan credit system which takes as input details about the customer (banking history, income, years in job, etc.) and details of a proposed loan to the customer (type of loan, interest rate, payment amount per month, etc.) and estimates the probability of a loan default over the life of a loan. In order to parameterize this network you will probably have to make guestimates based upon aggregate statistics about default rates for different segments of your community, since banks are unlikely to release their proprietary data about such things.

© 2004 by Chapman & Hall/CRC Press LLC

Part II

LEARNING CAUSAL MODELS

© 2004 by Chapman & Hall/CRC Press LLC

These three chapters describe how to apply machine learning to the task of learning causal models (Bayesian networks) from statistical data. This has become a hot topic in the data mining community. Modern databases are often so large they are impossible to make sense of without some kind of automated assistance. **Data mining** aims to render that assistance by discovering patterns in these very large databases and so making them intelligible to human decision makers and planners. Much of the activity in data mining concerns rapid learning of simple association rules which may assist us in predicting a target variable's value from some set of observations. But many of these associations are best understood as deriving from causal relations, hence the interest in automated causal learning, or **causal discovery**.

The machine learning algorithms we will examine here work best when they have large samples to learn from. This is just what large databases are: each row in a relational database of N columns is a joint observation across N variables. The number of rows is the sample size.

In machine learning samples are typically divided into two sets from the beginning: a **training set** and a **test set**. The training set is given to the machine learning algorithm so that it will learn whatever representation is most appropriate for the problem; here that means either learning the causal structure (the dag of a Bayesian network) or learning the parameters for such a structure (e.g., CPTs). Once such a representation has been learned, it can be used to predict the values of target variables. This might be done to test how good a representation it is for the domain. But if we test the model using the very same data employed to learn it in the first place, we will reward models which happen to fit noise in the original data. This is called **overfitting**. Since overfitting almost always leads to inaccurate modeling and prediction when dealing with new cases, it is to be avoided. Hence, the test set is isolated from the learning process and is used strictly for testing *after* learning has completed.

Almost all work in causal discovery has been looking at learning from **observational data** — that is, simultaneous observations of the values of the variables in the network. There has also been work on how to deal with joint observations where some values are missing, which we discuss in Chapter 7. And there has been some work on how to infer **latent structure**, meaning causal structure involving variables that have not been observed (also called **hidden variables**). That topic is beyond the scope of this text (see [200] for a discussion). Another relatively unexplored topic is how to learn from **experimental data**. Experimental data report observations under some set of causal interventions; equivalently, they report joint observations over the augmented models of §3.8, where the additional nodes report causal interventions. The primary focus of Part II of this book will be the presentation of methods that are relatively well understood, namely causal discovery with observational data. That is already a difficult problem involving two parts: searching through the **causal model space** $\{h_i\}$, looking for individual (causal) Bayesian networks h_i to evaluate; evaluating each such h_i relative to the data, perhaps using some score or metric, as in Chapter 8. Both parts are hard. The model space, in particular, is exponential in the number of variables (§6.2.3).

We present these causal discovery methods in the following way. We start in Chap-

© 2004 by Chapman & Hall/CRC Press LLC

ter 6 by describing algorithms for the discovery of linear causal models (also called path models, structural equation models). We begin with them for two reasons. First, this is the historical origin of graphical modeling and we think it is appropriate to pay our respects to those who invented the concepts underlying current algorithms — particularly since they are not very widely known in the AI community. Those who have a background in the social sciences and biology will already be familiar with the techniques. Second, in the linear domain the mathematics required is particularly simple, and so serves as a good introduction to the area. It is in this context that we introduce **constraint-based learning** of causal structure: that is, applying knowledge of conditional independencies to make inferences about what causal relationships are possible. In Chapter 7 we consider the learning of conditional probability distributions from sample data. We start with the simplest case of a binomial root node and build to multinomial nodes with any number of parents. We then discuss learning parameters when the data have missing values, as is normal in real-world problems. In dealing with this problem we introduce **expectation maximization (EM)** techniques and **Gibbs sampling**. In the final chapter of this part, Chapter 8, we introduce the metric learning of causal structure for discrete models, including our own, **CaMML**. Metric learners search through the exponentially complex space of causal structures, attempting to find that structure which optimizes their metric. The metric usually combines two scores, one rewarding fit to the data and the other penalizing overly complex causal structure.

© 2004 by Chapman & Hall/CRC Press LLC

6

Learning Linear Causal Models

6.1 Introduction

Thus far, we have seen that Bayesian networks are a powerful method for representing and reasoning with uncertainty, one which demonstrably supports normatively correct Bayesian reasoning and which is sufficiently flexible to support user modeling equally well when users are less than normative. BNs have been applied to a very large variety of problems; most of these applications have been academic exercises — that is to say, prototypes intended to demonstrate the potential of the technology, rather than applications that real businesses or government agencies would be relying upon. The main reason why BNs have not yet been deployed in many significant industrial-strength applications is the same reason earlier rule-based expert systems were not widely successful: the **knowledge bottleneck**.

Whether expert systems encode domain knowledge in rules or in conditional probability relations, that domain knowledge must come from somewhere. The main plausible source until recently has been human domain experts. When building knowledge representations from human experts, AI practitioners (called knowledge engineers in this role) must elicit the knowledge from the human experts, interviewing or testing them so as to discover compact representations of their understanding. This encounters a number of difficulties, which collectively make up the "knowledge bottleneck" (cf. [81]). For example, in many cases there simply are no human experts to interview; many tasks to which we would like to put robots and computers concern domains in which humans have had no opportunity to develop expertise — most obviously in exploration tasks, such as exploring volcanoes or exploring sea bottoms. In other cases it is difficult to articulate the humans' expertise; for example, every serious computer chess project has had human advisors, but human chess expertise is notoriously inarticulable, so no good chess program relies upon rules derived from human experts as its primary means of play — they all rely upon brute-force search instead. In all substantial applications the elicitation of knowledge from human experts is time consuming, error prone and expensive. It may nevertheless be an important means of developing some applications, but if it is the *only* means, then the spread of the underlying technology through any large range of serious applications will be bound to be slow and arduous.

The obvious alternative to relying entirely upon knowledge elicitation is to employ machine learning algorithms to automate the process of constructing knowledge rep-

© 2004 by Chapman & Hall/CRC Press LLC

resentations for different domains. With the demonstrated potential of Bayesian networks, interest in methods for automating their construction has grown enormously in recent years. In this chapter we look again at the relation between conditional independence and causality, as this provides a key to causal learning from observational data.

This key relates to Hans Reichenbach's work on causality in *The Direction of Time* (1956) [233]. Recall from §2.4.4 the types of causal structures available for three variables that are in an undirected chain; these are causal chains, common causes and common effects, as in Figure 6.1. Reichenbach proposed the following principle:

Conjecture 6.1 Principle of the Common Cause *If two variables are probabilistically dependent, then either one causes the other (directly or indirectly) or they have a common ancestor.*

Reichenbach also attempted to analyze time and causality in terms of the different dependency structures exemplified in Figure 6.1; in particular, he attempted to account for time asymmetry by reference to the dependency asymmetry between common causal structures and common effect structures. In this way he anticipated d-separation, since that concept depends directly upon the dependency asymmetries Reichenbach studied. But it is the Principle of the Common Cause which underlies causal discovery in general. That principle, in essence, simply asserts that behind every probabilistic dependency is an explanatory causal dependency. And that is something which all of science assumes. Indeed, we should likely abandon the search for an explanatory cause for a dependency only after an exhaustive and exhausting search for such a cause had failed — or, perhaps, if there is an impossibility proof for the existence of such a cause (as is arguably the case for the entangled systems of quantum mechanics). The causal discovery algorithms of this chapter explicitly search for causal structures to explain probabilistic dependencies and, so, implicitly pay homage to Reichenbach.

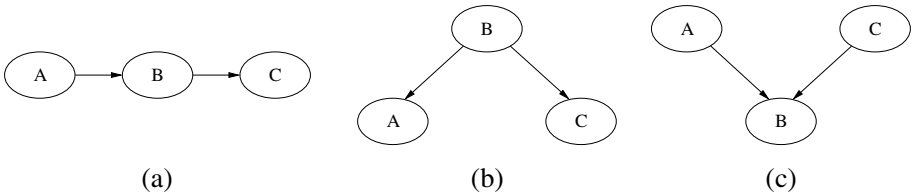


FIGURE 6.1
(a) Causal chain; (b) common cause; (c) common effect.

© 2004 by Chapman & Hall/CRC Press LLC

6.2 Path models

The use of graphical models to represent and reason about uncertain causal relationships began with the work of the early twentieth-century biologist and statistician Sewall Wright [302, 303]. Wright developed graphs for portraying linear causal relationships and parameterized them based upon sample correlations. His approach came to be called **path modeling** and has been extensively employed in the social sciences. Related techniques include structural equation modeling, widely employed in econometrics, and causal analysis. Behind the widespread adoption of these methods is, in part, just the restriction to linearity, since linearity allows for simpler mathematical and statistical analysis. Some of the restrictiveness of this assumption can be relaxed by considering transformations of non-linear to linear functions. Nevertheless, it is certain that many non-linear causal relations have been simplistically understood in linear terms, merely because of the nature of the tools available for analyzing them.

In using Bayesian networks to represent causal models we impose no such restrictions on our subject. Nevertheless, we shall first examine Wright's "method of path coefficients." Path modeling is preferable to other linear methods precisely because it directly employs a graphical approach which relates closely to that of Bayesian networks and which illustrates in a simpler context many of the causal features of Bayesian networks.

A linear model relates the effect variable Y to parent variables X_i via an additive, linear function, as in:

$$Y = a_1 X_1 + a_2 X_2 + U \tag{6.1}$$

In this case a_1 is a constant coefficient that reflects the extent to which parent variable X_1 influences, accounts for or explains the value of Y; similarly for a_2 and X_2. U represents the mean value of Y and the variation in its values which cannot be explained by the model. If Y is in fact a linear function of all of its parent variables, then U represents the cumulative linear effect of all the parents that are unknown or unrepresented in this simple model. If U is distributed as a Gaussian $N(\mu, \sigma^2)$ (normal), the model is known as **linear Gaussian**. This simple model is equally well represented by Figure 6.2. Usually, we will not bother to represent the factor U explicitly, especially in our graphical models. When dealing with standardized linear models, the impact of U can always be computed from the remainder of the model in any case.

A linear model may come from any source, for example, from someone's imagination. It may, of course, come from the statistical method of linear regression, which finds the linear function which minimizes unpredicted (residual) variation in the value of the dependent variable Y given values for the independent variables X_i. What we are concerned with in this chapter is the discovery of a joint system of linear equations by means distinct from linear regression. These means may not necessarily strictly minimize residual variation, but they will have other virtues, such

© 2004 by Chapman & Hall/CRC Press LLC

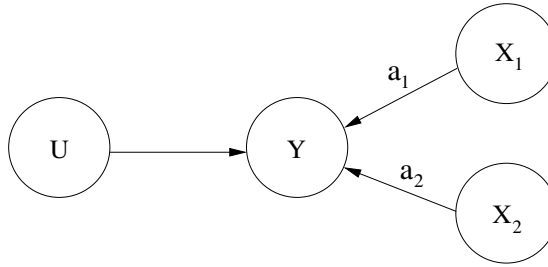


FIGURE 6.2
A linear model.

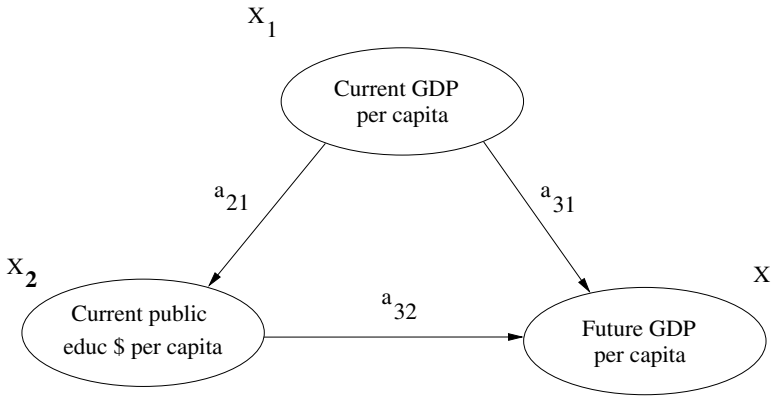


FIGURE 6.3
OECD public education spending model.

as discovering better explanatory models than regression methods can, as we shall see.

Let's consider a concrete example of a path model*. Figure 6.3 reports the relations between public educational spending in OECD countries† and per capita economic activity at the time of the spending and, again, 15 years later (measured in "GDP per capita," which means gross domestic product per person in the economy; this example is taken, with minor modifications, from [161]). This model indicates that a typical OECD nation will receive some amount of future economic benefit

*More precisely, we consider here a **recursive path model**. A path model is called recursive just in case it is a directed acyclic graph, as are all those we will consider here. It is possible also to deal with non-recursive path models, which replace some subnetwork relating two variables with a bidirectional arc parameterized with a correlation. Such models are essentially incomplete recursive models: if the subnetwork were known, it would be more informative than the simple correlation and so would be used instead.

†OECD stands for *Organization for Economic Co-operation and Development* and includes all of the major developed countries.

© 2004 by Chapman & Hall/CRC Press LLC

from an extra $1 added to the current economy in general (for example, via a tax cut) and some different future economic benefit from the dollar being added to public education instead. The graphical model can be transformed into a system of linear equations by writing down one equation per dependent variable in the model.

$$X_2 = a_{21}X_1$$
$$X_3 = a_{32}X_2 + a_{31}X_1$$

Focusing on X_3 (Future GDP per capita), there are three kinds of linear causal influence reported by the model: direct, $X_1 \rightarrow X_3$ (as well as $X_2 \rightarrow X_3$ and $X_1 \rightarrow X_2$); indirect, $X_1 \rightarrow X_2 \rightarrow X_3$; and the "spurious" effect of X_1 inducing a correlation between X_2 and X_3. Assuming the Markov property, there can be no other causal influences between variables in the model. This has the further implication, applying Reichenbach's Principle of the Common Cause, that there can also be no other sources of correlation between the variables in the model. This suggests that correlation between the variables and the linear causal influences should be interrelatable: if correlations can only exist where causal influences induce them, and if all causal influences are accounted for in a model (as required by the Markov property), then we should be able to transform correlations into causal weights and vice versa. And so we can.

The interrelation of correlation and linear causal influence is a key result. The causal influences are theoretical: agreeing with David Hume [118], we cannot simply observe causal forces as such. However, we can observe and measure correlations. Since the two can be precisely interrelated, as we will see immediately below, we can use the observations of correlation to discover, and specify the strength of, linear causal relationships.

6.2.1 Wright's first decomposition rule

Sewall Wright's first Decomposition Rule gives the exact relation between correlations and linear causal coefficients [303]. His rule assumes that all variables have been standardized; that is, the scale for each variable has been transformed into standard deviation units. This is easily done for any variable. For example, Future GDP per capita (X_3) values in dollars can be replaced by deviations from the OECD average (Z_3) via the transformation:

$$Z_3 = \frac{X_3 - \mu_3}{\sigma_3} \tag{6.2}$$

where μ_3 is the mean Future GDP per capita and σ_3 is its standard deviation. Standardized variables all have means of zero and standard deviations of 1. The path coefficient p_{ij} is the analog in the standardized model of the linear coefficient in the non-standardized model (the two are related below in Equation 6.5).

© 2004 by Chapman & Hall/CRC Press LLC

Rule 6.1 Wright's Decomposition Rule. *The correlation r_{ij} between variables X_i and X_j, where X_i is an ancestor of X_j, can be rewritten according to the equation:*

$$r_{ij} = \sum_k p_{jk} r_{ki} \tag{6.3}$$

where p_{jk} are path coefficients relating X_j with each of its direct parents X_k.

Application of this rule replaces a correlation, r_{ij}, with products of a path coefficient to a parent X_k and another correlation relating the ancestor X_i to this parent; since this last correlation must relate two nodes X_k and X_i more closely connected than the original pair X_j and X_i, it is clear that repeated application of the rule will eventually eliminate reference to correlations. In other words, using Wright's rule we can rewrite any correlation in terms of sums of products of path coefficients. This will give us a system of equations which we can then use to solve for the path coefficients.

For example, taking each variable in turn from X_1 to X_2 to X_3 in Figure 6.3 we can use Rule 6.1 to generate the following equations for modeling the dependent variables (bearing in mind that $r_{11} = r_2 2 = 1$):

$$\begin{aligned}
\text{Take } X_1 : & \quad \text{nothing} \\
\text{Take } X_2 : r_{12} &= \sum_k p_{2k} r_{k1} \\
&= p_{21} r_{11} \\
&= p_{21} \\
\text{Take } X_3 : r_{13} &= \sum_k p_{3k} r_{k1} \\
&= p_{31} r_{11} + p_{32} r_{12} \\
&= p_{31} + p_{32} r_{12} \\
r_{23} &= \sum_k p_{3k} r_{k2} \\
&= p_{31} r_{12} + p_{32} r_{22} \\
&= p_{31} r_{12} + p_{32}
\end{aligned}$$

We have three equations in three unknowns, so we can solve for the path coefficients:

$$\begin{aligned}
p_{21} &= r_{12} \\
p_{31} &= \frac{r_{13} - r_{23} r_{12}}{1 - r_{12}^2} \\
p_{32} &= \frac{r_{23} - r_{13} r_{12}}{1 - r_{12}^2}
\end{aligned}$$

In the public education example, the observed correlations were $r_{12} = 0.8212$, $r_{13} = 0.5816$, and $r_{23} = 0.6697$. Plugging these into the above equations gives us

© 2004 by Chapman & Hall/CRC Press LLC

the path coefficients

$$\begin{aligned} p_{21} &= 0.8212 \\ p_{32} &= 0.5899 \\ p_{31} &= 0.0972 \end{aligned}$$

These are standardized coefficients, reporting the impact of causal interventions on any parent variables upon their children.

There is an equivalent rule for decomposing correlations into path coefficients which is even simpler to apply, and which also relates matters back to Reichenbach's discussion of causality and the concept of d-separation.

Rule 6.2 Wright's Second Decomposition Rule. *The correlation r_{ij} between variables X_i and X_j, where X_i is an ancestor of X_j, can be rewritten according to the equation:*

$$r_{ij} = \sum_k v(\Phi_k) \tag{6.4}$$

where Φ_k is an active path between X_i and X_j and $v(\cdot)$ is a valuation of that path.

Intuitively, each active path represents a distinct line of causal influence, while its valuation measures the degree of causal influence. Note that these paths are not simply undirected paths, instead:

Definition 6.1 Active Path. *Φ_k is an active path between X_i and X_j if and only if it is an undirected path (see Definition 2.1) connecting X_i and X_j such that it does not go against the direction of an arc* **after** *having gone forward.*

The valuation of a path is

$$v(\Phi_k) = \begin{cases} p_{ij} \text{ if } \Phi_k = X_j \rightarrow X_i \\ \prod_{lm} p_{lm} \text{ for all } X_m \rightarrow X_l \in \Phi_k \end{cases}$$

This decomposition of correlation into causal influences corresponds directly to Reichenbach's treatment of causal asymmetry, and therefore also prefigures d-separation. A path traveling always forward from cause to (ultimate) effect identifies a causal chain, of course; a path traveling first backwards and then forwards (but never again backwards) identifies a common ancestry between each pair of variables along the two branches; and the prohibition of first traveling forwards and then backwards accounts for the conditional dependencies induced by common effects or their successors.

This version of Wright's rule also has the property of d-separation that has made that concept so successful: the causal influences represented by a path model can easily be read off of the graph representing that model. Applied to Figure 6.3, for

© 2004 by Chapman & Hall/CRC Press LLC

example, we readily find:

$$\begin{aligned} r_{12} &= \sum_k v(\Phi_k) \\ &= v(X_1 \rightarrow X_2) \\ &= p_{21} \end{aligned}$$

$$\begin{aligned} r_{13} &= v(X_1 \rightarrow X_3) + v(X_1 \rightarrow X_2 \rightarrow X_3) \\ &= p_{31} + p_{21}p_{32} \\ r_{23} &= v(X_2 \rightarrow X_3) + v(X_2 \leftarrow X_1 \rightarrow X_3) \\ &= p_{32} + p_{21}p_{31} \end{aligned}$$

which is identical, of course, with what we obtained with Rule 6.1 before.

The path coefficients of Wright's models are directly related to the linear coefficients of regression models: just as the variables are arrived at by standardizing, the path coefficient p_{ij} may be arrived at by standardizing the regression coefficient a_{ij}:

$$p_{ij} = a_{ij}\left(\frac{\sigma_j}{\sigma_i}\right) \tag{6.5}$$

As a consequence of the standardization process, the sum of squared path coefficients convergent on any variable X_i is constrained to equal one:

$$\sum_j p_{ij}^2 = 1$$

This is true assuming that one of the parent variables represents the variance unexplained by the regression model over the known parents — i.e., the U variable is included. Since these sum to one, the square of the path coefficient, p_{ij}^2, can be understood as the proportion of the variation in the child variable X_i attributable to parent X_j, and if p_{iu} is the coefficient associated with the residual variable U, then p_{iu}^2 is the amount of variance in X_i which is left unexplained by the linear model.

Returning again to the public education model of Figure 6.3, the path coefficients were

$$\begin{aligned} p_{21} &= 0.8212 \\ p_{32} &= 0.5899 \\ p_{31} &= 0.0972 \end{aligned}$$

Since these are standardized coefficients, they report the square root of the amount of variation in the child variable that is explained by the parent. In other words, for example, variations in public GDP (X_1) account for the fraction $(0.8212)^2 = 0.674 = 67.4\%$ of the variation in public spending on education across OECD countries (X_2).

© 2004 by Chapman & Hall/CRC Press LLC

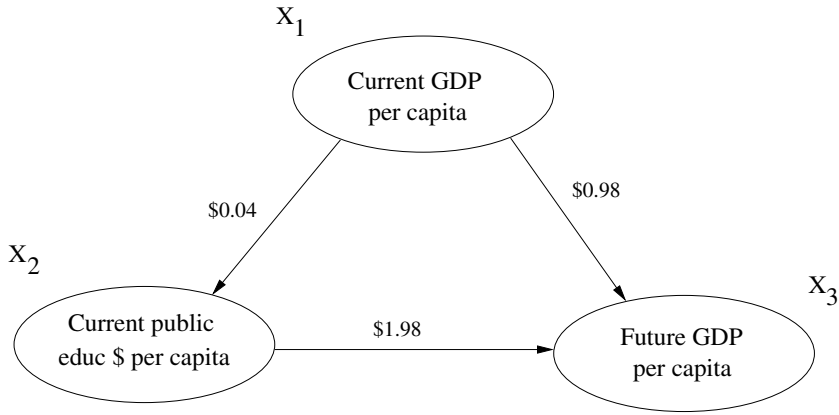


FIGURE 6.4
Non-standardized OECD public education spending model.

Standardization makes it very easy to understand the relation between correlations and coefficients, but it isn't everything. By reversing the process of standardization we may well find a more human-readable model. Doing so for public education results in Figure 6.4. From this model we can read off numbers that make more intuitive sense to us than the path coefficients. In particular, it asserts that for a normal OECD country in the period studied, an additional $1 added to the public education budget will expand the future economy by about $2, whereas an additional $1 added at random to the economy (via a tax cut, for example) will expand the future economy by about $1.

6.2.2 Parameterizing linear models

All of this work allows us to parameterize our path models in a straightforward way: apply Wright's Decomposition Rule (either one) to obtain a system of equations relating correlation coefficients with the path model's coefficients; solve the system of equations for the path coefficients; compute the solution given sample correlations. This is exactly what was done with the OECD public education model, in fact. It has been proven that recursive path models will always have a system of equations which are solvable, so the method is always available [28]. There are equivalent alternative methods, such as one due to Simon and Blalock [257, 21] and ordinary least squares regression. As we mentioned above, however, we prefer Wright's method because it is more intuitively connected to causal graph representations.

6.2.3 Learning linear models is complex

The ability to parameterize linear models in turn ushers in a very plausible methodology of learning linear models, which indeed is quite widely employed in the social sciences:

1. Find some correlations that are puzzling.

© 2004 by Chapman & Hall/CRC Press LLC

2. Invent a possible causal model that might, if true, explain those correlations (a step which is variously called **abduction** or **inference to the best explanation**).
3. Estimate ("identify") the model parameters, whether via Wright's Rule, least squares or the Simon-Blalock method, in order to complete the model.
4. Test the complete model by predicting new data and comparing reality with the prediction. If reality diverges from the model's expectation, GOTO step 2.

Numerous problems in the social sciences have been investigated in this way, and arguably many of them have been solved. For example, notwithstanding the obfuscation of some politicians, there is no serious doubt amongst economists who investigate such matters that investment in public education is a key ingredient in future economic well-being, partly because of such studies as ours. In other words, we know for a fact that it is possible to learn causal structure from correlation structure, because we humans can and do. Implementing such a process in a computer is another matter, to be sure: the above procedure, in particular, does not immediately commend itself, since no one knows how we perform step 2 — inventing explanatory models.

In AI the standard response to the need to invent or create is to employ search. If we have a clear criterion for evaluating an artifact, but no obvious preferred means for constructing it, then the standard AI move is to search the space of possible artifacts, stopping when we have found one that is satisfactory. Such a search might start with the simplest possible artifact — perhaps a null artifact — and, by testing out all possible single additions at each step, perform a breadth-first search for a satisfactory artifact. Such a search is perfectly fine when the size of some satisfactory artifact is small: since small artifacts are examined first, an acceptable one will be found quickly. But the time complexity of such a search is of the order b^d, where b is the **branching factor** (the number of possible single additions to an artifact) and d is the **solution depth** (the size of the simplest acceptable artifact). So, if we are looking for a sizable artifact, or working with a very large number of possible additions to an artifact at any step, then such a simple search will not work. The exponential time complexity of such brute-force search is what has made world-class computer chess difficult to achieve and world-class computer go (thus far) unattainable. AI as a field has investigated many other types of search, but given a highly complex search space, and no very helpful guide in how to search the space, they will all succumb to exponential search complexity — that is, they will not finish in reasonable time. We will now show that part of these conditions for unsolvability exist in the learning of causal models: the space of causal models is exponentially complex.

How many possible causal models (i.e., dags) are there? The number depends upon the number of variables you are prepared to entertain. A recursive expression for the number of dags $f(N)$ given N variables is [236]:

$$f(N) = \sum_{i=1}^{N} (-1)^{i+1} C_i^N 2^{i(N-i)} f(N-1) \tag{6.6}$$

© 2004 by Chapman & Hall/CRC Press LLC

with $f(0) = f(1) = 1$. With two variables, there are three dags; with three variables there are 25 dags; with five variables there are 25,000 dags; and with ten variables there are about 4.2×10^{18} possible models. As can be seen in (6.6), this grows exponentially in N.

This problem of exponential growth of the model space does not go away nicely, and we will return to it in Chapter 8 when we address metric learners of causal structure. In the meantime, we will introduce a heuristic search method for learning causal structure which is effective and useful.

6.3 Conditional independence learners

We can imagine a variety of different heuristic devices that might be brought to bear upon the search problem, and in particular that might be used to reduce the size of the space. Thus, if we had partial prior knowledge of some of the causal relations between variables, or prior knowledge of temporal relations between variables, that could rule out a great many possible models. We will consider the introduction of specific prior information later, in the section on adaptation (§9.4).

But there must also be methods of learning causal structure which do not depend on any special background knowledge: humans (and other animals), after all, learn about causality from an early age, and in the first instance without much background. Evolution may have built some understanding into us from the start, but it is also clear that our individual learning ability is highly flexible, allowing us severally and communally to adapt ourselves to a very wide range of environments. We should like to endow our machine learning systems with such abilities, for we should like our systems to be capable of supporting autonomous agency, as we argued in Chapter 1.

One approach to learning causal structure directly is to employ experimentation in addition to observation: whereas observing a joint correlation between A and B guarantees that there is *some* causal relation between them (via the Common Cause Principle), a large variety of causal relations will suffice. If, however, we intervene — changing the state of A — and subsequently see a correlated change in the state of B, then we can rule out both B being a cause of A and some common cause being the sole explanation. So, experimental learning is clearly a more powerful instrument for learning causal structure.

Our augmented model for causal reasoning of Chapter 3 suggests that learning from experimental data is a special variety of learning from observational data; it is, namely, learning from observational samples taken over the augmented model. Note that adding the intervention variable I_A to the common causal structure Figure 6.1 (b) yields $I_A \rightarrow A \leftarrow B \rightarrow C$. Observations of I_A without observations of A can be interpreted as causal manipulations of A. And, clearly, in such cases experimental data interpreted as an observation in the augmented model will find no dependency between the causal intervention and C, since the intervening v-structure blocks the

© 2004 by Chapman & Hall/CRC Press LLC

path.

Thus, the restriction to observational learning is apparently not a restriction at all. In any case, there is a surprising amount of useful work that can be done with observational data without augmented models. In particular, there is a powerful heuristic search method based upon the conditional independencies implied by a dag. This was introduced by Verma and Pearl [285], in their CI algorithm, and first used in a practical algorithm, the PC algorithm, in TETRAD II [264], presented in §6.3.2.

ALGORITHM 6.1

CI Algorithm

1. **Principle I** *This principle recovers all the direct causal dependencies using undirected arcs. Let* $X, Y \in \mathbf{V}$. *Then* X *and* Y *are directly causally connected if and only if for every* $\mathbf{S} \subset \mathbf{V}$ *such that* $X, Y \notin \mathbf{S}$ $\neg(X \perp\!\!\!\perp Y | \mathbf{S})$.
2. **Principle II** *This principle recovers some of the arc directions, namely those which are discoverable from the dependencies induced by common effect variables.*

   If $X - Y$ *and* $Y - Z$ *but not* $X - Z$ *(i.e., we have an undirected chain* $X - Y - Z$*), then replace the chain by* $X \rightarrow Y \leftarrow Z$ *if and only if for every* $\mathbf{S} \subset \mathbf{V}$ *such that* $X, Z \notin \mathbf{S}$ *and* $Y \in \mathbf{S}$ $\neg(X \perp\!\!\!\perp Z | Y)$. *The structure* $X \rightarrow Y \leftarrow Z$ *is called a v-structure (or: a collider).*
3. *Iterate through all undirected arcs* $Y - Z$ *in the graph. Orient* $Y \rightarrow Z$ *if and only if either*
   (a) *at a previous step* Y *appeared as the middle node in an undirected chain with* X *and* Z *(so, Principle II failed to indicate* Y *should be in v-structure between* X *and* Z*) and now the arc between* X *and* Y *is directed as* $X \rightarrow Y$*;*
   (b) *if we were to direct* $Y \leftarrow Z$*, then a cycle would be introduced.*

   Continue iterating through all the undirected arcs until one such pass fails to direct any arcs.

We can illustrate this algorithm with the simple example of Figure 6.5, a causal model intended to reflect influences on the college plans of high school students [247]. The variables are: *Sex* (male or female), *IQ* (intelligence quotient), *CP* (college plans), *PE* (parental encouragement), *SES* (socioeconomic status).

We can suppose that we have a causal system in these variables which is governed by this causal model and that we have sampled the variables jointly some reasonably large number of times ([247] had a sample of 10,318 students) and that we wish to learn the causal model which generated the sample. In order to operate the CI algorithm, let us imagine that we have an oracle who has access to the Truth — who can peek at the true causal model of Figure 6.5 — and who is forthcoming enough to answer any questions we pose about conditional dependencies and independencies

© 2004 by Chapman & Hall/CRC Press LLC

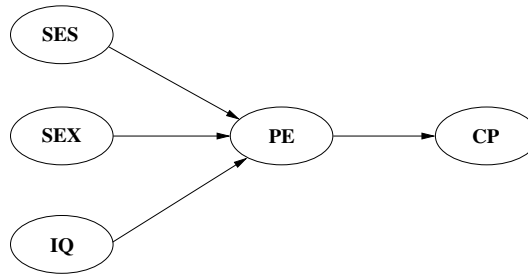


FIGURE 6.5
Causal model for college plans.

— but no more than that. In that case, the CI algorithm allows us to learn the entire structure of Figure 6.5.

First, Principle I of the CI algorithm will tell us all of the pairs of variables which are directly connected by some causal arc, without telling us the causal direction. That is, Principle I gives us Figure 6.6. Principle II of the algorithm might then examine the chain SES–PE–SEX, leading to SES $\rightarrow$ PE $\leftarrow$ SEX. Subsequently, Principle II will examine IQ in combination with PE and either one of SES or SEX, leading to IQ $\rightarrow$ PE. Principle II will lead to no orientation for the link PE–CP, since this step can only introduce v-structures. Step 3(a), however, will subsequently orient the arc PE $\rightarrow$ CP, recovering Figure 6.5 exactly.

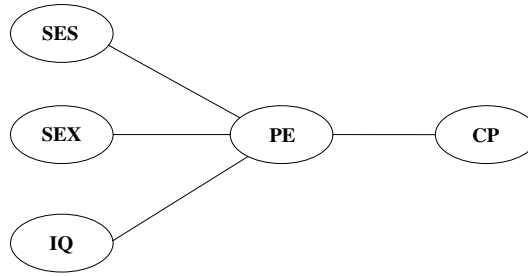


FIGURE 6.6
Causal model for college plans learned after CI Principle I.

The CI algorithm will not always recover the true model. If the true model were the alternative model Figure 6.7, for example, no arc directions would be discoverable: the only two chains available to Principle II are SES–PE–CP and SES–IQ–CP. So, CI will discover the skeleton of Figure 6.7 only.

© 2004 by Chapman & Hall/CRC Press LLC

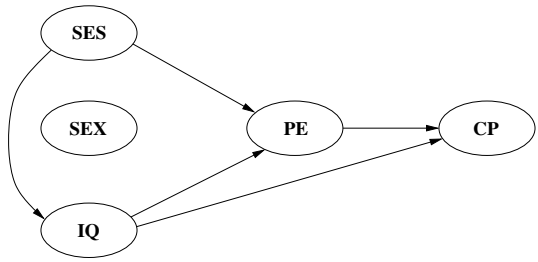


FIGURE 6.7
Alternative causal model for college plans.

6.3.1 Markov equivalence

What Verma and Pearl's CI algorithm is discovering in these cases is the set of Markov equivalent causal models (these are also known as **statistically equivalent** models). Two causal models H_1 and H_2 are **Markov equivalent** if and only if they contain the same variables and any probability distribution that can be represented by one can be represented by the other. In other words, given some parameterization for the one model, we can find a parameterization for the other which yields the very same probability distribution.

That the CI algorithm has this much power and no more follows from Verma and Pearl's proof of the following theorem [285]. (Note that the theorems of this section are known to hold only for linear Gaussian and discrete probability distributions. In general, we will not consider other probability distributions, and so can use them.)

Theorem 6.1 *(Verma and Pearl, 1990) Any two causal models over the same variables that have the same skeleton and the same v-structures are Markov equivalent.*

Given an infallible oracle (and, of course, the Markov property), Principle I will recover the skeleton. Principle II will recover all the v-structures. Step 3 merely enforces consistency with dag structure. Theorem 6.1, therefore, ensures that the partially oriented graph which CI discovers, called a **pattern** by Verma and Pearl, can only be completed by directing unoriented arcs in Markov equivalent ways. In addition to showing us this, the theorem also provides a simple, easily applied graphical criterion of Markov equivalence. Thus, some examples of Markov equivalence:

- All fully connected models (in the same variables) are Markov equivalent. Fully connected models contain no v-structures, since the end variables in any candidate chain must themselves be directly connected. Therefore, on the graphical criterion, they fall under the same pattern.
- $A \rightarrow B \rightarrow C$ and $A \leftarrow B \leftarrow C$ are Markov equivalent. There is no v-structure here.
- $A \rightarrow B \rightarrow D \leftarrow C$ and $A \leftarrow B \rightarrow D \leftarrow C$ are Markov equivalent. The only v-structure, centered on D, is retained.

© 2004 by Chapman & Hall/CRC Press LLC

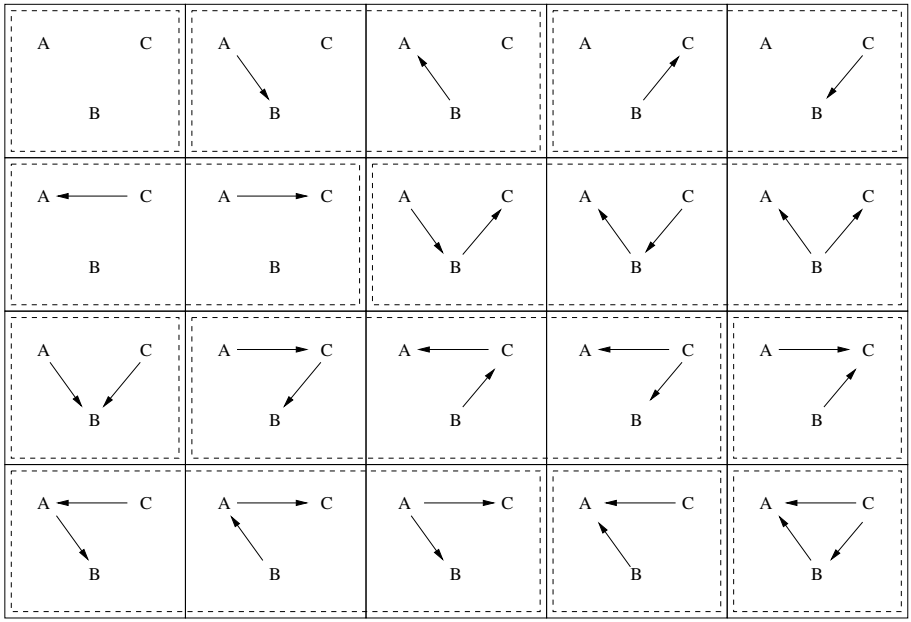


FIGURE 6.8
All patterns in three variables. Patterns are indicated with dotted lines. (Note that the fully connected last pattern shows only one of 6 dags.)

For illustration, all patterns in three variables are displayed in Figure 6.8.

Following on Verma and Pearl's ground-breaking work, D. Max Chickering investigated Markov equivalent models and published some important results in the 1995 *Uncertainty in AI* conference [45].

Theorem 6.2 *(Chickering, 1995) If H_1 and H_2 are Markov equivalent, then they have the same maximum likelihoods relative to any joint samples:*

$$\max P(e|H_1, \theta_1) = \max P(e|H_2, \theta_2)$$

where θ_i is a parameterization of H_i

Since standard statistical inference procedures are largely based upon maximum likelihood measures, it is unlikely that they will have the ability to distinguish between Markov equivalent hypotheses. This may sound like a truism — that statistically indistinguishable (equivalent) hypotheses shouldn't be distinguishable statistically. And, indeed, many researchers, including some calling themselves Bayesians, advocate metrics and methods which assume that causal discovery should be aimed at patterns (Markov equivalence classes) rather than at their constituent causal hypotheses. There are at least two reasons to disagree. First, Bayesian methods properly incorporate prior probabilities and do not merely respond to maximum likelihoods. Second,

© 2004 by Chapman & Hall/CRC Press LLC

what is statistically indistinguishable using observational data alone is not so using experimental data, which are far more discerning in revealing causal structure. In consequence of this received view, far less attention has been paid to experimental inference and to the causal semantics of Bayesian networks than is warranted. We shall have somewhat more to say about this in Chapter 8.

6.3.1.1 Arc reversal

A second result from Chickering's work is very helpful in thinking graphically about causal models.

Theorem 6.3 *(Chickering, 1995) Any two Markov equivalent dags H_i and H_j are connected in a chain of dags $< H_i, \ldots, H_j >$ where any two adjacent dags differ at most in one covered arc reversal.*

This is related to the arc reversal rule:

Rule 6.3 Arc Reversal. *H_2 can represent any probability distribution represented by H_1[‡] if they contain the same variables and arcs except that:*

(a) for some pair of nodes A and B, $A \rightarrow B \in H_1$ and $A \leftarrow B \in H_2$, and

(b) if this arc is in an uncovered v-structure $A \rightarrow B \leftarrow C$ in H_1, then it is covered in H_2, i.e., $C \rightarrow A \in H_2$, and

(c) if the reversal would introduce *an uncovered v-structure $C \rightarrow A \leftarrow B$ into H_2, then either $C \rightarrow B$ or $C \leftarrow B$ must be added to H_2.*

The arc reversal rule allows us to change the temporal ordering between any two variables. This will either keep us within a Markov equivalence class, because no v-structures are thereby introduced or destroyed, or else send us into a new equivalence class by requiring us to introduce a covering arc. The covering arc is necessary either to reestablish conditional independence between A and C given B (clause (b)) or else to reestablish conditional dependence (clause (c)). In either case, the arc reversal forces us to move from a simpler model H_1 to a denser model H_2, and, hence, to move from one equivalence class to another.

Many have thought that the ability to reverse arcs arbitrarily while continuing to represent the same probability distributions demonstrates that Bayesian networks are intrinsically *not* about causality at all, that all that matters for understanding them is that they can represent some class of probability distributions. However, since the arc reversal rule can only introduce and never eliminate arcs, it clearly suggests that among all the models which can represent an observed probability distribution, that model which is the sparsest (has the fewest arcs) is the true causal model. Since the true model by definition has no spurious arcs, it will be the sparsest representation of the observed probability distribution[§]; any alternative arrived at by reversing the

[‡] This rule holds if we restrict the probability distributions to linear Gaussian and multinomial distributions.

[§] To be sure, there are cases of measure zero where a non-causal linear model can represent a linear causal distribution more compactly.

© 2004 by Chapman & Hall/CRC Press LLC

temporal ordering of nodes (equivalently, reversing the arc directions relating those nodes) must be at least as dense as the true model. In any case, a little reflection will reveal that acceptance of Reichenbach's Principle of the Common Cause implies that non-causal distributions are strictly derivative from and explained by some causal model. Of course, it is possible to reject Reichenbach's principle (as, for example, Williamson does [299]), but it is not clear that we can reject the principle while still making sense of scientific method.

As we shall see below, many causal discovery algorithms evade, or presuppose some solution to, the problem of identifying the correct variable order. In principle, these algorithms can be made complete by iterating through all possible orderings to find the sparsest model that the algorithm can identify for each given ordering. Unfortunately, all possible orderings for N variables number $N!$, so this completion is exponential. Possibly, the problem can be resolved by sampling orderings and imposing constraints when subsets of nodes have been ordered. So far as we know, such an approach to causal discovery has yet to be attempted.

6.3.1.2 Markov equivalence summary

In summary, Markov equivalence classes of causal models, or patterns, are related to each other graphically by Verma and Pearl's Theorem 6.1: they share skeletons and v-structures. They are related statistically by having identical maximum likelihoods, and so, by orthodox statistical criteria, they are not distinguishable. Despite that limitation, learning the patterns from observational data is an important, and large, first step in causal learning. We do not yet know, however, how close we can get in practice towards that goal, since the CI algorithm is itself a highly idealized one. So: in reality, how good can we get at learning causal patterns?

6.3.2 PC algorithm

Verma and Pearl's CI algorithm appears to depend upon a number of unrealistic features. First, it depends upon knowledge of the actual conditional independencies between variables. How is such knowledge to be gained? Of course, if one has access to the true causal structure through an actual oracle, then the independencies and dependencies can be read off that structure using the d-separation criterion. But lacking such an oracle, one must somehow infer conditional independencies from observational data. The second difficulty with the algorithm is that it depends upon examining independencies between all pairs of variables given every subset of variables not containing the pair in question. But the number of such alternative subsets is exponential in the number of variables in the problem, making any direct implementation of this algorithm unworkable for large problems.

The causal discovery program TETRAD II copes with the first problem by applying a statistical significance test for conditional independence. For linear models, conditional independence $X \perp\!\!\!\perp Y | \mathbf{S}$ is represented by the zero partial correlation $\rho_{XY \cdot \mathbf{S}} = 0$ (also described as a **vanishing partial correlation**), that is, the correlation remaining between X and Y when the set $\mathbf{S}$ is held constant. The standard

© 2004 by Chapman & Hall/CRC Press LLC

significance test on sample correlations is used to decide whether or not the partial correlation is equal to zero. In what Spirtes et al. [264] call the **PC algorithm**, they combine this significance testing with a small trick to reduce the complexity of the search. The PC algorithm, because it is easy to understand and implement, has recently been taken up by two of the leading Bayesian network software tools, Hugin and Netica (see Appendix B).

ALGORITHM 6.2
PC Algorithm

1. *Begin with the fully connected skeleton model; i.e., every node is adjacent to every other node.*
2. *Set* $k \leftarrow 0$*; k identifies the order of the set of variables to be held fixed. For all pairs of nodes* X *and* Y *set* $\mathbf{DSep}(\mathbf{X}, \mathbf{Y}) = \emptyset$*; this will keep track of nodes which ought to d-separate the pair in the final graph.*
3. *For every adjacent pair of nodes* X *and* Y*, remove the arc between them if and only if for all subsets* $\mathbf{S}$ *of order* k *containing nodes adjacent to* X *(but not containing* Y*) the sample partial correlation* $r_{XY \cdot \mathbf{S}}$ *is not significantly different from zero. (This corresponds to Principle I of the CI Algorithm.) Add the nodes in* $\mathbf{S}$ *to* $\mathbf{DSep}(\mathbf{X}, \mathbf{Y})$.
4. *If any arcs were removed, increment* k *and* `goto` *Step 3.*
5. *For each triple* X, Y, Z *in an undirected chain (such that* X *and* Y *are connected and* Y *and* Z *are connected, but not* X *and* Z*), replace the chain with* $X \rightarrow Y \leftarrow Z$ *if and only if* $Y \notin \mathbf{DSep}(\mathbf{X}, \mathbf{Z})$*. (This corresponds to Principle II of the CI Algorithm.)*
6. *Apply Step 3 of the CI Algorithm.*

The PC algorithm thus begins with a fully connected model, removing arcs whenever the removal can be justified on grounds of conditional independence. One computational trick is to keep track of d-separating nodes (and, therefore, non-d-separating nodes implicitly) during this removal process, for use in Step 5, avoiding the need to search for them a second time. A second trick for reducing complexity of the algorithm is just that, since the partial correlation tests are applied by fixing the values of sets $\mathbf{S}$ of adjacent nodes only and since arcs are removed early on during low-order tests, by the time the algorithm reaches larger orders there should be relatively few such sets to examine. Of course, this depends upon the connectivity of the true model giving rise to the data: dense models will support relatively fewer removals of arcs in Step 3, so the algorithm will be relatively complex. If in practice most models you work with are sparse, then you can reasonably hope for computational gains from the trick. This is a reasonable hope in general, since highly dense networks, being hard to interpret and use in principle, are less likely to be of interest in any case. Certainly the vast majority of networks actually published in the scientific research literature are very much less dense than, for example, the maximal density divided by two.

© 2004 by Chapman & Hall/CRC Press LLC

Compared with the metric learning algorithms, discussed in Chapter 8, the CI and PC Algorithms are very straightforward. Of course, the CI Algorithm is not really an algorithm, since oracles are not generally available. But statistical significance tests could be substituted directly. The PC Algorithm improves upon that option by finding some computational simplifications that speed things up in ordinary cases. But there remain computational difficulties. For one thing, if an arc is removed in error early in the search, the error is likely to cascade when looking at partial correlations of higher order. And as the order of the partial correlations increases, the number of sample correlations that need to be estimated increases dramatically, since every pair of variables involved has a correlation that must be estimated for the significance test. This will inevitably introduce errors in the discovery algorithm for moderately large networks. We should expect such an algorithm to work well on small models with large samples, but not so well on large models with moderately sized samples; and, indeed, we have found this to be the case empirically [65].

6.3.3 Causal discovery versus regression

Regression models aim to reduce the unexplained variation in dependent variables; ordinary least squares regression specifically parameterizes models by computing regression coefficients which minimize unexplained variance. Of course, since the application of Wright's Rule is numerically equivalent, path modeling does the same thing. What causal discovery does, on the other hand, is quite a different matter. Almost no statistician using regression models believes that every independent variable that helps reduce unexplained variation in the dependent variable is actually a relevant causal factor: it is a truism that whenever there is random variation in one variable its sample correlation with another variable subject to random variation will not be identically zero. Hence, the first can be used to reduce, however marginally, "unexplained" variation in the second. So, for a random example, almost certainly variations in seismic activity on Io are apparently correlated with variations in muggings in New York City. But, since it is useless to run around pointing out tiny correlations between evidently causally unrelated events, statisticians wish to rule out such variables.

Regression methods provide no principled way for ruling out such variables. Orthodox statistics claims that any correlation that survives a significance test is as good as any other. Of course, the correlation between Io's seismic events and muggings may not survive a significance test, so we would then be relieved of having to consider it further. However, the probability of a Type I error — getting a significant result when there is no true correlation — is typically set at five percent of cases; so examining any large number of variables will result in introducing spurious causal structure into regressions. This is called the problem of variable selection in statistics. Various heuristics have been invented for identifying spurious causal variables and throwing them out, including some looking for vanishing partial correlations. These variable selection methods, however, have been ad hoc rather than principled; thus, for example, they have not considered the possibility of accidentally inducing correlations by holding fixed common effect variables.

© 2004 by Chapman & Hall/CRC Press LLC

It is only with the concept of causal discovery, via conditional independence learning in particular, that the use of vanishing partial correlations has received any clear justification. Because of the relation between d-separation in causal graphs and conditional independence in probability distributions (or, vanishing partial correlations in linear Gaussian models), we can justify variable selection in causal discovery. In particular, the arc deletion/addition rules of the CI and PC algorithms are so justified. The metric causal learners addressed in Chapter 8 have even better claim to providing principled variable selection, as we shall see.

6.4 Summary

Reichenbach's Principle of the Common Cause suggests that conditional dependencies and independencies arise from causal structure and therefore that an inverse inference from observations of dependency to causality should be possible. In this chapter we have considered specifically the question of whether and how linear causal models can be inferred from observational data. Using the concept of conditional independence and its relation to causal structure, Sewall Wright was able to develop exact relations between parameters representing causal forces (path coefficients) and conditional independence (zero partial correlation, in the case of linear Gaussian models). This allows linear models to be parameterized from observational data. Verma and Pearl have further applied the relation between causality and conditional independence to develop the CI algorithm for discovering causal structure, which Spirtes et al. then implemented in a practical way in the PC algorithm. Thus, the skepticism that has sometimes greeted the idea of inferring causal from correlational structure is defeated pragmatically by an existence proof: a working algorithm exists. Although these methods are here presented in the context of linear models only, they are readily extensible to discrete models, as we shall see in Chapter 8.

6.5 Bibliographic notes

Wright's 1931 paper [303] remains rewarding, especially as an example of lucid applied mathematics. A simple and good introduction to path modeling can be found in Asher's *Causal modeling* [11]. For a clear and simple introduction to ordinary least squares regression, correlation and identifying linear models see Edwards [79]. The PC algorithm, causal discovery program TETRAD II and their theoretical underpinnings are described in Spirtes et al. [265]. The current version of this program is TETRAD IV; details may be found in Appendix B).

© 2004 by Chapman & Hall/CRC Press LLC

6.6 Technical notes

Correlation

The correlation between two variables X and Y is written ρ_{XY} and describes the degree of linear relation between them. That is, the correlation identifies the degree to which values of one can be predicted by a linear function of changes in value of the other. For those interested, the relation between linear correlation and linear algebra is nicely set out by Michael Jordan [134].

Sample correlation is a measurement on a sample providing an estimate of population correlation. Pearson's **product moment correlation coefficient** is the most commonly used statistic:

$$r_{XY} = \frac{S_{XY}}{S_X S_Y}$$

where

$$S_{XY} = \sum_{i=1,\ldots n} \frac{(x_i - \overline{X})(y_i - \overline{Y})}{n}$$

is the estimate of covariance between X and Y in a sample of size n and

$$S_X = \sum_i \frac{(x_i - \overline{X})^2}{n-1}$$

$$S_Y = \sum_i \frac{(y_i - \overline{Y})^2}{n-1}$$

are estimates of standard deviation.

Partial correlation

Partial correlation measures the degree of linear association between two variables when the values of another variable (or set of variables) are held fixed. The partial correlation between X and Y when Z is held constant is written $\rho_{XY.Z}$. The "holding fixed" of Z here is *not* manipulating the value of Z to any value; the association between X and Y is measured while *observed* values of Z are not allowed to vary.

The sample partial correlation is computed from the sample correlations relating each pair of variables:

$$r_{XY.Z} = \frac{r_{XY} - r_{XZ} r_{YZ}}{\sqrt{1 - r_{XZ}^2}\sqrt{1 - r_{YZ}^2}}$$

This can be extended recursively to accommodate any number of variables being partialed out.

© 2004 by Chapman & Hall/CRC Press LLC

Significance tests for correlation

The PC algorithm (or CI algorithm, for that matter) can be implemented with significance tests on sample data reporting whether correlations and partial correlations are vanishing. The standard t test for the product moment correlation coefficient is:

$$t = \frac{r_{XY}\sqrt{n-2}}{\sqrt{1-r_{XY}^2}}$$

with $n-2$ degrees of freedom. For partial correlation the t test is:

$$t = \frac{r_{XY.Z}\sqrt{n-3}}{\sqrt{1-r_{XY.Z}^2}}$$

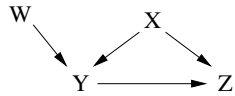
with $n-3$ degrees of freedom. This can be extended to larger sets of partialed variables in the obvious way.

Given a t value and the degrees of freedom, you can look up the result in a "t Table," which will tell you how probable the result is on the assumption that the correlation is vanishing. If that probability is less than some pre-selected value (called α), say 0.05, then orthodox testing theory says to reject the hypothesis that there is no correlation. The test is said to be "significant at the .05 level." In this case, the PC algorithm will accept that there is a real positive (or negative) correlation.

6.7 Problems

Problem 1

Consider the model:



1. Identify all of the active paths between pairs of variables.
2. Use Wright's decomposition rule (in either form) to generate a system of equations for the correlations in terms of the path coefficients.
3. Solve the system of equations — i.e., convert it into a set of equations for the path coefficients in terms of the correlations.
4. Given the following correlation table, compute the path coefficients.

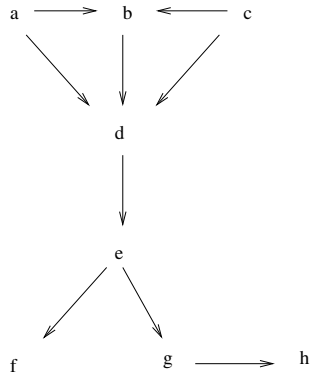
	W	X	Y	Z
W	1			
X	0	1		
Y	.4	.5	1	
Z	.12	.95	.7	1

5. How much of the variation of Z is unexplained? How much for Y?

© 2004 by Chapman & Hall/CRC Press LLC

Problem 2

Consider the model:



Find all the dags which are Markov equivalent. (You can use the CI algorithm to do this.)

Problem 3

If you have not already done so, complete the oracle from the problems in Chapter 2.

Run your d-separation oracle on the additional test network for the CI learning example, `LearningEg.dne`, which can be found in

`http://www.csse.monash.edu.au/bai`

Problem 4

Implement the CI algorithm using your d-separation oracle. That is, your CI learning algorithm should take as input a list of variables and then, using the oracle supplied with some Bayesian network, discover as much of the Bayesian network structure as possible. It should output the network structure in some convenient ascii format (**not** a graph layout, unless you happen to find that easy!). For example, you can generate an alphabetical list of nodes, with their adjacencies listed and the arc types indicated (parent, child or undirected).

Note that this algorithm is exponential in the number of variables. You will have to deal with this in some way to complete the assignment. You may want to implement some kind of heuristic for which subsets $\mathbf{Z}$ are worth looking at relative to a given pair X and Y. Or, you might simply implement a cutoff to prevent your program from examining any more than k subsets, looking at lower order subsets first. Whatever you decide to do, document it.

Run your algorithm on a set of test networks, including at least

- The CI Learning example, `LearningEg.dne`
- `Cancer_Neapolitan.dne`
- `ALARM.dne`

Summarize the results of these experiments.

© 2004 by Chapman & Hall/CRC Press LLC

Problem 5

Instead of, or in addition to, the CI algorithm above, implement the PC algorithm, again using the oracle from Chapter 2 (rather than significance tests). Your PC learning algorithm should take as input a list of variables and then, using the oracle supplied with some Bayesian network, discover as much of the Bayesian network structure as possible. It should output the network structure in some convenient ascii format (**not** a graph layout, unless you happen to find that easy!). For example, you can generate an alphabetical list of nodes, with their adjacencies listed and the arc types indicated (parent, child or undirected).

Run your algorithm on a set of test networks, including at least

- The CI Learning example, `LearningEg.dne`
- `Cancer_Neapolitan.dne`
- `ALARM.dne`

Summarize the results of these experiments.

If you have also implemented the CI algorithm, compare the results of the CI with the PC Algorithm.

Problem 6

Reimplement your oracle-based algorithm, whether CI or PC, using significance testing of partial correlations instead of the oracle, with the input being joint samples across the variables rather than a known Bayesian net. Using a variety of α levels (say, 0.01, 0.05, 0.1), compare the results you get with your oracle-based version.

© 2004 by Chapman & Hall/CRC Press LLC

7

Learning Probabilities

7.1 Introduction

We have just been considering how to learn both causal structure and probability distributions, in the restricted setting of linear models. The CI and PC learning algorithms of Chapter 6 take advantage of conditional independencies in joint samples to discover the qualitative causal structure of a model. Assuming that we then have the right causal structure, we can use Wright's decomposition rule (either Rule 6.1 or Rule 6.2) to parameterize the linear model, giving us equations of the form

$$Y = a_1 X_1 + \ldots a_k X_k + U \tag{7.1}$$

where U describes the residual uncertainty in Y once the best linear prediction using X_i has been made. U is typically taken to be normally distributed from $N(0, \sigma)$ — that is, the random noise left over after getting our best prediction is taken to be described by a normal (Gaussian, bell curve) distribution. All of the parameters in this equation (i.e., the a_i and σ) can be estimated using Wright's method. Since the normal distribution *is* a probability distribution, the result is that we have a probability distribution conditional upon any joint instantiation of the parent set $Parents(Y) = \{X_i\}$.

We now turn to the question of how to parameterize models which are not linear, but **discrete** (or, multinomial) — i.e., whose variables take a fixed number of values, such as *On* and *Off* or *Momma-size*, *Papa-size* and *Baby-size*. Instead of treating causal structure learning and parameter learning together for discrete networks, we defer the structure learning to the next chapter. For now we will assume the causal structures are known to us.

We will first examine how to learn the probability parameters that make up the conditional probability tables of discrete networks when we have non-problematic samples of all the variables. Non-problematic here means that there is no **noise** in the data — all the variables are measured and measured accurately. We will then consider how to handle sample data where some of the variables fail to have been observed, that is when some of the data are incomplete or missing. Finally, we will look at a few methods for speeding up the learning of parameters when the conditional probabilities to be learned depend not just upon the parent variables' values but also upon each other, called **local structure**.

© 2004 by Chapman & Hall/CRC Press LLC

7.2 Parameterizing discrete models

In presenting methods for parameterizing discrete networks we will first consider parameterizing **binomial models** (models with binary variables only) and then generalize the method to parameterizing arbitrary discrete models.

7.2.1 Parameterizing a binomial model

The simplest possible Bayesian network is one of a single binomial variable X. Suppose X reports the outcome of a coin toss, taking the values $heads$ and $tails$. We wish to learn the parameter value $\Theta = \theta$, which is the probability $P(X = heads)$. Suppose we learn exactly and only that the next toss is $heads$. Then by Bayes' theorem:

$$P(\theta|heads) = \beta P(heads|\theta)P(\theta) \tag{7.2}$$

(where β is the inverse of the probability of the evidence). $P(heads|\Theta = \theta) = \theta$, so

$$P(\theta|heads) = \beta\theta P(\theta) \tag{7.3}$$

which multiplication we can see graphically in Figure 7.1. Of course, the observation of heads skews the posterior distribution over $\Theta = \theta$ towards the right, while tails would skew it towards the left.

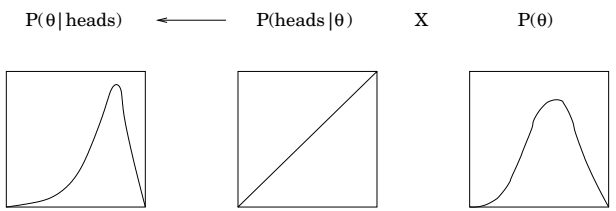


FIGURE 7.1
Updating a binomial estimate in a visual rendition of Bayes' Theorem.

If we get two heads in a row $e =< heads, heads >$ (letting e represent our evidence), then our Bayesian updating yields:

$$P(\theta|e) = \beta\theta^2 P(\theta) \tag{7.4}$$

In general, evidence e consisting of m heads and $n - m$ tails gives us:

$$P(\theta|e) = \beta\theta^m(1-\theta)^{n-m}P(\theta) \tag{7.5}$$

on the assumption that all the coin tosses are **independently identically distributed** (i.i.d.), which means: each toss is independent of every other toss, and each toss is a sample drawn from the very same probability distribution as every other toss.

© 2004 by Chapman & Hall/CRC Press LLC

7.2.1.1 The beta distribution

Equation (7.5) needs a prior distribution over Θ to get started. It doesn't matter what that distribution may be (i.e., the update procedure applies regardless), but it's natural — particularly for automating the learning of Bayesian networks — to choose a distribution that is easy to work with and gives reasonable performance. So, for simplicity we can restrict our prior to the family of **beta distributions** $B(\alpha_1, \alpha_2)$*:

$$P(\theta|\alpha_1, \alpha_2) = \beta\theta^{\alpha_1 - 1}(1-\theta)^{\alpha_2 - 1} \tag{7.6}$$

To identify a particular beta distribution within the family we must set the hyperparameters α_1 and α_2 to positive integers. (A **parameter**, when set, takes us from a class of models to a specific model; a **hyperparameter**, when set, takes us from a superclass [or family] of models to a class of models, distinguished from one another by their parameter values.) We identify *heads* with the value 1 and *tails* with the value 0 (as is common), and again *heads* with α_1 and *tails* with α_2. Figure 7.2 displays two beta distributions for fixed small values of α_1 and α_2, namely $< 2, 2 >$ and $< 2, 8 >$.

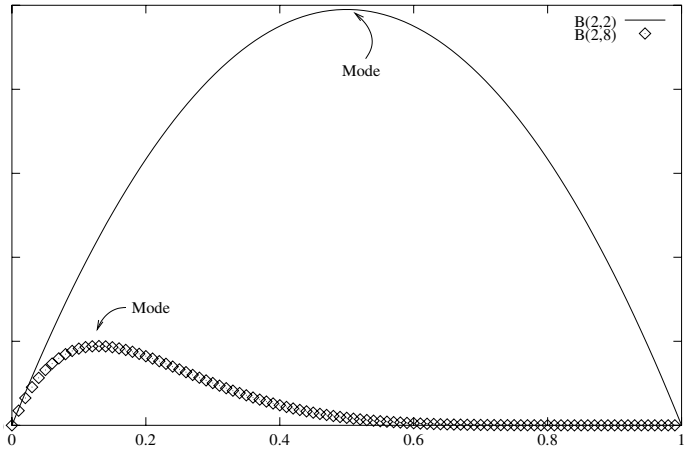


FIGURE 7.2
Two beta distributions: B(2,2) and B(2,8).

The expected value of the beta distribution is:

$$E_{B(\alpha_1,\alpha_2)} = \int \theta P(\theta|\alpha_1, \alpha_2)d\theta = \frac{\alpha_1}{\alpha_1 + \alpha_2} \tag{7.7}$$

* β in this equation remains the normalization constant, but for the beta distribution (and Dirichlet distribution below) it has a specific form, which is given in Technical Notes §7.7.

© 2004 by Chapman & Hall/CRC Press LLC

Having selected a specific beta distribution, then, by Equation (7.5), after observing evidence e of m heads and $n - m$ tails we will have:

$$P(\theta|e, \alpha_1, \alpha_2) = \beta\theta^{\alpha_1+m-1}(1-\theta)^{\alpha_2+(n-m)-1} \tag{7.8}$$

Interestingly, we remain within the family of beta distributions, since this has the same form as Equation (7.6), with α_1 replaced by $\alpha_1 + m$ and α_2 replaced by $\alpha_2 + n - m$. A family of distributions which has this property — where Bayesian updating always keeps you within the same family, but with altered hyperparameters — is called a **conjugate family of distributions**.

So, after we have observed m heads and $n - m$ tails, Bayesian updating will move us from $B(\alpha_1, \alpha_2)$ to $B(\alpha_1 + m, \alpha_2 + n - m)$. Thus, in Figure 7.2 $B(2, 2)$ may represent our prior belief in the value of the binomial parameter for the probability of heads. Then $B(2, 8)$ will represent our posterior distribution over the parameter after six tails are observed, with the posterior mode around 0.10.

The expected result of the next coin toss will, of course, have moved from $E = \alpha_1/(\alpha_1+\alpha_2)$ to $E = (\alpha_1+m)/(\alpha_1+\alpha_2+n)$. Selecting the hyperparameters α_1 and α_2 thus fixes how quickly the estimate of θ adjusts in the light of the evidence. When $\alpha_1 + \alpha_2$ is small, the denominator in the posterior expectation, $\alpha_1 + \alpha_2 + n$, will quickly become dominated by the sample size (number of observations) n. When $\alpha_1 + \alpha_2$ is large, it will take a much larger sample size to alter significantly the prior estimate for θ. Selecting a prior beta distribution with $\alpha_1 + \alpha_2 = 4$ in Figure 7.2 represents a readiness to accept that the coin is biased, with the expectation for the next toss moving from 0.5 to 0.2 after only 6 tails. Compare that with $B(10, 10)$ and $B(10, 16)$ in Figure 7.3, when both the posterior mode and posterior mean (expectation) shift far less than before, with the latter moving from 0.5 to 0.38.

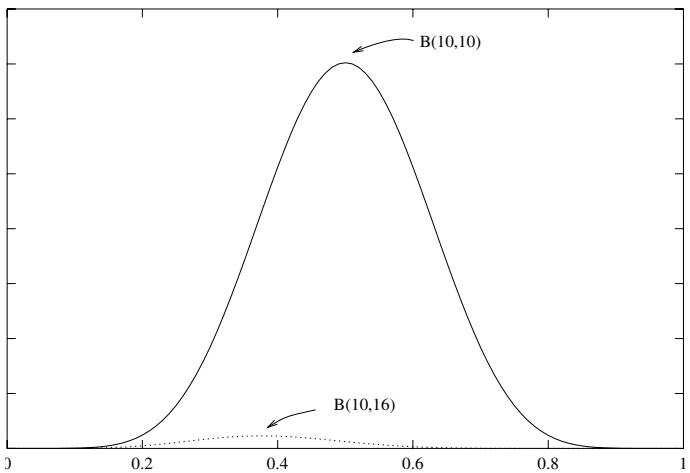


FIGURE 7.3
Two beta distributions: B(10,10) and B(10,16).

© 2004 by Chapman & Hall/CRC Press LLC

In fact, setting the size of $\alpha_1 + \alpha_2$ plays the same role in subsequent learning as would an initial sample of the same size $\alpha_1 + \alpha_2$ (ignoring the fact that our initial beta distribution requires positive hyperparameters). So, an initial selection of values for α_1 and α_2 can be thought of as a kind of "pretend initial sample," leading people to refer to $\alpha_1 + \alpha_2$ as the **equivalent sample size**.

In short, estimating the binomial parameter under these assumptions — i.i.d. samples and a beta prior — just means mixing the prior hyperparameters with the frequency of heads in the sample.

7.2.2 Parameterizing a multinomial model

This process generalizes directly to variables with more than two states — that is, to **multinomial variables** — using another conjugate family of distributions, namely the **Dirichlet** family of distributions. The Dirichlet distribution with τ states is written $D[\alpha_1, \ldots, \alpha_i, \ldots, \alpha_\tau]$ with α_i being the hyperparameter for state i. In direct generalization of the binomial case, the probability of state i is

$$P(X = i) = \frac{\alpha_i}{\sum_{j=1}^{\tau} \alpha_j} \tag{7.9}$$

The learning problem is to estimate the set of parameters $\{\Theta_i = \theta_i : 1 \leq i < \tau\}$. Let $\vec{\theta}$ refer to the vector $< \theta_1, \ldots, \theta_\tau >$. A possible simplification is to assume **local parameter independence:**

$$P(\vec{\theta}) = \prod_{i=1,\ldots,\tau} P(\theta_i) \tag{7.10}$$

that is, that the probability of each state is independent of the probability of every other state. With this assumption, we can update each multinomial parameter in the same way as we did binomial parameters, following Spiegelhalter and Lauritzen [263]. Thus, observing state i moves you from the original distribution $D[\alpha_1, \ldots, \alpha_i, \ldots, \alpha_\tau]$ to $D[\alpha_1, \ldots, \alpha_i + 1, \ldots, \alpha_\tau]$. In this case the equivalent sample size is the original $\sum_{j=1}^{\tau} \alpha_j$.

These techniques provide parameter estimation for BNs with a single node. In order to parameterize an entire network we simply iterate over its nodes:

ALGORITHM 7.1

Multinomial Parameterization (Spiegelhalter and Lauritzen method)

1. *For each node* X_j

   For each instantiation of $Parents(X_j)$*, assign some Dirichlet distribution for the* τ *states of* X_j $D[\alpha_1, \ldots, \alpha_i, \ldots, \alpha_\tau]$

2. *For each node* X_j

© 2004 by Chapman & Hall/CRC Press LLC

For each joint observation of all variables $X_1, \ldots, X_k$

(a) Identify which state i X_j *takes*

(b) Update $D[\alpha_1, \ldots, \alpha_i, \ldots, \alpha_\tau]$ *to* $D[\alpha_1, \ldots, \alpha_i + 1, \ldots, \alpha_\tau]$ *for the distribution corresponding to the parent instantiation in the observation*

Thus, we have a very simple counting solution to the problem of parameterizing multinomial networks. This solution is certainly the most widely used and is available in the standard Bayesian network tools.

The assumptions behind this algorithm are:

1. Local parameter independence, per Equation (7.10).
2. Parameter independence *across* distinct parent instantiations. That is, the parameter values when the parents take one state do not influence the parameter values when parents take a different state.
3. Parameter independence across non-local states. That is, the states adopted by other parts of the network do not influence the parameter values for a node once its parent instantiation is given.
4. The parameter distributions are within a conjugate family of priors; specifically they are Dirichlet distributed.

The third assumption is already guaranteed by the Markov property assumed as a matter of general practice for the Bayesian network as a whole[†]. The first and second assumptions are more substantial and, frequently, wrong. When they are wrong, the implication is that dependencies between parameter values are not being recognized in the learning process, with the result that the information afforded by such dependencies is neglected. The upshot is that Algorithm 7.1 will still work, but it will work more slowly than methods which take advantage of parameter dependencies to re-estimate the values of some parameters given those of others. The algorithm must painstakingly count up values for each and every cell in each and every conditional probability table without any reference to other cells. This slowness of Algorithm 7.1 can be troublesome because many parent instantiations, especially when dealing with large arity (large numbers of joint parent states), may be rare in the data, leaving us with a weak parameterization of the network. We will examine different methods of taking advantage of parameter dependence in probability learning in §7.4 below.

The fourth assumption, that the parameter priors are Dirichlet distributed, enables the application of the simple Algorithm 7.1 to parameterization. Of course, there are infinities of other possible prior distributions over parameters; but choosing outside of the Dirichlet family requires a different estimation algorithm. The exponential family of distributions, which subsumes the Dirichlet family, admits of tractable estimation methods [71]. In any case, choosing inaccurate hyperparameters for the Dirichlet is a more likely source of practical trouble in estimating parameters than

[†] To be sure, the Markov property does not imply parameter independence from the parameters of descendants, so the third assumption has this stronger implication.

© 2004 by Chapman & Hall/CRC Press LLC

the initial restriction to the Dirichlet family. The hyperparameters should be selected so as to reflect what is known or guessed about the probability of each state, as revealed in Equation (7.9), as well as the degree of confidence in those guesstimates, expressed in equivalent sample size.

7.3 Incomplete data

Machine learning from real data very commonly has to deal with data of poor quality. One way in which data are often poor is that the measurements are **noisy**, meaning that many of the attribute (variable) values reported are incorrect. In the next chapter we will look at some information-theoretic techniques which deal with noise.

Another kind of data poverty is when some attribute values are simply missing in some of the joint observations, that is, when the data samples are incomplete. Thus, when responding to surveys some people may fail to state their ages or their incomes, while reporting their other attributes, as in Table 7.1. This section will present a number of techniques for parameterizing a Bayesian network even though some of the data available are corrupted in such a manner.

TABLE 7.1
Example of incomplete data in joint observations of four variables (— indicates a missing value)

Name	Occupation	Income	Automobile
Jones	surgeon	120K	Mercedes
Smith	student	3K	none
Johnson	lawyer	—	Jaguar
Peters	receptionist	23K	Hyundai
Taylor	pilot	—	BMW
—	programmer	50K	BMW

A more extreme kind of incompleteness is simply to be missing all the measurements for some relevant variables. Rectifying that problem — learning causal structure and parameterizing a network lacking all the values for one or more variables — is called **latent variable** (or hidden variable) discovery. The complexities of latent variable discovery will, for the most part, not be addressed in this text; they are under active research in the Bayesian network community. In statistics research there are various known methods for introducing and testing for latent variables, including factor analysis (see, e.g., [174]).

In this section we will first look at the most complete solution to dealing with missing data, namely directly computing the conditional density $P(\theta|\mathbf{Y})$ when the

© 2004 by Chapman & Hall/CRC Press LLC

observed variables $\mathbf{Y}$ are a proper subset of all the variables $\mathbf{V}$ (and so, incomplete). This turns out to be an intractable computation in general, so we move on to consider two approximate solutions.

7.3.1 The Bayesian solution

There is an exact Bayesian answer to the question: What is the conditional density $P(\theta|\mathbf{Y})$? — where the observed variables $\mathbf{Y} \subset \mathbf{V}$ are incomplete. We demonstrate the idea of the answer with the simplest possible scenario. Suppose the set of binary variables $\mathbf{Y}$ are observed and the single binary variable X is unobserved, so that the entire set of variables is $\mathbf{V} = \mathbf{Y} \cup X$. Here we are dealing simply with binomial parameterization. By the chain rule (Theorem 1.3) we have, for any binomial parameter, $P(\theta|\mathbf{Y}) = P(\theta|X, \mathbf{y})P(X|\mathbf{y}) + P(\theta|\neg X, \mathbf{y})P(\neg X|\mathbf{y})$. We can see that, in effect, we must compute each possible way of completing the incomplete data (i.e., by observing X or else by observing $\neg X$) and then find the weighted average across these possible completions. Under the assumptions we have been applying above, both $P(\theta|X, \mathbf{y})$ and $P(\theta|\neg X, \mathbf{y})$ will be beta densities, and $P(\theta|\mathbf{Y})$ will be a linear mixture of beta densities. If the set of unobserved attributes $\mathbf{X}$ contains more than one variable, then the mixture will be more complex — exponentially complex, with the number of products being $2^{|\mathbf{X}|}$.

The generalization of this analysis to multinomial networks is straightforward, resulting in a linear mixture of Dirichlet densities. In any case, the mixed densities must be computed over every possible completion of the data, across all joint samples which are incomplete. This exact solution to dealing with incomplete data is clearly intractable.

7.3.2 Approximate solutions

We will examine two approaches to approximating the estimation of parameters with incomplete data: a stochastic sampling technique, called Gibbs sampling, and an iterative, deterministic algorithm, called expectation maximization (EM). Both techniques make the strong simplifying assumption that the missing data are independent of the observed data. That assumption will frequently be false. For example, in a consumer survey it may be that income data are missing predominately when wealthy people prefer to cover up their wealth; in that case, missing income data will be independent of other, observed values only in the unlikely circumstance that wealth has nothing to do with how other questions are answered. Nevertheless, it is useful to have approximative methods that are easy to use, relying upon the independence assumption.

7.3.2.1 Gibbs sampling

Gibbs sampling is a stochastic procedure introduced by Geman and Geman [91], which can be used to sample the results of computing any function f of $\mathbf{X}$, yielding an estimate of $P(f(\mathbf{X}))$. In particular, it can be used to estimate by sampling the

© 2004 by Chapman & Hall/CRC Press LLC

conditional distribution $P(\mathbf{X}|e)$ where the evidence e is partial. Here we present the Gibbs sampling algorithm in a simple form for computing the expected value of the function $f(\mathbf{X})$, i.e., $E[f(\mathbf{X})]$. At each step we simply compute $f(\mathbf{x})$ (the algorithm assumes we know how to compute this), accumulating the result in sum, and in the end return the average value. To estimate the full distribution of values $P(f(\mathbf{X})|e)$ we only need to alter the algorithm to collect a histogram of values.

Intuitively, Gibbs sampling estimates $P(\mathbf{X})$ by beginning at an arbitrary initial point in the state space of $\mathbf{X}$ and then sampling an adjacent state, with the conditional probability $P(\mathbf{X}|e)$ governing the sampling process. Although we may start out at an improbable location, the probability pressure exerted on the sampling process guarantees convergence on the right distribution (subject to the convergence requirement, described below).

ALGORITHM 7.2

Gibbs Sampling Algorithm for the expected value $E[f]$
Let j index the unobserved variables in the full set of variables $\mathbf{X}$ and $limit$ be the number of sampling steps you wish to take.

0. *(a) Choose any legitimate initial state for the joint $\mathbf{X} = \mathbf{x}$; the observed variables take their observed values and the unobserved variables take any value which is allowed in $P(\mathbf{X})$.*

(b) $sum = 0.0$

(c) $i = 0$

1. While $i < limit$ do

(a) Select the next unobserved X_j

(b) Replace its value with a sample from $P(X_j|\mathbf{X}\backslash X_j)$

(c) $sum \leftarrow sum + f(\mathbf{x})$

(d) $i \leftarrow i + 1$

2. Return sum/i.

In the initialization Step (0.a) it does not matter what joint state for $\mathbf{X}$ is selected, so long as the observed variables in $\mathbf{X}$ are set to their observed values and also the convergence requirement (below) is satisfied. In Step (1.b) we are required to compute the conditional probability distribution over the unobserved X_j given what we know (or guess) about the other variables; that is, we have to compute $P(X_j|\mathbf{X}\backslash X_j)$, which is easy enough if we have a complete Bayesian network. The next version of the algorithm (Algorithm 7.3 just below) provides a substitute for having a complete Bayesian network.

Note that this algorithm is written as though there is a *single* joint observation. In particular, Step (0.a) assumes that variables in $\mathbf{X}$ either are observed or are not, whereas across many joint observations some variables will be both observed in some

© 2004 by Chapman & Hall/CRC Press LLC

and unobserved in others. To implement the algorithm for multiple joint observations, we need only embed it in an iteration that cycles through the different joint observations.

Now we present another variation of the Gibbs sampling algorithm which specifically computes an approximation for multinomial networks to the posterior density $P(\theta|e)$, where e reports a set of incomplete observations. This algorithm assumes Dirichlet priors.

ALGORITHM 7.3

Gibbs Sampling Algorithm for estimating $P(\theta|e)$

Let k *index the sample — i.e.,* x_{jk} *is the value observed for the* j*th variable in the* k*th joint observation in the sample. This algorithm is not presented in complete form: where the phrase "per Cooper & Herskovits" appears, this means the probability function* $P(\cdot)$ *can be computed according to Theorem 8.1 in Chapter 8.*

0. *Complete* $e \rightarrow e^*$ *arbitrarily, choosing any legitimate initial state for the joint samples; set* $limit$ *to the number of sampling steps;* $i = 0$.
1. *While* $i < limit$ *do*
   (a) *For each unobserved* $X_{jk} \in e^* \backslash e$

   Reassign X_{jk} *via a sample from*

   $$P(x_{jk}|e^* \backslash x_{jk}) = \frac{P(x_{jk}, e^* \backslash x_{jk})}{\sum_{x'_{jk}} P(x'_{jk}, e^* \backslash x_{jk})}$$

   per Cooper & Herskovits. Here the denominator sums over all possible values for $X_{jk} = x'_{jk}$.

   This produces a new, complete sample, e^{**}.

   (b) *Compute the conditional density* $P(\theta|e^{**})$ *using Algorithm 7.1.*

   (c) $e^* \leftarrow e^{**}$; $i \leftarrow i + 1$.

2. *Use the average value during sampling to estimate* $P(\theta|e)$.

Convergence requirement. For Gibbs sampling to converge on the correct value, when the sampling process is called **ergodic**, two preconditions must be satisfied (see Madigan and York [180]). In particular:

1. From any state $\mathbf{X} = \mathbf{x}$ it must be possible to sample any other state $\mathbf{X} = \mathbf{x}'$. This will be true if the distribution is positive.
2. Each value for an unobserved variable X_j is chosen equally often (e.g., by round robin selection or uniformly randomly).

© 2004 by Chapman & Hall/CRC Press LLC

7.3.2.2 Expectation maximization

Expectation maximization (EM) is a deterministic approach to estimating θ asymptotically with incomplete evidence, and again it assumes missing values are independent of the observed values; it was introduced by Dempster et al. [74]. EM returns a point estimate $\hat{\theta}$ of θ, which can either be a **maximum likelihood** (ML) estimate (i.e., one which maximizes $P(e|\hat{\theta})$) or a **maximum aposteriori probability** (MAP) estimate (i.e., one which maximizes $P(\hat{\theta}|e)$, taking the mode of the posterior density). First, a general description of the algorithm:

ALGORITHM 7.4
Expectation Maximization (EM)

0. *Set $\hat{\theta}$ to an arbitrary legal value; select a desired degree of precision ϵ for $\hat{\theta}$; set the update value $\hat{\theta}'$ to an illegally large value (e.g.,* MAXDOUBLE*, ensuring the loop is executed at least once).*

While $|\hat{\theta}' - \hat{\theta}| > \epsilon$ do:

$\hat{\theta} \leftarrow \hat{\theta}'$ *(except on the first iteration)*

1. **Expectation Step:** *Compute the probability distribution over missing values:*

$$P(e^*|e, \hat{\theta}) = \frac{P(e|e^*, \hat{\theta})P(e^*|\hat{\theta})}{\sum_{e^*} P(e|e^*, \hat{\theta})P(e^*|\hat{\theta})}$$

2. **Maximization Step:** *Compute the new ML or MAP estimate $\hat{\theta}'$ given $P(e^*|e, \hat{\theta})$.*

Step 1 is called the expectation step, since it is normally implemented by computing an expected **sufficient statistic** for the missing e^*, rather than the distribution itself. A sufficient statistic is a statistic which summarizes the data and which itself contains all the information in the data relevant to the particular inference in question. Hence, s is a sufficient statistic for θ relative to e^* if and only if θ is independent of e^* given the statistic — i.e., $\theta \perp\!\!\!\perp e^*|s$. Given such a statistic, the second step uses it to maximize a new estimate for the parameter being learned. The EM algorithm generally converges quickly on the best point estimate for the parameter; however, it is a best estimate locally and may not be the best globally — in other words, like hill climbing, EM will get stuck on local maxima [74]. The other limiting factor is that we obtain a point estimate of θ and not a probability distribution.

We now present EM in its maximum likelihood (ML) and maximum aposteriori (MAP) forms.

© 2004 by Chapman & Hall/CRC Press LLC

ALGORITHM 7.5

Maximum Likelihood EM

ML estimation of θ given incomplete e.

0. Set $\hat{\theta}$ arbitrarily; select a desired degree of precision ϵ for $\hat{\theta}$; set the update value $\hat{\theta}'$ to MAXDOUBLE.

While $|\hat{\theta}' - \hat{\theta}| > \epsilon$ do:

$\hat{\theta} \leftarrow \hat{\theta}'$ *(except on the first iteration)*

1. Compute the expected sufficient statistic for e^*:

$$E_{P(X|e,\hat{\theta})} N_{ijk} = \sum_{l=1}^{N} P(x_{ik}, Parents(X_{ij})|y_l, \hat{\theta})$$

N_{ijk} counts the instances of possible joint instantiations of X_i and $Parents(X_i)$, which are indexed by k (for X_i) and j (for $Parents(X_i)$). These expected counts collectively (across all possible instantiations for all variables) provide a sufficient statistic for e^*. For any one N_{ijk} this is computed by summing over all (possibly incomplete) joint observations y_l the probability on the right hand side (RHS). Since in this step we have a (tentative) estimated parameter $\hat{\theta}$ and a causal structure, we can compute the RHS using a Bayesian network.

2. Use the expected statistics as if actual; maximize $P(e^*|\hat{\theta}')$ using

$$\hat{\theta}'_{ijk} = \frac{E_{P(X|e,\hat{\theta})} N_{ijk}}{\sum'_k E_{P(X|e,\hat{\theta})} N_{ijk'}}$$

ALGORITHM 7.6

Maximum Aposteriori Probability EM

0. Set $\hat{\theta}$ arbitrarily; select a desired degree of precision ϵ for $\hat{\theta}$; set the update value $\hat{\theta}'$ to MAXDOUBLE.

While $|\hat{\theta}' - \hat{\theta}| > \epsilon$ do:

$\hat{\theta} \leftarrow \hat{\theta}'$ *(except on the first iteration)*

1. Compute the expected sufficient statistic for e^*:

$$E_{P(X|e,\hat{\theta})} N_{ijk} = \sum_{l=1}^{N} P(x_{ik}, Parents(X_{ij})|y_l, \hat{\theta})$$

(See Algorithm 7.5 Step 1 for explanation.)

2. Use the expected statistics as if actual; maximize $P(\hat{\theta}'|e^*)$ using

$$\hat{\theta}'_{ijk} = \frac{\alpha_{ijk} + E_{P(X|e,\hat{\theta})} N_{ijk}}{\sum_{k'} \alpha_{ijk'} + E_{P(X|e,\hat{\theta})} N_{ijk'}}$$

where α_{ijk} is the Dirichlet parameter.

© 2004 by Chapman & Hall/CRC Press LLC

7.3.3 Incomplete data: summary

In summary, when attribute values are missing in observational data, the optimal method for learning probabilities is to compute the full conditional probability distribution over the parameters. This method, however, is exponential in the arity of the joint missing attribute measurements, and so computationally intractable. There are two useful approximation techniques, Gibbs sampling and expectation maximization, for asymptotically approaching the best estimated parameter values. Both of these require strong independence assumptions — especially, that the missing values are independent of the observed values — which limit their applicability. The alternative of actively modeling the missing data, and using such models to assist in parameterizing the Bayesian network, is one which commends itself to further research. In any case, the approximation techniques are a useful start.

7.4 Learning local structure

We now turn to a different kind of potential dependence between parameters: not between missing and observed values, but between different observed values. Algorithm 7.1, as you will recall, assumed that the different states which a child variable takes under different parent instantiations are independent of each other, with the consequence that when there *are* dependencies, they are ignored, resulting in slower learning time. When there are dependencies between parameters relating the parents and their child, this is called **local structure**, in contrast to the broader structure specified by the arcs in the network.

7.4.1 Causal interaction

One of the major advantages of Bayesian networks over most alternative uncertainty formalisms (such as PROSPECTOR [78] and Certainty Factors [34]) is that Bayesian networks allow, but do not require, conditional independencies to be modeled. Where there are dependencies, of any complexity, they can be specified to any degree required. And there are many situations with local dependencies, namely all those in which there is at most limited **causal interaction** between the parent variables. To take a simple example of interaction: one might ingest alkali, and die; one might instead ingest acid, and die; but if one ingests both alkali and acid together (to be sure, only if measured and mixed fairly exactly!) then one may well not die. That is an interaction between the two potential causes of death. When two parent causes fully interact, each possible instantiation of their values produces a probability distribution over the child's values which is entirely independent of all their other distributions. In such a case, the full power, and slowness, of the Spiegelhalter and Lauritzen method of learning CPTs (Algorithm 7.1) is required.

The most obvious case of local structure is that where the variables are continuous

© 2004 by Chapman & Hall/CRC Press LLC

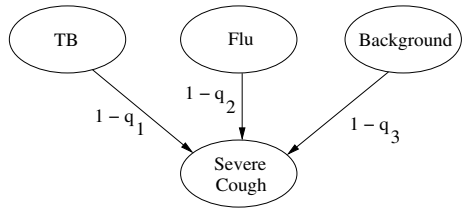


FIGURE 7.4
A noisy-or model.

and the child is an additive linear function of its parents, as in path models. In this case, the magnitude of the child variable is influenced independently by the magnitudes of each of its parents. And the learning problem (solved already in Chapter 6) is greatly reduced: one parameter is learned for each parent, so the learning problem is linear in the number of parents, rather than exponential.

We shall now briefly consider three ways of *modeling* local structure in the CPTs of discrete models and of taking advantage of such structure to learn parameterizations faster, namely with noisy-or connections, classification trees and graphs, and with logit models. Each of these model different kinds of non-interactive models.

7.4.2 Noisy-or connections

Noisy-or models are the most popular for dealing with non-interactive binomial causal factors. Figure 7.4 shows a noisy-or model of severe coughing. This model assumes that the illnesses *TB* and *Flu* are independent of each other and that each has a probability $(1 - q_i)$ of causing the coughing which is independent of the other causes and some background (unattributed) probability of *Severe Cough*. Thus, the q_i parameters of the model can be thought of as the probability of each cause *failing* — it is the "noise" interfering with the cause. Since they are required to operate independently, the CPT relating the three causal factors can be easily computed, on the assumption that the *Background* is always active (*On*), as in Table 7.2.

Thus, to parameterize the noisy-or model we need find only three parameters versus the four in the CPT. Although that may not be an impressive savings in this particular case, once again the simpler model has one parameter per parent, and so the task grows linearly rather than exponentially.

TABLE 7.2
CPT for *Severe Cough* generated from noisy-or parameters

$Severe\ Cough$	TB, Flu	$TB, \neg Flu$	$\neg TB, Flu$	$\neg TB, \neg Flu$
F	$q_1 q_2 q_3$	$q_1 q_3$	$q_2 q_3$	q_3
T	$1 - q_1 q_2 q_3$	$1 - q_1 q_3$	$1 - q_2 q_3$	$1 - q_3$

© 2004 by Chapman & Hall/CRC Press LLC

Algorithm 7.1 can be readily adapted to learning noisy-or parameters: first, learn the probability of the effect given that all parent variables (other than *Background*) are absent (q_3 in our example); then learn the probability of each parent in the absence of all others, dividing out the *Background* parameter (q_3). Since all causal factors operate independently, we are then done.

7.4.3 Classification trees and graphs

A more general technique for learning local structures simpler than a full CPT is to apply the mature technology of **classification tree** and graph learning to learning the local structure[‡]. Classification trees are made up of nodes representing the relevant attributes. Branches coming off a node represent the different values that attribute may take. By following a branch out to its end, the leaf node, we will have selected values for all of the attributes represented in the branch; the leaf node will then make some prediction about the target class. Any instance which matches all of the selected values along a branch will be predicted by the corresponding leaf node. As an example, consider the classification tree for the acid-alkali ingestion problem in Figure 7.5. This tree shows that the ingestion of *Alkali* without the ingestion of *Acid* leads to a 0.95 probability of *Death*; that is, $P(Death|Alkali, \neg Acid) = 0.95$. Every other cell in the CPT for the *Death* variable can be similarly computed. Indeed, any classification tree which has a node for each parent variable along every branch, and which splits those nodes according to all possible values for the variable, will (at the leaf nodes) provide every probability required for filling in the CPT.

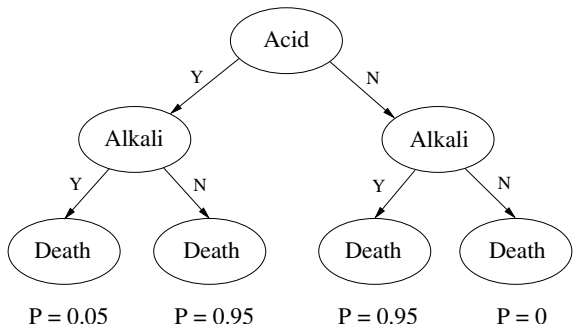


FIGURE 7.5
A classification tree for acid and alkali.

[‡]In much of the AI community these are called decision trees and graphs. However, in the statistics community they are called classification trees and graphs. We prefer the latter, since decision trees have a prior important use in referring to representations used for decision making under uncertainty, as we have done ourselves in Chapter 4.

© 2004 by Chapman & Hall/CRC Press LLC

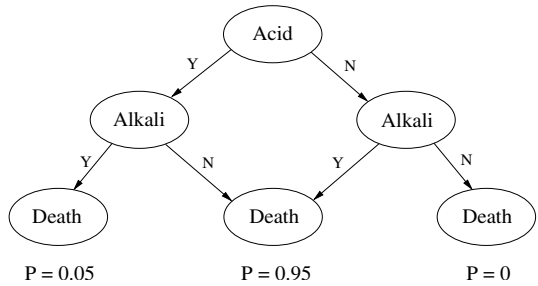


FIGURE 7.6
A classification graph for acid and alkali.

In this case (and many others) the equivalent **classification graph**, which allows branches to *join* as well as split at attribute nodes, is simpler, as in Figure 7.6. Since the classification graph combines two branches, it also has the advantage of combining all the sample observations matching the attributes along the two branches, providing a larger sample size when estimating the corresponding parameter.

One of the major aims of classification tree and graph learning is to minimize the complexity of the representation — of the tree (graph) which is learned. More exactly, beginning already with the Quinlan's ID3 [226], the goal has been to find the right trade-off between model complexity and fit to the data. This **complexity trade-off** is a recurring theme in machine learning, and we shall continue to encounter it through the remainder of this text. Quinlan's approach to it was to use a simple information-theoretic measure to choose at each step in building the tree the optimal node for splitting, and he stopped growing the tree when further splits failed to improve anticipated predictive accuracy. The result was that the least predictively useful attributes were unused. From the point of view of filling in a CPT, this is equivalent to merging CPT cells together when the probabilities are sufficiently close (or, to put it another way, when the probabilities in the cells are not sufficiently distinct for the available evidence to distinguish between them). More recent classification tree and graph learners play the same trade-off in more sophisticated ways (e.g., [211, 227]).

Should the parents-to-child relation have local structure, so that some CPT entries are strict functions of others, the classification tree can take advantage of that. Current methods take advantage of the local structure to produce smaller trees just in case the probabilities in distinct cells are approximately identical, rather than standing in any other functional relation [30].

The technology of classification tree learning is, or ought to be, the technology of learning probabilities, and so its fruits can be directly applied to parameterizing Bayesian networks[§].

[§]The reason for the reservation here is just that many researchers have thought that the classifications in the leaf nodes ought to be *categorical* rather than probabilistic. We have elsewhere argued that this is a mistake [157].

© 2004 by Chapman & Hall/CRC Press LLC

7.4.4 Logit models

Log-linear approaches to representing local structure apply logarithmic transformations to variables or their probability distributions before looking for linear relations to explain them. One of this kind is the logit model.

Suppose we have the binomial variables X, Y, Z in the v-structure $X \rightarrow Z \leftarrow Y$. A **logit model** of this relationship is:

$$\log \frac{P(Z=1|x,y)}{P(Z=0|x,y)} = a + bx + cy + dxy \tag{7.11}$$

This models any causal interaction between X and Y explicitly. The causal effect upon Z is decomposed into terms of three different orders[¶]:

- Order 0 term (a): Identifies the propensity for Z to be true independent of the parents' state.
- Order 1 terms (b, c): Identify the propensity for Z dependent upon each parent independent of the other.
- Order 2 term (d): Identifies the propensity for Z dependent upon the interaction of both parents.

If we have a **saturated** logit model, one where all parameters are non-zero, then clearly we have a term (d) describing a causal interaction between the parents. In that case the complexity of the logit model is equivalent to that of a full CPT; and, clearly, the learning problem is equally complex.

On the other hand, if we have an unsaturated logit model, and in particular when the second-order term is zero, we have a simpler logit model than we do a CPT, resulting in a simpler learning problem. Again, as with classification trees and noisy-or models, learning fewer parameters allows us to learn the parameters with greater precision — or equal precision using smaller samples — than with full CPT learning.

If we are learning a first-order model (i.e., when $d = 0$), some simple algebra reveals the relation between the CPT and the logit parameters, given in Table 7.3.

7.4.5 Dual model discovery

All of these models of local structure — noisy-or, classification graphs and logit models — allow the easy expression of a prior preference for simpler models over more complex models. Dual model discovery refers to causal discovery which searches for models which can learn local structure of one of these types, but also can fall back upon a full CPT characterization when the discovered local structure turns out to be no simpler than that. So far as we know, only CaMML (§8.5) does dual model discovery at this point: it scores logit models versus CPTs and chooses the local structure which provides the greatest posterior probability, with prior probabilities favoring the less saturated logit models [202]. Ideally, of course, this would

[¶] The order refers to the number of variables in a product. For example, in the product dxy d is an order 2 term.

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 7.3
CPT for the first-order logit model

X	Y	Z
0	0	$\frac{\exp(a)}{1+\exp(a)}$
1	0	$\frac{\exp(a+b)}{1+\exp(a+b)}$
0	1	$\frac{\exp(a+c)}{1+\exp(a+c)}$
1	1	$\frac{\exp(a+b+c)}{1+\exp(a+b+c)}$

be generalized to "multiple model discovery," supporting any of the above model types and more.

7.5 Summary

Parameterizing discrete Bayesian networks when the data are not problematic (no data values are missing, the parameter independence assumptions hold, etc.) is straightforward, following the work of Spiegelhalter and Lauritzen leading to Algorithm 7.1. That algorithm has been incorporated into many Bayesian network tools. The Gibbs sampling and EM algorithms for estimating parameters in the face of missing data are also fairly straightforward to apply, so long as their assumptions are satisfied — especially, that missing values are independent of observed values. What to do when the simplifying assumptions behind these methods fail is not clear and remains an area of active research. Another applied research area is the learning of local structure in conditional probability distributions, where classification trees and graphs, noisy-or models and logit models can all be readily employed.

7.6 Bibliographic notes

An excellent concise and accessible introduction to parameter learning is David Heckerman's tutorial [101], published in Michael Jordan's anthology *Learning in Graphical Models* [135]. That collection includes numerous articles elaborating

© 2004 by Chapman & Hall/CRC Press LLC

ideas in this chapter, including an overview of **Markov Chain Monte Carlo** methods, such as Gibbs sampling, by David Mackay [176] and a discussion of learning local structure by Friedman and Goldszmidt [86]. For more thorough and technically advanced treatments of parameterizing discrete networks, see the texts by Cowell et al. [61] and Neapolitan [200].

7.7 Technical notes

We give the formal definitions of the beta and Dirichlet distributions, building upon some preliminary definitions.

Gamma function

The gamma function generalizes the factorial function. Limiting it to integers alone, it can be identified by the recurrence equation:

$$\Gamma(n+1) = \begin{cases} n\Gamma(n) & n > 1 \\ 1 & n = 1 \end{cases} \tag{7.12}$$

If n is a positive integer, then $\Gamma(n+1) = n!$

Beta distribution

$\Theta = \theta$ has the beta distribution with hyperparameters $\alpha_1, \alpha_2 > 0$ if its density function is

$$f(\theta) = \begin{cases} \frac{\Gamma(\alpha_1+\alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)}\theta^{\alpha_1-1}(1-\theta)^{\alpha_2-1} & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{7.13}$$

From this, we see that the normalization factor β in Equation (7.6) is $\frac{\Gamma(\alpha_1+\alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)}$.

Dirichlet distribution

$\Theta = \vec{\theta}$ is Dirichlet distributed with hyperparameters $\alpha_1, \ldots, \alpha_\tau > 0$ if its density function is

$$f(\Theta) = \frac{\Gamma\left(\sum_{i=1}^{\tau} \alpha_i\right)}{\prod_{i=1}^{\tau} \Gamma(\alpha_i)} \prod_{i=1}^{\tau} \theta_i^{\alpha_i - 1} \tag{7.14}$$

when $\theta_1, \ldots, \theta_\tau \geq 0$ and $\sum_{i=1}^{\tau} \theta_i = 1$.

© 2004 by Chapman & Hall/CRC Press LLC

7.8 Problems

Distribution Problems

Problem 1

Suppose your prior distribution for the probability of heads of a coin in your pocket is B(2,2). Toss the coin ten times. Assuming you update your distribution as in §7.2.1, what is your posterior distribution? What is the expected value of tossing the coin on your posterior distribution? Select a larger equivalent sample size for your starting point, such as B(10,10). What then is the expected value of a posterior toss?

Problem 2

Suppose your prior Dirichlet distribution for the roll of a die is $D[2, 2, 2, 2, 2, 2]$ and that you update this distribution as in §7.2.2. Roll a die ten times and update this. Is the posterior distribution flat? How might you get a flatter posterior distribution?

Experimental Problem

Problem 3

In this problem you will analyze a very simple artificially created data set from the book Web site

`http://www.csse.monash.edu.au/bai.html`

which was created by a v-structure process $X \rightarrow Y \leftarrow Z$ — i.e., one with three variables of which one is the child of the other two which are themselves not directly related. However, it will be instructive if you *also* locate a real data set generated by a similar v-structure process and answer the questions for both data sets.

Parameterize the Bayesian network $X \rightarrow Y \leftarrow Z$ from the data set in at least two of the following ways:

- using the algorithm for the full CPT, that is, Algorithm 7.1
- using a noisy-or parameterization
- using a classification tree algorithm, such as J48 (available from the WEKA Web site: `http://www.cs.waikato.ac.nz/~ml/weka/`)
- using an order 1 logit model

Compare the results. Which parameterization fits the data better? For example, which one gives better classification accuracy?

Since in answering this last question you have (presumably) used the very same data both to parameterize the network and to test it, a close fit to the data may be more an indication of overfitting than of predictive accuracy. In order to test a model's **generalization accuracy** you can divide the data set into a training set and a test set, using only the former to parameterize it and the latter to test predictive (classification) accuracy (see Part II introduction).

© 2004 by Chapman & Hall/CRC Press LLC

Programming Problems

Problem 4

Implement the multinomial parameterization algorithm (7.1). Test it on some of the data sets on the book Web site and report the results.

Problem 5

Implement the Gibbs Sampling algorithm (7.3) for estimating parameters with incomplete data. The missing Cooper & Herskovits computations can be filled in by copying the Lisp function for them on the book Web site, or by skipping forward and implementing Equation (8.3) from Chapter 8. Try your algorithm out using some of the data sets with missing values from the book Web site and report the results. How many sampling steps are needed to get good estimates of the parameters in the different cases? How do you know when you have a good estimate for a parameter?

Problem 6

1. Implement maximum likelihood expectation maximization (Algorithm 7.5).
2. Implement MAP expectation maximization (Algorithm 7.6).

Try your algorithm(s) out using some of the data sets with missing values from the book Web site and report the results. Note that there is Lisp code available on the book Web site for computing the expected sufficient statistics for these particular cases. Use a variety of different convergence tests — values for ϵ — and report the relation between the number of iterations and ϵ.

© 2004 by Chapman & Hall/CRC Press LLC

8

Learning Discrete Causal Structure

8.1 Introduction

In Chapter 6 we saw how linear causal structure can be learned and how such structures can be parameterized. In the last chapter we saw how to parameterize a discrete (multinomial) causal structure with conditional probability tables, disregarding how the structure might have been found. Here we will complete the picture of the machine learning of Bayesian networks by considering how to automate the learning of discrete causal structures.

To be sure, we already have in hand a clear and plausible method for structure learning with discrete variables: the PC algorithm (Algorithm 6.2) of TETRAD II is easily extended to cope with the discrete case. That algorithm implements the Verma-Pearl constraint-based approach to causal learning by discovering vanishing partial correlations in the data and using them to find direct dependencies (Step 1 of Verma-Pearl) and v-structure (Step 2). But instead of employing a test for partial correlations between continuous variables, we can substitute a χ^2 significance test (see §8.10) comparing $P(Y = y|X = x, Z = z)$ with the expected value if $X \perp\!\!\!\perp Y|Z$, namely $P(Y = y|Z = z)$ — where the $P(\cdot)$ values are actually estimates based on the sample. Thus, we can test directly for conditional independencies with discrete data. This is just what Spirtes et al. in fact do with TETRAD II [265, p. 129].

What we will look at in this chapter is the learning of causal structure using **Bayesian metrics**, rather than orthodox statistical tests. In other words, the algorithms here will search the causal model space $\{h_i\}$, with some metric like $P(\cdot|e)$ in hand — or, some approximation to that conditional probability function — aiming to select an h_i that maximizes the function. So, there are two computationally difficult tasks these learners need to perform. First, they need to compute their metric, scoring individual hypotheses. This scoring procedure is often itself computationally intractable. Second, they need to search the space of causal structures, which, as we saw in §6.2.3, is exponential. Most of the search methods applied have been variants of greedy search, and we will spend little time discussing them. We describe a more interesting stochastic search used with the MML metric in more detail, partly because the search process itself results in an important simplification to the metric.

We will be presenting the metric learning algorithms in roughly chronological order, but also in roughly conceptual order, in that the later ones often build upon the earlier ones.

© 2004 by Chapman & Hall/CRC Press LLC

8.2 Cooper & Herskovits' K2

The first significant attempt at a Bayesian approach to learning discrete Bayesian networks without topological restrictions was made by Cooper and Herskovits in 1991 [53]. Their approach is to compute the metric for individual hypotheses, $P(h_i|e)$, by brute force, by turning its computation into a combinatorial counting problem. This led to their causal discovery program, K2. Because the counting required is largely the counting of possible instantiations of variables and parent sets, the technique is intrinsically restricted to discrete networks. Other restrictions will become apparent as we develop the method.

Since our goal is to find that h_i which maximizes $P(h_i|e)$, we can satisfy this by maximizing $P(h_i, e)$, as we can see from Bayes' theorem:

$$\begin{aligned} P(h_i|e) &= \frac{P(e|h_i)P(h_i)}{P(e)} \\ &= \frac{P(h_i, e)}{P(e)} \\ &= \beta P(h_i, e) \end{aligned}$$

where β is a (positive) normalizing constant.

To get the combinatorics (i.e., counting) to work we need some simplifying assumptions. Cooper and Herkovits start with these fairly unsurprising assumptions:

1. The data are joint samples and all variables are discrete. So:

$$P(h_i, e) = \int_\theta P(e|h_i, \theta) f(\theta|h_i) P(h_i) d\theta \tag{8.1}$$

   where θ is the parameter vector (e.g., conditional probabilities) and $f(\cdot|h_i)$ is a prior density over parameters conditional upon the causal structure.

2. Samples are independently and identically distributed (i.i.d.). That is, for K sample cases, and breaking down the evidence e into its K components e_k,

$$P(e|h_i, \theta) = \prod_{k=1}^{K} P(e_k|h_i, \theta)$$

   Hence, by substitution into (8.1)

$$P(h_i, e) = P(h_i) \int_\theta \prod_{k=1}^{K} P(e_k|h_i, \theta) f(\theta|h_i) d\theta \tag{8.2}$$

Cooper and Herkovits next make somewhat more problematic simplifications, which are nevertheless needed for the counting process to work.

© 2004 by Chapman & Hall/CRC Press LLC

3. The data contain no missing values. If, in fact, they do contain missing values, then they need to be filled in, perhaps by the Gibbs sampling procedure from Chapter 7.
4. For each variable X_k in h_i and for each instantiation of its parents $\pi(X_k)$, $P(X_k = x|h_i, \theta, \pi(X_k))$ is uniformly distributed over possible values $X_k = x^*$.
5. Assume the uniform prior over the causal model space; i.e., $P(h_i) = \frac{1}{|\{h_i\}|}$.

These last two assumptions are certainly disputable. Assumption 4 does not allow causal factors to be *additive*, let alone interactive! Despite that, it might be reasonable to hope that even this false assumption may not throw the causal structure search too far in the wrong direction: the qualitative fact of dependency between parent and child variables is likely insensitive to the precise quantitative relation between them. And if the qualitative causal structure can be discovered, the quantitative details representing such interactions — the parameters — can be learned as in the last chapter. Of course, if relevant prior knowledge is available, this uniformity assumption can be readily dismissed by employing non-uniform Dirchlet priors over the parameter space, as discussed in Chapter 7.

Something similar might be said of the fifth assumption, that the prior over causal models is uniform: that crude though it may be, it is not likely to throw the search so far off that the best causal models are ultimately missed. However, we shall below introduce a specific reason to be skeptical of this last assumption.

We now have the necessary ingredients for Cooper and Herskovits' main result. (For the proof itself see [53].)

Theorem 8.1 *(Cooper and Herkovits, 1991)*
Under the assumptions above, the joint probability is:

$$P_{CH}(h_i, e) = P(h_i) \prod_{k=1}^{N} \prod_{j=1}^{|\Phi_k|} \frac{(s_k - 1)!}{(S_{kj} + s_k - 1)!} \prod_{l=1}^{s_k} \alpha_{kjl}! \tag{8.3}$$

Where

- N is the number of variables.
- $|\Phi_k|$ is the number of assignments possible to $\pi(X_k)$.
- s_k is the number of assignments possible to X_k.
- α_{kjl} is the number of cases in sample where X_k takes its l-th value and $\pi(X_k)$ takes its j-th value.
- S_{kj} is the number of cases in the sample where $\pi(X_k)$ takes its j-th value (i.e., $\sum_{l=1}^{s_k} \alpha_{kjl}$).

*Note that $\pi(X)$ in this chapter refers to $Parents(X)$. We use it here to shorten equations; it has nothing to do with the message passing of Part I.

© 2004 by Chapman & Hall/CRC Press LLC

Each of these values is the result of some counting process. The point is that, with this theorem, computing $P(h_i, e)$ has become a straightforward counting problem, and is equal to $P(h_i)$ times a simple function of the number of assignments to parent and child variables and the number of matching cases in the sample. Furthermore, Cooper and Herskovits showed that this computation of $P_{CH}(h_i, e)$ is polynomial, i.e., computing $P_{CH}(h_i, e)$ given a particular h_i is tractable under the assumptions so far.

Unfortunately, while the metric may be tractable, we still have to *search* the space $\{h_i\}$, which we know is exponentially large. At this point, Cooper and Herskovits go for a dramatic final simplifying assumption:

6. Assume we know the temporal ordering of the variables.

If we rely on this assumption, the search space is greatly reduced. In fact, for any pair of variables either they are connected by an arc or they are not. Given prior knowledge of the ordering, we need no longer worry about arc orientation, as that is fixed. Hence, the model space is determined by the number of pairs of variables: two raised to that power being the number of possible skeleton models. That is, the new hypothesis space has size only $2^{C_2^N} = 2^{(N^2-N)/2}$. The K2 algorithm simply performs a greedy search through this reduced space. This reduced space remains, of course, exponential.

In any case, so far we have in hand two algorithms for discovering discrete causal structure: TETRAD (i.e., PC with a χ^2 test) and K2.

8.2.1 Learning variable order

Our view is that reliance upon the variable order being provided is a major drawback to K2, as it is to many other algorithms we will not have time to examine in detail (e.g., [35, 27, 273, 179])[†]. Why should we care? It is certainly the case that in many problems we have either a partial or a total ordering of variables in pre-existing background knowledge, and it would be foolish not to use all available information to aid causal discovery. Both TETRAD II and CaMML, for example, allow such prior information to be used to boost the discovery process (see 8.6.3). But it is one thing to *allow* such information to be used and quite another to *depend upon* that information. This is a particularly odd restriction in the domain of causal discovery, where it is plain from Chapter 6 that a fair amount of information about causal ordering can be learned directly from the data, using the Verma-Pearl CI algorithm.

In principle, what artificial intelligence is after is the development of an agent which has some hope of overcoming problems on its own, rather than requiring engineers and domain experts to hold its hand constantly. If *intelligence* is going to be engineered, this is simply a requirement. One of the serious impediments to the success of first-generation expert systems in the 1970s and 80s was that they were brittle: when the domain changed, or the problem focus changed to include anything

[†] We should point out that there are again many others in addition to our CaMML which do not depend upon a prior variable ordering, such as TETRAD, MDL (§8.3), GES [188].

© 2004 by Chapman & Hall/CRC Press LLC

new, the systems would break. They had little or no ability to adapt to changing circumstances, that is, to learn. The same is likely to be true of any Bayesian expert system which requires human experts continually to assist it by informing it of what total ordering it should be considering. It is probably fair to say that learning the variable ordering is half the problem of causal discovery: if we already know A comes before B, the only remaining issue is whether there is a direct dependency between the two — so, half of the Verma-Pearl algorithm in constraint-based approach becomes otiose, namely Principle II. Finally, any algorithm which depends upon being provided the total ordering to get started will not scale up, since the number of orderings consistent with a dag is apparently exponential (see [32]).

8.3 MDL causal discovery

Minimum Description Length (MDL) inference was invented by Jorma Rissanen [235], based upon the Minimum Message Length (MML) approach invented by Wallace [290] in 1968. Both techniques are inspired by information theory and were anticipated by Ray Solomonoff in interesting early work on information-theoretic induction [260]. All of this work is closely related to foundational work on complexity theory, randomness and the interpretation of probability (see [287, 154, 40]).

The basic idea behind both MDL and MML is to play a tradeoff between model simplicity and fit to the data by minimizing the length of a *joint* description of the model and the data assuming the model is correct. Thus, if a model is allowed to grow arbitrarily complex, and if it has sufficient representational power (e.g., sufficiently many parameters), then eventually it will be able to record directly all the evidence that has been gathered. In that case, the part of the message communicating the data given the model will be of length zero, but the first part communicating the model itself will be quite long. Similarly, one can communicate the simplest possible model in a very short first part, but then the equation will be balanced by the necessity of detailing *every* aspect of the data in the second part of the message, since none of that will be implied by the model itself. Minimum encoding inference seeks a golden mean between these two extremes, where any extra complexity in the optimal model is justified by savings in inferring the data from the model.

In principle, minimum encoding inference is inspired by Claude Shannon's measure of information (see Figure 8.1).

Definition 8.1 Shannon information measure

$$I(m) = -\log P(m)$$

Applied to joint messages of hypothesis and evidence:

$$I(h, e) = -\log P(h, e) \tag{8.4}$$

© 2004 by Chapman & Hall/CRC Press LLC

Shannon's concept was inspired by the hunt for an efficient code for telecommunications; his goal, that is, was to find a code which maximized use of a telecommunications channel by minimizing expected message length. If we have a coding scheme which satisfies Shannon's definition 8.1, then we have what is called an **efficient code**. Since efficient codes yield probability distributions (multiply by -1 and exponentiate), efficiency requires observance of the probability axioms. Indeed, in consequence we can derive the optimality of minimizing the two-part message length from Bayes' Theorem:

$$\begin{aligned} P(h,e) &= P(h)P(e|h) \\ -\log P(h,e) &= -\log[P(h)P(e|h)] \\ -\log P(h,e) &= -\log P(h) - \log P(e|h) \\ I(h,e) &= I(h) + I(e|h) \end{aligned}$$

A further consequence is that an efficient code cannot encode the same hypothesis in two different lengths, since that would imply two distinct probabilities for the very same hypothesis.

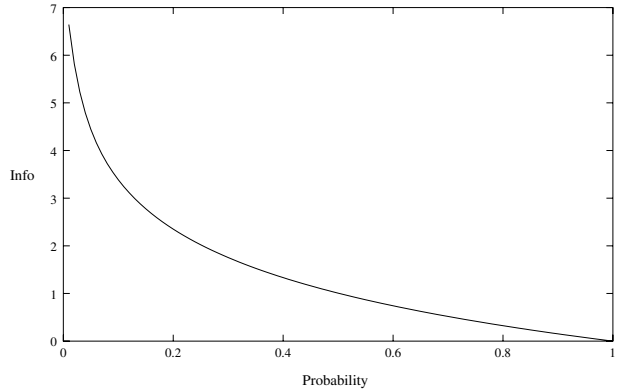


FIGURE 8.1
Shannon's information measure.

Minimum encoding inference metrics thus can provide an estimate of the joint probability $P(h,e)$. Since at least one plausible goal of causal discovery is to find that hypothesis which maximizes the conditional probability $P(h|e)$, such a metric suffices, since maximizing $P(h,e)$ is equivalent to maximizing $P(h|e)$. It is worth noting that in order to compute such a metric we need to compute how long the joint message of h together with $e|h$ *would be* were we to build it. It is not actually necessary to build the message itself, so long as we can determine how long it would be without building it.

© 2004 by Chapman & Hall/CRC Press LLC

8.3.1 Lam and Bacchus's MDL code for causal models

The differences between MDL and MML are largely ideological: MDL is offered specifically as a non-Bayesian inference method, which eschews the probabilistic interpretation of its code, whereas MML specifically is offered as a Bayesian technique. As MDL suffers by the lack of foundational support, its justification lies entirely in its ability to produce the goods, that is, in any empirical support its methods may find.

An MDL encoding of causal models was put forward by Lam and Bacchus in 1993 [165]. Their code length for the causal model (structure plus parameters) is:

$$I_{LB}(h) = \sum_{i=1}^{N} \left(k_i \log N + d(s_i - 1) \prod_{j \in \pi(i)} s_j \right) \tag{8.5}$$

where

- N is the number of nodes,
- k_i is the number of parents of the i-th node,
- d is the word size of the computer being used in bits,
- s_i is the number of states of the i-th node.

The explanation of this code length is that to communicate the causal model we must specify in the first instance each node's parents. Since there are N nodes this will take $\log N$ bits (using base 2 logs) for each such parent. For each distinct instantiation of the parents (there are $\prod_{j \in \pi(i)} s_j$ such instantiations, which is equal to $|\Phi_i|$ in Cooper and Herskovits' notation), we need to specify a probability distribution over the child node's states. This requires $s_i - 1$ parameters (since, by Total Probability, the last such value can be deduced). Hence, the CPT for each parent instantiation requires $d(s_i - 1)$ bits to specify.

Before going any further, it is worth noting that this falls well short of being an efficient code. It presumes that for each node both sides of the communication knows how many parents the node has. Further, in identifying those parents, it is entirely ignored that the model must be acyclic, so that, for example, the node cannot be a parent of itself. Again, once one parent has been identified, a second parent cannot be identical with it; the code length ignores that as well. Thus, the Lam and Bacchus code length computed for the causal structure is only a loose upper bound, rather than the basis for a determinate probability function.

The code length for parameters is also inefficient, and the problem there touches on another difference between MDL and MML. It is a principle of the MML procedure that the precision with which parameters are estimated be a function of the amount of information available within the data to estimate them. Where data relevant to a parameter are extensive, it will repay encoding the parameter to great precision, since then those data will themselves be encoded in the second part of the message more compactly. Where the data are weak, it rewards us to encode the parameter only vaguely, reserving our efforts for encoding the few data required for the second

© 2004 by Chapman & Hall/CRC Press LLC

part. The MDL code of Lam and Bacchus eschews such considerations, making an arbitrary decision on precision. This practice, and the shortcuts taken above on causal structure encoding, might be defended on the grounds of practicality: it is certainly easier to develop codes which are not precisely efficient, and it may well be that the empirical results do not suffer greatly in consequence.

Now let us consider the length of the second part of the message, encoding the data, is given as

$$I_{LB}(e|h) = K\left(\sum_{i=1}^{N} H(X_i) - \sum_{i=1}^{N} H(X_i, \pi(i))\right) \tag{8.6}$$

where

- K is the number of joint observations in the data,
- $H(X)$ is the entropy of variable X,
- $H(X_i, \pi(i))$ is the mutual information between X_i and its parent set.

Definition 8.2 Entropy

$$H(X) = -\sum_{x} P(X = x) \log P(X = x)$$

Definition 8.3 Mutual information

$$H(X_i, \pi(i)) = H(X_i) - H(X_i|\pi(i)) = \sum_{x_i, \phi(i)} P(x_i, \phi(i)) \log \frac{P(x_i, \phi(i))}{P(x_i)P(\phi(i))}$$

where $\phi(i)$ ranges over the distinct instantiations of the parent set $\pi(i)$. Intuitively, this mutual information reports the expected degree to which the joint probability of child and parent values diverges from what it would be were the child independent of its parents. I_{LB}, therefore, measures how much entropy in each variable is expected *not* to be accounted for by its parents and multiplies this by the number of data items that need to be reported. Clearly, this term by itself maximizes likelihood, which is as we would expect, being counterbalanced by the complexity penalizing term (8.5) for the final metric:

$$I_{LB}(h, e) = I_{LB}(h) + I_{LB}(e|h) \tag{8.7}$$

Lam and Bacchus need to apply this metric in a search algorithm, of course:

ALGORITHM 8.1
MDL causal discovery

1. *Initial constraints are taken from a domain expert. This includes a partial variable order and whatever direct connections might be known.*
2. *Greedy search is then applied: every possible arc addition is tested; the one with the best MDL measure is accepted. If none results in an improvement in the MDL measure, then the algorithm stops with the current model. Note that no arcs are deleted — the search is always from less to more complex networks.*

© 2004 by Chapman & Hall/CRC Press LLC

3. *Arcs are then checked individually for an improved MDL measure via arc reversal.*
4. *Loop at Step 2.*

The results achieved by this algorithm were similar to those of K2, but without requiring a full variable ordering. Results without significant initial information from humans (i.e., starting from an empty partial ordering and no arcs) were not good, however.

8.3.2 Suzuki's MDL code for causal discovery

Joe Suzuki, in an alternative MDL implementation [273], points out that Lam and Bacchus's parameter encoding is not a function of the size of the data set. This implies that the code length cannot be efficient. Suzuki proposes as an alternative MDL code for causal structures:

$$I_S(h) = \frac{\log K}{2} \sum_{i=1}^{N} \left((s_i - 1) \prod_{j \in \pi(i)} s_j \right) \tag{8.8}$$

which can be added to $I_{LB}(e|h)$ to get a total MDL score for the causal model. This gives a better account of the complexity of the parameter encoding, but entirely does away with the structural complexity of the Bayesian network! To be sure, structural complexity is reflected in the number of parameters. Complexity in Bayesian networks corresponds to the density of connectivity and the arity of parent sets, both of which increase the number of parameters required for CPTs. Nevertheless, that is by no means the whole story to network complexity, as there is complexity already in some topologies over others entirely independent of parameterization, as we shall see below in §8.5.1.1.

8.4 Metric pattern discovery

From Chapter 6 we know that dags within a single pattern are Markov equivalent. That is, two models sharing the same skeleton and v-structures can be parameterized to identical maximum likelihoods and so apparently cannot be distinguished given only observational data. This has suggested to many that the real object of metric-based causal discovery should be patterns rather than the dags being coded for by Lam and Bacchus. This is a possible interpretation of Suzuki's approach of ignoring the dag structure entirely. And some pursuing Bayesian metrics have also adopted this idea (e.g., [240, 46]). The result is a form of metric discovery that explicitly mimics the constraint-based approach, in that both aim strictly at causal patterns.

The most common approach to finding an **uninformed prior** probability over the model space — that is, a general-purpose probability reflecting no prior domain

© 2004 by Chapman & Hall/CRC Press LLC

knowledge — is to use a uniform distribution over the models. This is exactly what Cooper and Herskovits did, in fact, when assigning the prior $1/|\{h_i\}|$ to all dag structures. Pattern discovery assigns a uniform prior over the Markov equivalence classes of dags (patterns), rather than the dags themselves. The reason adopted is that if there is no empirical basis for distinguishing between the constituent dags, then empirical considerations begin with the patterns themselves. So, considerations *before* the empirical — prior considerations — should not bias matters; we should let the empirical data decide what pattern is best. And the only way to do that is to assign a uniform prior over patterns.

We believe that such reasoning is in error. The goal of causal discovery is to find the best causal model, not the best pattern. It is a general and well-accepted principle in Bayesian inference that our prior expectations about the world should not be limited by what our measuring apparatus is capable of recording (see, e.g., [173]). For example, if we thought that all four possible outcomes of a double toss of a coin were equally likely, we would hardly revise our opinion simply because we were told in advance that the results would be reported to us only as "Two heads" or "Not two heads." But something like that is being proposed here: since our measuring apparatus (observational data) cannot distinguish between dags within a pattern, we should consider only the patterns themselves.

The relevant point, however, is that the distinct dags within a pattern have quite distinct implications about causal structure. Thus, all of

- $A \leftarrow B \rightarrow C$
- $A \rightarrow B \rightarrow C$
- $A \leftarrow B \leftarrow C$

are in the same pattern, but they have entirely different implications about the effect of causal interventions. Even if we are restricted to observational data, we are not *in principle* restricted to observational data. The limitation, if it exists, is very like the restriction to a two-state description of the outcome of a double coin toss: it is one that can be removed. To put it another way: the pattern of three dags above can be realized in at least those three different ways; the alternative pattern of $A \rightarrow B \leftarrow C$ can be realized only by that single dag. Indeed, the number of dags making up a pattern over N variables ranges from a pattern with one dag (patterns with no arcs) to a pattern with $N!$ dags (fully connected patterns). If we have no prior reason to prefer one dag over another in representing causal structure, then *by logical necessity* we do have a prior reason to prefer patterns with more dags over those with fewer.

Those advocating pattern discovery need to explain why one dag is superior to another and so should receive a greater prior probability. We, in fact, shall find prior reasons to prefer some dags over others (§8.5.1.1), but not in a way supporting a uniform distribution over patterns.

© 2004 by Chapman & Hall/CRC Press LLC

8.5 CaMML: Causal discovery via MML

We shall now present our own method for automated causal discovery, implemented in the program CaMML, which was developed in large measure by Chris Wallace, the inventor of MML [290]. First we present the MML metric for linear models. Although we dealt with linear models in Chapter 6, motivating constraint-based learning with them, we find it easier to introduce the causal MML codes with linear models initially. In the next section we present a stochastic sampling approach to search for MML, in much more detail than any other search algorithm. We do so because the search is interestingly different from other causal search algorithms, but also because the search affords an important simplification to the MML causal structure code. Finally, only at the end of that section, we present the MML code for discrete models. Because of the nature of the search, that code can be presented as a simple modification of the Cooper and Herskovits probability distribution over discrete causal models.

Most of what was said in introducing MDL applies directly to MML. In particular, both methods perform model selection by trading off model complexity with fit to the data. The main conceptual difference is that MML is explicitly Bayesian and it takes the prior distribution implied by the MML code seriously, as a genuine probability distribution.

One implication of this is that if there is a given prior probability distribution for the problem at hand, then the correct MML procedure is to encode that prior probability distribution directly, for example, with a Huffman code (see [60, Chapter 5] for an introduction to coding theory). The standard practice to be seen in the literature is for an MML (or, MDL) paper to start off with a code that makes sense for communicating messages between two parties who are both quite ignorant of what messages are likely. The simplest messages (the simplest models) are given short codes and messages communicating more complex models are typically composites of them, and so are longer in an effectively computable way. That standard practice makes perfect sense (and, we shall follow it here) — *unless* a specific prior distribution is available, which implies something less than total prior ignorance about the problem. In any such case, considerations of efficiency, and the relation between code length and probability in (8.4), require the direct Huffman encoding (or similar). In short, MML is genuinely a Bayesian method of inference.

8.5.1 An MML code for causal structures

We shall begin by presenting an MML code for causal dag structures. We assume in developing this code no more than was assumed for the MDL code above (indeed, somewhat less). In particular, we assume that the number of variables is known. The MML metric for the causal structure h has to be an efficient code for a dag with N variables. h can be communicated by:

© 2004 by Chapman & Hall/CRC Press LLC

- First, specify a total ordering. This takes $\log N!$ bits.
- Next, specify all arcs between pairs of variables. If we assume an arc density = 0.5 (i.e., for a random pair of variables a 50:50 chance of an arc)[‡], we need one bit per pair, for the total number of bits:

$$\binom{N}{2} = \frac{N(N-1)}{2}$$

- However, this allows multiple ways to specify the dag if there is more than one total ordering consistent with it (which are known as **linear extensions** of the dag). Hence, this code is inefficient. Rather than correct the code, we can simply compensate by reducing the estimated code length by $\log M$, where M is the number of linear extensions of the dag. (Remember: we are not actually in the business of *communication*, but only of computing how long efficient communications would be!)

Hence,

$$I_{MML}(h) = logN! + \frac{N(N-1)}{2} - \log M \tag{8.9}$$

8.5.1.1 Totally ordered models (TOMs)

Before continuing with the MML code, we consider somewhat further the question of linear extensions. MML is, on principle, constrained by the Shannon efficiency requirement to count linear extensions and reduce the code length by the redundant amount. This is equivalent to increasing the prior probability for dags with a greater number of linear extensions over an otherwise similar dag with fewer linear extensions. Since these will often be dags within the same Markov equivalence class (pattern), this is a point of some interest. Consider the two dags of Figure 8.2: the chain has only one linear extension — the total ordering $< A, B, C >$ — while the common cause structure has two — namely, $< B, A, C >$ and $< B, C, A >$. And both dags are Markov equivalent.

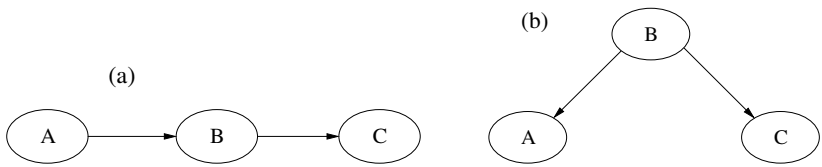


FIGURE 8.2
A Markov equivalent: (a) chain; (b) common cause.

[‡] We shall remove this assumption below in §8.6.3.

© 2004 by Chapman & Hall/CRC Press LLC

The most common approach by those pursuing Bayesian metrics for causal discovery thus far has been to assume that dags within Markov equivalence classes are inherently indistinguishable, and to apply a uniform prior probability over all of them; see, for example, Madigan et al. [177]. Again, according to Heckerman and Geiger [103], equal scores for Markov equivalent causal structures will often be appropriate, even though the different dag structures *can* be distinguished under the causal interpretation.

Note that we are not here referring to a uniform prior over patterns, which we considered in §8.4, but a uniform prior *within* patterns. Nonetheless, the considerations turn out to be analogous. The kind of indistinguishability that has been justified for causal structures within a single Markov equivalence class is *observational* indistinguishability. The tendency to interpret this as *in-principle* indistinguishability is not justified. After all, the distinguishability under the causal interpretation is clear: a causal intervention on A in Figure 8.2 will influence C in the chain but not in the common causal structure. Even if we are limited to observational data, the differences between the chain and the common cause structure will become manifest if we expand the scope of our observations. For example, if we include a new parent of A, a new v-structure will be introduced only if A is participating in the common causal structure, resulting in the augmented dags falling into distinct Markov equivalence classes.

We can develop our reasoning to treat linear extensions. A **totally ordered model**, or **TOM**, is a dag together with one of its linear extensions. It is a plausible view of causal structures that they are, at bottom, TOMs. The dags represent causal processes (chains) linking together events which take place, in any given instance, at particular times, or during particular time intervals. All of the events are ordered by time. When we adopt a dag, without a total ordering, to represent a causal process, we are representing our ignorance about the underlying causal story by allowing multiple, consistent TOMs to be entertained. Our ignorance is not in-principle ignorance: as our causal understanding grows, new variables will be identified and placed within it. It is entirely possible that in the end we shall be left with only one possible linear extension for our original problem. Hence, the more TOMs (linear extensions) that are compatible with the original dag we consider, the more possible ways there are for the dag to be realized and, thus, the greater the prior probability of its being true.

In short, not only is it correct MML coding practice to adjust for the number of linear extensions in estimating a causal model's code length, it is also the correct Bayesian interpretation of causal inference.

This subsection might well appear to be an unimportant aside to the reader, especially in view of this fact mentioned previously in §8.2.1: counting linear extensions is exponential in practice [32]. In consequence, the MML code presented so far does not directly translate into a tractable algorithm for scoring causal models. Indeed, its direct implementation in a greedy search was never applied to problems with more than ten variables for that reason [293]. However, TOMs figure directly in the sampling solution of the search and MML scoring problem, in section §8.6 below.

© 2004 by Chapman & Hall/CRC Press LLC

8.5.2 An MML metric for linear models

So, we return to developing the MML metric. We have seen the MML metric for causal structure; now we need to extend it to parameters given structure and to data given both parameters and structure. We do this now for linear models; we develop the MML metric for discrete models only after describing CaMML's stochastic search in §8.6, since the discrete metric takes advantage of an aspect of the search.

The parameter code. Following Wallace and Freeman [292], the message length for encoding the linear parameters is

$$I_{MML}(\theta|h) = \sum_{X_j} -\log \frac{f(\theta_j|h)}{\sqrt{F(\theta_j)}} \tag{8.10}$$

$f(\theta_j|h)$ is the prior density function over the parameters for X_j given h. $F(\theta_j)$ is the Fisher information for the parameters to X_j. It is the Fisher information which controls the precision of the parameter estimation. In effect, it is being used to discretize the parameter space for θ_j, with cells being smaller (parameters being more precisely estimated) when the information afforded by the data is larger. (For details see [292].)

Wallace et al. [293] describe an approximation to (8.10), using the standard assumptions of parameter independence and the prior density $f(\theta_j|h)$ being the normal $N(0, \sigma_j)$, with the prior for σ_j being proportional to $1/\sigma_j$.

The data code. The code for the sample values at variable X_j given both the parameters and dag is, for i.i.d. samples,

$$I_{MML}(x_j|h, \theta_j) = -\log P(x_j|h, \theta_j) = -\log \prod_{k=1}^{K} \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\delta_{jk}^2/2\sigma_j^2} \tag{8.11}$$

where K is the number of joint observations in the sample and δ_{jk} is the difference between the observed value of X_j and its linear prediction. Note that this is just a coding version of the Normal density function, which is the standard modeling assumption for representing unpredicted variation in a linear dependent variable.

The MML linear causal model code. Combining all of the pieces, we have the total MML score for a linear causal model being:

$$I_{MML}(h, e) = I_{MML}(h) + I_{MML}(\theta|h) + \sum_{j=1}^{N} I_{MML}(x_j|h, \theta_j) \tag{8.12}$$

© 2004 by Chapman & Hall/CRC Press LLC

8.6 CaMML stochastic search

Having the MML metric for linear causal models in hand, we can reconsider the question of how to search the space of causal models. Rather than apply a greedy search, or some variant such as beam search, we have implemented two stochastic searches: a genetic algorithm search [201] and a Metropolis sampling process [294].

8.6.1 Genetic algorithm (GA) search

The CaMML genetic algorithm searches the dag space; that is, the "chromosomes" in the population are the causal model dags themselves. The algorithm simulates some of the aspects of evolution, tending to produce better dags over time — i.e., those with a lower MML score. The best dags, as reported by the MML metric, are allowed to reproduce. Offspring for the next generation are produced by swapping a subgraph from one parent into the graph of the other parent and then applying mutations in the form of arc deletions, additions and reversals. A minimal repair is done to correct any introduced cycles or reconnect any dangling arcs. Genetic algorithms frequently suffer from getting trapped in local minima, and ours was no exception. In order to avoid this, we introduced a temperature parameter (as in simulated annealing) to encourage a more wide-ranging search early in the process. None of this addresses the problem of counting linear extensions for the MML metric itself. We handled that by an approximative estimation technique based upon that of Karzanov and Khachiyan [141]. For more details about our genetic algorithm see [201]. Although the GA search performs well, here we shall concentrate upon the Metropolis sampling approach, as it offers a more telling solution to the problem of counting linear extensions.

8.6.2 Metropolis search

Metropolis sampling allows us to approximate the posterior probability distribution over the space of TOMs by a sampling process. From that process we can estimate probabilities for dags. The process works by the probability measure providing a kind of pressure on the sampling process, ensuring that over the long run models are visited with a frequency approximating their probabilities [189, 176]. This is a type of **Markov Chain Monte Carlo** (MCMC) search, meaning that the probabilistically selected sequence of sampled models forms a Markov Chain (the probability of reaching any particular model depends only upon what model is currently being sampled and not on the sampling history). The probabilities for our CaMML implementation are, of course, derived from the MML metric. Since the models being sampled are TOMs, rather than dags, the MML metric is computed for TOMs only at each step. Hence there is no (direct) problem with counting linear extensions.

Since we are searching in the space of TOMs, we need an adjustment of $I_{MML}(h)$

© 2004 by Chapman & Hall/CRC Press LLC

(8.9) to reflect the change. This is, very conveniently, simply to drop the term requiring us to count linear extensions. Hence,

$$I_{MML}(h) = logN! + \frac{N(N-1)}{2} \tag{8.13}$$

with h now ranging over TOMs rather than dags.

ALGORITHM 8.2
CaMML Metropolis Algorithm

1. *First, an initial TOM is selected at random. Call this M.*
2. *Uniformly randomly choose between the following possible changes to the current model M producing M':*
   (a) **Temporal order change:** *Swap the order of two neighboring nodes. (If there is an arc between them, it reverses direction.)*
   (b) **Skeletal change:** *Add (or delete) an arc between two randomly selected nodes.*
   (c) **Double skeletal change:** *Randomly choose three nodes. Add (or delete) arcs between the first two nodes and the last (in the temporal order). (Double changes are not necessary for the Metropolis algorithm to work, but are included to help accelerate the sampling process.)*
3. *Accept M' as the new M if $\frac{P_{MML}(M')}{P_{MML}(M)} > U[0,1]$; otherwise retain M.*

   where

   $$P_{MML}(x) = e^{-I_{MML}(x)}$$

   and $U[0,1]$ is a uniform random variate in the interval $[0,1]$.
4. *Count the current visit to M.*
5. *Loop at 2 (until an input number of samples has been completed).*

This Monte Carlo process meets the sufficient conditions for the Metropolis algorithm to apply, viz., the number of possible transitions from each TOM is constant, and every transition is reversible [189]. The process will therefore visit every TOM with a frequency proportional to its joint probability with the data, and hence proportional to its MML posterior.

We are not directly interested in the posterior probabilities of individual TOMs, but instead in the probabilities of sets of TOMs which are sufficiently similar to be grouped together and represented by a single **MML model**. (We expand on this grouping process in §8.6.4.) The total posterior given by MML to the representative is the sum of the posteriors of the member TOMs. We therefore do not count visits to individual TOMs, but visits to MML models.

Whenever the MML model is changed by a step, the skeleton and maximized likelihood is found. Recall from Chapter 6 that all Markov equivalent models share

© 2004 by Chapman & Hall/CRC Press LLC

these properties. We then attempt to count visits to (skeleton-likelihood) pairs. In a problem with many variables, there will be far too many such pairs to allow for exact counting. So, we form a hash index from the (skeleton-likelihood) pair into a 65536-entry table, using different random keys for each possible arc. Distinct MML models are therefore unlikely to yield the same hash indices, which would throw off the estimated posterior probability. To be even safer, we actually maintain two distinct hash tables with distinct random keys and use the lesser count at the end of the sampling process to estimate posterior probabilities. That estimate is simply the final count, divided by the number of sampling steps.

Note that the process of finding a representative MML model is used only to update the hash table count and does not influence the sampling process by changing the current TOM from which it is derived.

During sampling, the program accumulates a list of up to 50 MML models, being those with the highest estimated posterior. Each retained MML model is represented by its highest-posterior dag. The dag posterior is estimated using a further two tables for which the hash indices are calculated from "dag signatures." These signatures are formed like the MML model signatures, but use a different matrix of pseudo-random constants and do not use likelihoods. This matrix is not symmetrical, allowing the dag signature to capture arc directions as well as skeletons. The dag counts are, of course, updated after each sampling step.

At the end of the sampling, the selected models are displayed. For each retained MML model CaMML displays: its posterior, the posterior of the highest-posterior dag, the arcs and estimated parameters. It also reports the weighted frequency of directed arcs across the retained models. For more details of the sampling process see [294].

8.6.3 Prior constraints

TETRAD II allows prior variable orderings to be specified, by grouping variables in *tiers*. Variables in earlier tiers are constrained during search not to be effects of variables in later tiers. Variables within a tier can be discovered to exist in causal chains in any (non-cyclic) order. Of course, by having as many tiers as variables, one can impose a total ordering upon the variables.

An interesting alternative method of specifying prior probability is that adopted by Heckerman and Geiger [103] for their Bayesian BDe and BGe metrics. They have the user generate a "best guess" causal model in the beginning. Their prior probability metric then rewards models which are closer to this best guess model (in edit distance terms of arc deletions, additions and reversals).

CaMML has a more flexible system of prior constraints. For each pair of variables the user can specify a prior probability for the existence of a particular directed arc between them. Any arc probabilities left unspecified are assigned the default $p = 0.5$. In the structure code, instead of coding each possible arc, or absence of an arc, as one bit, as in Equations (8.9) and (8.13), those equations are modified to reflect the prior probability p_i for each possible directed arc i. In particular, for each arc in a model CaMML adds $-\log p$ bits and for each missing possible arc it adds

© 2004 by Chapman & Hall/CRC Press LLC

$-\log(1-p)$. Thus we get the following MML structural metric:

$$I_{MML}(h) = logN! - \sum_i \log p_i - \sum_j \log(1-p_j) \tag{8.14}$$

where i indexes the arcs in a TOM and j indexes the possible arcs that are absent in the TOM.

By assigning a zero prior probability to sets of directed arcs, one can force all causal relations between two non-overlapping sets of variables to hold in a fixed direction — in other words, tiers of variables can be identified, as with TETRAD II. Similarly, CaMML's soft prior constraints on arcs can be made to mimic the edit distance metric of Heckerman and Geiger. Much prior expert "knowledge" is not knowledge at all, but opinion. This relevant human expertise in general is valuable, opinion or not. Hence, the soft constraint of selecting prior probabilities for directed arcs near, but not identical to, 0 or 1 is valuable, and it provides options which neither the Heckerman and Geiger nor TETRAD regimes allow.

8.6.4 MML models

We now describe the grouping of TOMs into the **MML model** for which visit counts are kept. First, TOMs which differ only in the order of the variables, but with the same skeleton and arc directions, are clearly indistinguishable given the data, and so will be grouped. All TOMs in the group are linear extensions of the same dag.

Suppose two TOMs A and B differ only in that B contains an arc not present in A. If the magnitude of this causal effect is sufficiently small, the MML code length may be reduced by grouping A and B together, and using the simpler A as the representative model. This can shorten the total message, because the prior probability of the group is the sum of the priors of A and B. The data may be encoded in a longer message with A rather than with the **small effect** model B, but if the coefficient of the extra arc is small, that increase in the length of the data part of the message may well be less than the reduction in first part encoding the model.

CaMML performs an approximate check whether neighboring TOMs should be grouped together on this basis. The TOM within such a group without small effects is chosen as its representative model and used to compute parameters and data length.

It may be thought that deleting small effect arcs is an unimportant refinement of the MML process. Others using stochastic sampling for causal discovery have not employed it (e.g., [177]). However, typically for a TOM A there will be about $\frac{N^2}{4}$ other TOMs differing from it only by the addition of a small effect. The total joint probability (and hence posterior probability) of all such TOMs may therefore be about $\frac{N^2}{4\sqrt{K}}$ times that of A, where K is the data set size. Unless K is greater than N^4, there may be more total posterior probability over these small effect TOMs than on A itself; so inclusion of them in the set represented by A adds significantly to the posterior probability of the set. Indeed, in a large model space the search may never actually sample the representative model. Of course, in the limit it must do so, but lacking infinite patience, we prefer to group models.

© 2004 by Chapman & Hall/CRC Press LLC

8.6.5 An MML metric for discrete models

We can readily extend this work from linear to discrete causal model discovery. The MML structural score for TOMs is unaffected, as is the Metropolis search process. All we need is a new metric for parameters and data. The simplest approach is to apply an adjustment to the probability distribution of Cooper and Herkovits $P_{CH}(h, e)$. Bearing in mind that the most significant distinction between their approach and the MML approach is their assumption of a uniform prior over hypotheses (their assumption 5, implicitly denying the relevance of multiple linear extensions), the search being conducted over the TOM space eliminates that distinction. Thus, all that is required is a simpler adjustment, reflecting the difference between integrating the parameters out of the computation (as in §8.1) and the MML method, which *estimates* the parameters by partitioning the parameter space and selecting the optimal cell for communicating that estimate. The difference amounts to a penalty per parameter of $\log(e^{3/2}\frac{\pi}{6})$ bits (i.e., $\frac{1}{2}\log\frac{\pi e}{6}$ **nits** — meaning units to the base of the natural log — see [292]). Hence:

$$I_{MML}(h, e) = \sum_{j=1}^{N} s_j \left(\frac{3}{2}\log e + \log\frac{\pi}{6}\right) - \log P_{CH}(h, e) \qquad (8.15)$$

where h ranges over TOMs and s_j is the number of parameters needed for X_j.

8.7 Experimental evaluation

Having seen a number of algorithms for learning causal structure developed, the natural question to ask is: Which one is best? Unfortunately, there is no agreement about exactly what the question means, let alone which algorithm answers to it. There has been no adequate experimental survey of the main contending algorithms. Some kind of story could be pieced together by an extensive review of the literature, since nearly every publication in the field attempts to make some kind of empirical case for the particular algorithm being described in that publication. However, the evaluative standards applied are uneven, to say the least.

Note that the evaluative question here is not identical to the evaluative question being asked by the algorithms themselves. That is, the causal discovery algorithms are aiming to find the best explanatory *model* for whatever data they are given, presumably for use in future predictive and modeling tasks. For this reason they may well apply some kind of complexity penalty to avoid overfitting the training data. Here, however, we are interested in the question of the best model-discoverer, rather than the best model. We are not necessarily interested in the complexities of the model representations being found, because we aren't necessarily interested in the models as such. Analogically, we are here interested in the intelligence of a learner, rather than in the "smartness" (or, elegance) of its solutions.

© 2004 by Chapman & Hall/CRC Press LLC

8.7.1 Qualitative evaluation

The TETRAD II publications (e.g., [265, 244]) have used stochastic sampling from known causal models to generate artificial data and then reported percentages of errors of four types:

- Arc omission, when the learned model fails to have an arc in the true model
- Arc commission, when the learned model has an arc not in the true model
- Direction omission, when the learned model has not directed an arc required by the pattern of the true model
- Direction commission, when the learned model orients an arc incorrectly according to the pattern of the true model

This is an attempt to quantify qualitative errors in causal discovery. It is, however, quite crude. For example, some arcs will be far more important determiners of the values of variables of interest than others, but these metrics assume all arcs, and all arc directions within a pattern, are of equal importance.

Regardless, this kind of metric is by far the most common in the published literature. Indeed, the most common evaluative report consists of using the ALARM network (Figure 5.2) to generate an artificial sample, applying the causal discovery algorithm of interest, and counting the number of errors of omission and commission. Every algorithm reported in this chapter is capable of recovering the ALARM network to within a few arcs and arc directions, so this "test" is of little interest in differentiating between them. Cooper and Herskovits's K2, for example, recovered the network with one arc missing and one spurious arc added, from a sample size of 10,000 [54]. TETRAD II also recovers the ALARM network to within a few arcs [265], although this is more impressive than the K2 result, since it needed no prior temporal ordering of the variables. Again, Suzuki's MDL algorithm recovered the original network to within 6 arcs on a sample size of only 1000 [273].

Perhaps slightly more interesting is our own empirical study comparing TETRAD II and CaMML on linear models, systematically varying arc strengths and sample sizes [65]. The result was a nearly uniform superiority in CaMML's ability to recover the original network to within its pattern faster than (i.e., on smaller samples than) TETRAD II.

8.7.2 Quantitative evaluation

Because of the maximum-likelihood equivalence of dags within a single pattern, it is clear that two algorithms selecting identical models, or Markov equivalent models, will be scored alike on ordinary evaluative metrics. But it should be equally clear that non-equivalent models may well deserve equal scores as well, which the qualitative scores above do not reflect. Thus, if a link reflects a nearly vanishing magnitude of causal impact, a model containing it and another lacking it but otherwise the same may properly receive (nearly) the same score. Again, the parameters for an association between parents may lead to a simpler v-structure representing the very same probability distribution as a fully connected network of three variables (see [293] for

© 2004 by Chapman & Hall/CRC Press LLC

an example). So, we clearly want a more discriminating and accurate metric than the metrics of omission/commission.

The traditional such metric is **predictive accuracy**: the percentage of correctly predicted values of some target variable. In classification problems this has a clear meaning, but for learning causal models it is generally less than clear which variable(s) might be "targeted" for classification. Predictive accuracy, in any case, suffers from other problems, which we examine in §10.5.1.

Kullback-Leibler divergence (§3.6.5) of the learned model from the model generating the data is preferable and has been used by some. Suzuki, for example, reported the KL metric over the ALARM network, showing a better result for his MDL algorithm than that for K2 [273]. KL divergence is normally measured over all the variables in a model. Because of the Markov equivalence of all models within a pattern (Theorem 6.1), this implies that what is being evaluated is how close the learned model is to the original *pattern*, rather than to the original causal model. So this is not obviously the best method for evaluating *causal* discovery.

In one study [201], we compared linear CaMML with Heckerman and Geiger's BGe metric [103], using the same search algorithm for both. We also used two distinct kinds of prior probabilities for both: one applying CaMML's uniform prior over TOMs (P1) and the other applying a uniform prior over dags within a pattern (P2), which is more consistent with Heckerman and Geiger's views. KL divergence from the original model was measured in two ways. When KL was measured over all variables in the network, metrics using P2 tended to do best. When KL was measured over the original leaves only (nodes without children), metrics using P1 did best. The latter measure arguably better reflects the prediction task for causal discovery; for one thing, if causal structure is misidentified, the leaves are likely to be misidentified, leading to a worse score on this metric. In this particular study, the only interesting differences in performance arose from the priors; that is, BGe and CaMML otherwise performed alike. We discuss some issues of evaluation further in Chapter 10.

8.8 Summary

There are two distinct approaches to learning causal structure from data. Constraint-based learning attempts to identify conditional independencies in isolation and to construct causal models, or patterns, from them. Given perfect access to the conditional independencies, and Reichenbach's Principle of the Common Cause, we can reasonably infer that the true causal model lies in one and only one pattern. However, it is generally not optimal to judge conditional independencies in isolation: the presence (or absence) of one dependency frequently will confer support or undermine the presence of another. Metric learners can take advantage of such evidential relevance, since they score causal models (or patterns) as a whole. Constraint-based learners are

© 2004 by Chapman & Hall/CRC Press LLC

the more popular, because they are simple to understand and implement, and they can now be found in the leading Bayesian network tools, as well as in TETRAD. The different metric learners appear to have more promise for the long haul, however. Such experimental literature as exists favors them, although the methodology of evaluation for Bayesian network learners remains unclear. We expand on the question of evaluation, and especially the evaluation of the Bayesian networks themselves, in Chapter 10.

8.9 Bibliographic notes

In addition to what we have presented here, there are a host of other learning techniques that attempt to ply the tradeoff between model simplicity and data fit by applying "penalties" to some measure of model complexity, including BIC (Bayesian information criterion) [245] and AIC (Akaike information criterion) [242].

A good early review of the causal discovery literature is Buntine's guide [36]. Jordan's anthology contains a number of useful articles on various aspects of causal discovery [135]. Neapolitan's *Learning Bayesian Networks* [200] treats some aspects of causal discovery we do not have the time for here.

8.10 Technical notes

χ^2 test.
The χ^2 test is a standard significance test in statistics for deciding whether an observed frequency fails to match an expected frequency. It was proposed in the text as a substitute in the PC algorithm for the partial correlation significance test when applying the algorithm to discrete variables. In particular, it can be used to test whether $P(Y = y|X = x, Z = z) = P(Y = y|Z = z)$ across the different possible instantiations of the three variables (or, sets of variables), in order to decide whether $X \perp\!\!\!\perp Y|Z$.

The assumption that $P(Y = y|X = x, Z = z) = P(Y = y|Z = z)$ can be represented by taking the expected frequencies in the CPT cells (treating X and Z as parents of Y) to be $P(Y = y|X = x_1, Z = z) = \ldots = P(Y = y|X = x_j, Z = z) = f(Y = y|Z = z)$, where this last is the frequency with which $Y = y$ given that $Z = z$. The observed frequencies are then just $f(Y = y|X = x_1, Z = z), \ldots, f(Y = y|X = x_j, Z = z)$. The statistic for running a significance test on the discrepancy between these two measures is:

$$\chi^2 = \sum_{ijk} \frac{n_{jk}(f(y_i|x_j, z_k) - f(y_i|z_k))^2}{f(y_i|z_k)}$$

© 2004 by Chapman & Hall/CRC Press LLC

where i, j, k index the possible instantiations $\Omega_Y, \Omega_X, \Omega_Z$ of Y, X, Z respectively and there are n_{jk} samples where X takes value j and Z takes value k.

For conducting such a significance test, it is generally recommended that $n_{jk} f(y_i|z_k) \geq 5$ for each possible combination. The degrees of freedom for the test are $\nu = |\Omega_{YXZ}| - |\Omega_{YZ}|$.

8.11 Problems

Programming Problems

Problem 1

Implement the Bayesian metric $P_{CH}(h_i, e)$ of (8.3) for discrete causal models.

Problem 2

Implement one of the alternative metrics for discrete causal models, namely MDL, or MML (or BDe, after reading [103]).

Problem 3

Implement a simple greedy search through the dag or TOM space. Test the result on artificial data using one or more metrics from prior problems.

Problem 4

Implement the PC algorithm using the χ^2 from §8.10. Compare the results experimentally with one of the metric causal discovery programs from prior problems.

Evaluation Problem

For this problem you should get and install one or more of the causal discovery programs: TETRAD IV, WinMine or CaMML. For instructions see Appendix B.

Problem 5

Run the causal discovery program on some of the data sets at our book web site. Try the program with and without giving it prior information (such as variable order). Evaluate how well it has done in one of two ways:

1. In terms of either predictive accuracy, KL distance or information reward (see Chapter 10).
2. By qualitative or quantitative comparison with an alternative causal discovery program (perhaps from one of the problems above, or also downloaded by you).

© 2004 by Chapman & Hall/CRC Press LLC

Part III

KNOWLEDGE ENGINEERING

© 2004 by Chapman & Hall/CRC Press LLC

By now we have seen what Bayesian networks are, how they can represent uncertain processes of a considerable variety, how they can be conditioned to reflect new information, make decisions, perform causal modeling and optimize planning under uncertainty. We have seen how causal structures can be parameterized from data once learned or elicited. And we have also seen how such structures can be learned in the first place from observational data, either by taking advantage of conditional independencies in the data or by using Bayesian metrics. What is largely missing from the story so far is a *method* for putting all of these ingredients together in a systematic way.

In Chapter 9 we apply some of the more useful ideas of software engineering, and recent experiences of ourselves and others working with Bayesian networks, toward the development of such a method, which we call **KEBN**: Knowledge Engineering with Bayesian Networks. Part of the method necessarily includes techniques for evaluating Bayesian networks once developed or learned. In Chapter 10 we first present evaluation methods employing expert judgment, including sensitivity analysis, which tests the sensitivity of networks to parameters. Then we discuss some of the more prominent traditional and new approaches to evaluating Bayesian networks using sample data.

In Chapter 11 we return to the presentation of Bayesian network applications, but this time with the idea of illustrating some of the KEBN processes and other issues raised in Part III.

© 2004 by Chapman & Hall/CRC Press LLC

9

Knowledge Engineering with Bayesian Networks

9.1 Introduction

Within the Bayesian network research community, the initial work in the 1980's and early 1990's focused on inference algorithms to make the technology computationally feasible. As it became clear that the "knowledge bottleneck" of the early expert systems was back — meaning the difficulties of finding human domain experts, extracting their knowledge and putting it into production systems — the research emphasis shifted to automated learning methods. That is necessary and inevitable. But what practitioners require then is a overarching methodology which combines these diverse techniques into a single "knowledge engineering" process, allowing for the construction Bayesian models under a variety of circumstances, which we call **Knowledge Engineering with Bayesian Networks**, or **KEBN**.

In this chapter we tie together the various techniques and algorithms we have previously introduced for building BNs and supplement them with additional methods, largely drawn from the software engineering discipline, in order to propose a general methodology for the development and deployment of BNs. We supplement this in the next chapter with methods for evaluating Bayesian networks, giving an outline of a comprehensive methodology for modeling with Bayesian networks. No one has fully tested such a methodology, so our KEBN model must remain somewhat speculative. In any case, we can illustrate some of its major features with a number of case studies from our experience, which we proceed to do in Chapter 11.

9.1.1 Bayesian network modeling tasks

When constructing a Bayesian network, the major modeling issues that arise are:

1. What are the variables? What are their values/states?
2. What is the graph structure?
3. What are the parameters (probabilities)?

When building decision nets, the additional questions are:

4. What are the available actions/decisions, and what impact do they have?
5. What are the utility nodes and their dependencies?
6. What are the preferences (utilities)?

© 2004 by Chapman & Hall/CRC Press LLC

Expert elicitation is a major method for all of these tasks. Methods involving automated learning from data, and adapting from data, can be used for tasks 1-3 (if suitable data are available). We have described the main techniques for tasks 2 and 3 in Chapters 6, 7 and 8. Task 1 has been automated as well, although we do not go into these methods in this text. Identifying variables is known in the machine learning literature as unsupervised classification; see, for example, Chris Wallace's work on Snob [290, 291]. There are many techniques for automated discretization, for example [193]. On the other hand, very little has been done to automate methods for tasks associated with building decision networks and we do not cover them in this text. In the remainder of this chapter we shall first focus on how to perform all the tasks using expert elicitation, then we shall consider methods for **adaptation**, combining elicitation with machine learning, in §9.4.

9.2 The KEBN process

9.2.1 KEBN lifecycle model

A simple view of the software engineering process construes it as having a lifecycle: the software is born (design), matures (coding), has a lengthy middle age (maintenance) and dies of old age (obsolescence). Our best effort at construing KEBN in such a **lifecycle model** (also called a "waterfall" model) is shown in Figure 9.1. Although we prefer a different view of KEBN (presented just below), the lifecycle is a convenient way of introducing many aspects of the problem, partly because it is so widely known and understood.

Building the Bayesian network is where the vast majority of research effort in KEBN has gone to date. In the construction phase, the major network components of structure, parameters and, if a decision network, utilities (preferences) must be determined through elicitation from experts, or learned with data mining methods, or some combination of the two.

Evaluation aims to establish that the network is right for the job, answering such questions as: Is the predictive accuracy for a query node satisfactory? Does it respect any known temporal order of the variables? Does it incorporate known causal structure? **Sensitivity analysis** looks at how sensitive the network is to changes in input and parameter values, which can be useful both for validating that the network is correct and for understanding how best to use the network in the field.

Field testing first puts the BN into actual use, allowing its usability and performance to be gauged. **Alpha testing** refers to an intermediate test of the system by inhouse people who were not directly involved in developing it; for example, by other inhouse BN experts. **Beta testing** is testing in an actual application by a "friendly" end-user, who is prepared to accept hitting bugs in early release software. For software that is not being widely marketed, such as most BNs, this idea may be inapplicable — although domain experts may take on this role. Acceptance testing is surely required:

© 2004 by Chapman & Hall/CRC Press LLC

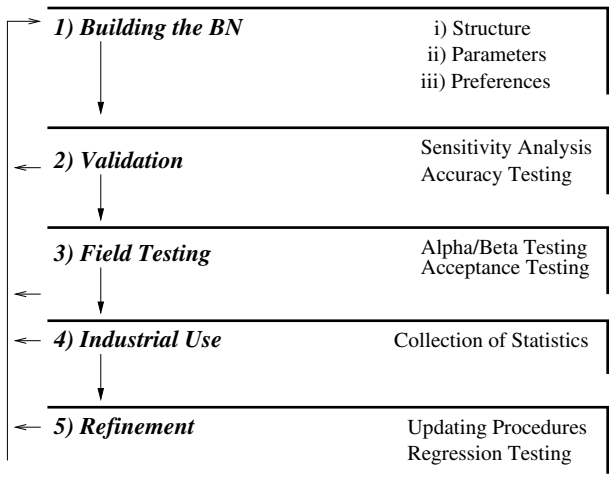


FIGURE 9.1
A KEBN lifecycle model.

it means getting the end users to accept that the BN software meets their criteria for use.
Industrial use sees the BN in regular use in the field and requires that procedures be put in place for this continued use. This may require the establishment of a new regime for collecting statistics on the performance of the BN and statistics monitoring the application domain, in order to further validate and refine the network.
Refinement requires some kind of change management regime to deal with requests for enhancement or fixing bugs. **Regression testing** verifies that any changes do not cause a degradation (regression) in prior performance.

In this chapter we will describe detailed procedures for implementing many of these steps for Bayesian network modeling. Those which we do not address specifically, such as how to do regression testing and acceptance testing, do not seem to have features specific to Bayesian network modeling which are not already addressed here. We refer you to other works on software engineering which treat those matters in the notes at the end of the chapter (§9.6).

9.2.2 Prototyping and spiral KEBN

We prefer the idea of prototyping for the KEBN process to the lifecycle model. Prototyping interprets the analogy of life somewhat differently: as an "organism," the software should grow by stages from childhood to adulthood, but at any given stage it is a self-sufficient, if limited, organism. **Prototypes** are functional implementations of software: they accept real input, such as the final system can be expected to deal with, and produce output of the type end-users will expect to find in the final system. What distinguishes early form prototypes from the final software is that the func-

© 2004 by Chapman & Hall/CRC Press LLC

tions they implement are limited, so that they are fairly easily developed, whereas later ones approximate the full functionality envisioned. Since each prototype in the entire sequence supports a limited form of the targeted functionality, end-users can experiment with them in just the way they are intended to use the final product, and so they can provide feedback and advice from the early stages of development. Much of the testing, then, is done in a setting as close to the target usage environment as possible.

The **initial prototypes** should be used for planning the KEBN process. The subproblem addressed should be self-contained but reasonably representative of the global problem. It should be scoped to minimize development risk, with the prototype employing available capabilities as much as possible and using simplified variables and structure. As a result you avoid one of the main risks in the BN development process, overselling the capabilities of the final system [166]. Since initial prototypes are both functional and representative, they provide a working product for provisional assessment and planning.

The **incremental prototypes** require relatively simple extensions to the preceding prototype. They should attack a high priority, but small, subset of the remaining difficult issues. The size of the subset of issues tackled, and their difficulty, is used to control the continuing development risk. The incremental development refines both the domain expert's and the knowledge engineer's understanding of the requirements and approach.

Why prototype? We believe that it just is the best software development process overall, agreeing with Fred Brooks [33] and Barry Boehm [22]. Prototyping allows the organic growth of software, tracking the problem specifications and growing in manageable spurts. The use of prototypes attacks the trade-off between comprehensiveness and intelligibility from the right starting point, namely the small end. Those are general attributes favoring prototyping for software engineering of all kinds. However, because Bayesian network development usually involves building graphical models using visual aids from early on, it is even easier than normal to provide end-users with the kind of graphical user interface the end product is likely to have, easing the prototyping approach.

In order to highlight the differences between prototyping and the usual approach to software development, as typified in lifecycle models, Barry Boehm introduced the **Spiral Model** of software development, which we illustrate in Figure 9.2. The construction of a sequence of prototypes can be viewed as a repeating cycle of analyzing requirements, design, implementation, operation and evaluation. In evaluation the behavior of each prototype model on sample problems is explored, in conjunction with end-users, and the next stage is planned as problems in the current prototype are uncovered.

9.2.3 Are BNs suitable for the domain problem?

As we have already seen, Bayesian networks can handle a wide variety of domains and types of application. They explicitly model uncertainty, allow for the representation of complex interactions between variables, and they can be extended with

© 2004 by Chapman & Hall/CRC Press LLC

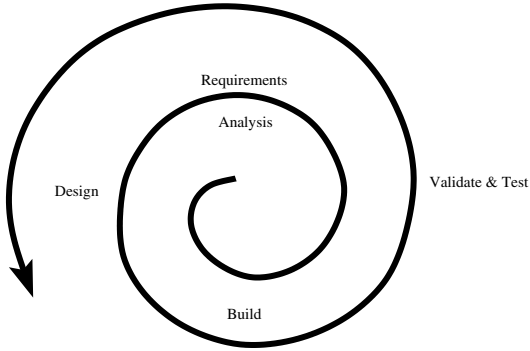


FIGURE 9.2
A spiral model for KEBN.

utilities and decision nodes for planning and decision making. Furthermore, the representations developed are not just black boxes, but have a clear semantics, available for inspection. Nevertheless, BNs are not suitable to any and every application, so before launching any large-scale KEBN process, it is important to make sure that BN technology is suitable for the particular problem. It is easiest to list features that would suggest BNs are *not* appropriate.

- If the problem is a "one-off," for which there is no data available and any model built won't be used again, then the overhead of the KE process may not be worth it. Bayesian networks might still be used in a one-off modeling process, of course, without going through all of the KEBN knowledge engineering overhead.
- There are no domain experts, nor useful data.
- If the problem is very complex or not obviously decomposable, it may not be worth attempting to analyze into a Bayesian network.
- If the problem is essentially one of learning a function from available data, and a "black box" model is all that is required, an artificial neural network or other standard machine learning technique may be applied.

9.2.4 Process management

It is important that the knowledge engineering process be properly managed. Foremost is the management of human relations. It is unrealistic to expect that putting the domain expert and the knowledge engineer in a room together will result in the smooth production of a BN model for the domain. It is far more likely that they will talk past each other! The knowledge engineer must learn about the problem domain and the domain expert must learn what BN models are and what they can do. This learning aspect of the process can be time-consuming and frustrating, but both sides must expect and tolerate it.

© 2004 by Chapman & Hall/CRC Press LLC

One productive way of training the participants is to start by constructing very simple models of simplified, throw-away domain problems. These are "pre-prototypes:" whereas prototypes are intended to feed into the next stage, at least conceptually, the throw-away models are intended to build mutual understanding. As knowledge engineering proceeds, both the domain expert and the knowledge engineer's understanding of the problem domain deepens. Throughout this process building communication between them is vital!

It is particularly important to get commitment from the expert to the project and the knowledge engineering process. It is going to take much time and effort from the expert, so the expert has to be convinced early on that it will be worth it. Certain characteristics are desirable in an expert: their expertise is acknowledged by their peers, they are articulate and they have the interest and the ability to *reason about* the reasoning process. Support, or at least tolerance, from the expert's management is, of course, necessary.

The rationale for modeling decisions should be recorded in a "style guide," which can be used to ensure that there is consistency across different parts of the model, even if they are developed at different times and by different people. Such a style guide should include naming conventions, definitions and any other modeling conventions. Most important is documenting the history of significant design decisions, so that subsequent decision making can be informed by that process. This is an aspect of **change management**. Another aspect is archiving the sequence of models developed. An automated tool, such as Unix's CVS, can be used to generate such archives, as well as avoid colliding changes when many people are working on the same project.

9.3 Modeling and elicitation

In this section we introduce methods for eliciting Bayesian network structure, parameters and preferences (utilities).

9.3.1 Variables and values

9.3.1.1 Types of node

When attempting to model large, complex domains it is important to limit the number of variables in the model, at least in the beginning, in order to keep the KE task tractable. The key is to determine which are the most important variables/nodes.

- One class of variables consists of those whose values the end-user wants to know about, the "output" nodes of the network, and are often referred to as the **target** or **query** nodes.
- The **evidence** or **observation** nodes play the role of "inputs" and can be identified by considering what sources of information about the domain are available, in particular, what evidence could be observed that would be useful in inferring the state of another variable.

© 2004 by Chapman & Hall/CRC Press LLC

- **Context** variables can be determined by considering sensing conditions and background causal conditions.
- **Controllable** variables are those whose values can be set by intervention in the domain environment (as opposed to simply observing their value).

It is important to note that the roles of nodes may change, depending how the BN is to be used. It is often useful to work backwards by identifying the query variables and "spreading out" to the related variables.

Let us return to the cancer diagnosis example used throughout Chapter 2 and look at it in terms of variable identification. The main interest of medical diagnosis is identifying the disease from which the patient is suffering; in this example, there are three candidate diagnoses. An initial modeling choice might be to have the query node *Disease*, with the observation nodes being *Dyspnoea* (shortness of breath), which is a possible symptom, and *X-ray*, which will provide another source of information. The context variables in this case are the background information about the patient, such as whether or not he is a *Smoker*, and what sort of exposure to *Pollution* he has had. In this simple diagnosis example, nothing has been described thus far that plainly falls into the category of a controllable variable. However, the doctor may well prefer to treat *Smoker* as a controllable variable, instead of as context, by attempting to get the patient to quit smoking. That may well turn into an example of a not-fully-effective intervention, as many doctors have discovered!

9.3.1.2 Types of values

When considering the variables, we must also decide what states, or values, the variable can take. Some common types of discrete nodes were introduced in §2.2.1: Boolean nodes, integer valued or multinomial categories. For its simplicity, and because people tend to think in terms of propositions (which are true or false), Boolean variables are very commonly employed. Equivalently, two-valued (binary) variables may be used, depending upon what seems most natural to the users. For example, when modeling the weather, the main weather node could be called *Weather*, and take the values {*fine, wet*}, or the node could be made a Boolean called *FineWeather* and take the values {*T, F*}. Other discrete node types will likely be chosen when potential observations are more fine-grained.

9.3.1.3 Common modeling errors

Discrete variable values must be **exhaustive** and **exclusive**, which means that the variable must take on exactly one of these values at a time. Modeling mistakes relating to each of these factors are common. For example, suppose that a preliminary choice for the *Disease* query node is to give it the values {*lungCancer, bronchitis, tuberculosis*}. This modeling choice isn't exhaustive, as it doesn't allow for the possibility of another disease being the cause of the symptoms; adding a fourth alternative $other$ alleviates the problem. However, this doesn't solve the second problem, since taking these as exclusive would imply that the patient can only suffer from one of these diseases. In reality it is possible (though of course uncommon) for a patient

© 2004 by Chapman & Hall/CRC Press LLC

to suffer from more than one of these, for example, *both* lung cancer and bronchitis. The best modeling solution here is to have distinct Boolean variables for each disease of interest, say the nodes *LungCancer*, *Bronchitis* and *Tuberculosis*. This model does not explicitly represent the situation where the patient suffers from another disease, but doesn't exclude it either.

Another common problem made by naive BN modelers is the creation of separate variables for different states of the same variable. For example, they might create both a *FineWeather* variable and a *WetWeather* variable (both Boolean). These states ought to be mutually exclusive, but once they are created as separate variables, the error is often "solved" by the addition of an additional arc between the nodes and a deterministic CPT that enforces the mutual exclusion. This is not very satisfactory, however, as the resultant structure is more complex than necessary. This is also an example of how the choices made when modeling nodes and states affects structure modeling.

9.3.1.4 Discretization

While it is possible to build BNs with continuous variables without discretization, the simplest approach is to **discretize** them, meaning that they are converted into multinomial variables where each value identifies a different subrange of the original range of continuous values. Indeed, many of the current BN software tools available (including Netica) require this. Netica provides a choice between its doing the discretization for you crudely, into even-sized chunks, or allowing the knowledge engineer more control over the process. We recommend exercising this control and discretize manually or else using a more sophisticated algorithm, such as [193].

TABLE 9.1
Alternative discretizations of an *Age* node with 4 values

Whole Population	Pension Population	Students
0-18	50-60	4-12
19-40	61-67	13-17
41-60	68-72	18-22
61-110	73-110	23-90

Consider the situation where you must discretize an *Age* node. One possible discretization might reflect the age distribution in the population. However, the optimal discretization may vary depending on the situation, from the whole population, to people who receive government pensions, or people engaged in full time study — see Table 9.1. Here, each range covers 25% of the target population.

This discretization is still based on the idea of even-sized chunks. It may be better to base the discretization on differences in effect on related variables. Table 9.2 shows a possible discretization when modeling the connection between *Age* and number of children, represented by the node *NumChildren*.

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 9.2
Discretization of an *Age* node based on differences in number of children

Age	P(NumChildren|Age)				
	0	1	2	3	>4
0-11	1	0	0	0	0
12-16	0.95	0.04	0.01	0	0
17-21	0.90	0.07	0.02	0.01	0
22-25	0.80	0.12	0.05	0.02	0.01
26-30	0.40	0.25	0.18	0.10	0.07
31-34	0.30	0.25	0.25	0.14	0.06
35-42	0.25	0.20	0.30	0.20	0.05
43-110	0.22	0.23	0.25	0.22	0.08

9.3.2 Graphical structure

There are several competing goals when building the graphical structure of a network. First, we would like to minimize the number of parameters, both in order to make the probability elicitation task easier and to simplify belief updating. These goals suggest fewer nodes, fewer arcs and smaller state spaces. On the other hand, we would obviously like to maximize the fidelity of the model, which sometimes requires more nodes, arcs and states (although excess detail can also decrease accuracy). A tradeoff must be made between building a more accurate model and the cost of additional modeling.

When deciding on the structure of the network, the key is to focus on the **relationships** between variables. There are many types of qualitative understanding that can help determine the appropriate structure for a domain.

9.3.2.1 Causal relationships

The first, and most important, are the **causal** relationships. As we discussed earlier (see §2.4), while orienting the arcs in a causal direction is not required, doing so maximizes the representation of conditional independence, leading to a more compact, simpler model. To establish the causal relationships, the knowledge engineer must identify the variables that could cause a variable to take a particular state, or prevent it from taking a particular state. Once identified, arcs should be added from those causal variables, to the affected variable. In all the following examples, we will see a natural combination of variable elicitation with the identification of relationships between variables. Sometimes it is appropriate to ask direct questions about causes. For example:

Q: *"What can cause lung cancer?"*
A: *"Smoking and pollution."*
Modeling: suggests arcs from those nodes to the *LungCancer* node.

© 2004 by Chapman & Hall/CRC Press LLC

Q: *"Is there anything which prevents TB?"*
A: *"There is a TB immunization available."*
Modeling: suggests an arc from *Immunization* to *TB*.

Alternatively, the same cause-to-effect structure may be identified by asking about effects. For example:
Q: *"What are the effects of lung cancer?"*
A: *"Shortness of breath and a spot on the lungs that may show up on the X-ray."*
Modeling: suggests the arcs from *LungCancer* to *X-ray* and *Dyspnoea*.

Another kind of causal relationship is **prevention**, when an effect will occur *unless* a preventative action is taken first.
Q: *"Is there anything that can prevent HIV causing AIDS?"*
A: *"Anti-viral drugs such as AZT can prevent the development of AIDS."*
Modeling: suggests arcs from both *HIV* and *AZT* to *AIDS*.

Another way of looking at this is to consider the possibility of **interference** to the causal relationship.
Q: *"Is there any factor that might interfere with cholesterol-lowering medication treating heart disease?"*
A: *"Yes, if the patient doesn't modify her diet, the medication won't be effective."*
Modeling: suggests arcs from both *Medication* and *Diet* to *HeartDisease*.

We have already seen other ways of describing this sort of interference relationship in §5.4.4 when we looked at problems with a sensor that affect its measuring capacity, with terms such as **moderates** and **invalidates** being used to describe an **unless** condition.

Enabling relationships exist where the enabling variable may under certain conditions permit, enhance or inhibit the operation of a cause.
Q: *"Is anything else required to enable the cholesterol-lowering medication to be affective against heart disease?"*
A: *"Yes, the patient must also modify her diet."*
Modeling: again, suggests arcs from both *Medication* and *Diet* to *HeartDisease*.

As we discussed earlier in §2.3.1, a v-structure in Bayesian networks, with two parents sharing a common effect, gives rise to a form of reasoning called "explaining away." To investigate whether this substructure exists, the knowledge engineer should ask a series of questions around **explanations** such as:
Q: *"Are both X and Y possible explanations for Z?"*
Q: *"Would finding out that Z is true increase your belief that both X and Y may be true?"*
Q: *"Would then finding out that X is true undermine your previously increased belief in Y?"*
We look further at causal interactions in §9.3.4.

© 2004 by Chapman & Hall/CRC Press LLC

9.3.2.2 Dependence and independence relationships

As we know, Bayesian networks encode conditional dependency and independency relationships. Very generally, a dependency can be identified by asking the domain expert:

Q: *"Does knowing something about the value of one variable influence your beliefs as to the value of the other variable, and vice versa?"*

Once we obtain a positive answer to this, we must determine what kind of dependency exists. If two variables are dependent regardless of the values of all other variables, then they should be directly connected.

Alternatively, some pairs of variables will be dependent only through other variables, that is, they are only conditionally dependent. d-separation tests can be used to check that the encoded relationships agree with the domain expert's intuitions. Unfortunately, the concept of d-separation can be difficult to interpret and takes time for domain experts to understand. We have been involved in the development of Matilda [24], a software tool that can help domain experts explore these dependencies. It supports a visual exploration of d-separation in a network, supplemented by a non-technical explanation of possible interactions between variables.

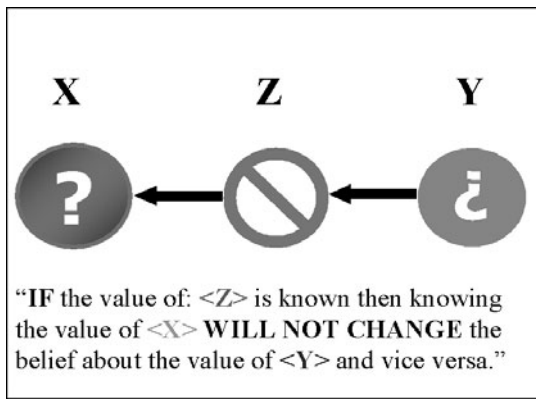


FIGURE 9.3
Matilda's visualization and explanation of "X is d-separated from Y given Z."

Matilda uses terms like 'direct causes,' 'causes' and 'paths' to explain graph relations. The term 'd-separation' is replaced by the more intuitive notion of 'blocking.' Extending this, a d-separating set found among parent nodes is called a 'simple blocking set.' This is a d-separating set but may contain proper subsets that can also d-separate. A minimal d-separating set found among parent nodes is called an 'essential blocking set.' This is a d-separating set from which no node can be removed without impairing the d-separation. A minimal d-separating set found anywhere in the network is called 'minimal blocking set.' Matilda's focus on the parent nodes

© 2004 by Chapman & Hall/CRC Press LLC

reflects the importance of these relationships in the modeling process.

In order to describe the general relation "X is d-separated from Y by Z" (i.e., $X \perp\!\!\!\perp Y \,|\, \mathbf{Z}$) Matilda gives the terms X, Y and Z a specific temporary role in the model. The nodes X become the **query** nodes, the nodes we want to reason about. The nodes Y become the **observation** nodes, the nodes we may observe. The nodes Z are **known** nodes, the values of which we already have. The notion of **influence**, in the context of "a change in the value of node X does not influence the value of node Y," is described by the term **change of belief** (*"knowing the value of node X does not change our belief in the value of node Y"*). To be sure, by giving the sets of nodes these roles the symmetry of the relation could be lost. To overcome this, the phrase *"vice versa"* needs to be added. Using these terms, Matilda describes the relation $X \perp\!\!\!\perp Y \,|\, \mathbf{Z}$ as:

> If the known nodes are observed, then knowing the value of the observation nodes will not change the belief about the value of the query nodes and vice versa.

Matilda's visualization of the relation "X is d-separated from Y given Z," with the corresponding verbal explanation is shown in Figure 9.3. The blocking relationship is shown using a common symbol for 'stop' or 'not allowed' (in red), for the Z node, with query nodes X indicated by "?" (purple), and observation nodes Y by an upside down "?" (blue). The color scheme ties the verbal and visual explanations together*.

Matilda allows the user to ask various types of questions about the relationships between variables. A BN for a fire alarm domain (extending one used in [219]), shown in Figure 9.4, will be used to illustrate this process.

Fire alarm example

We receive a phone call saying that everyone is leaving the building. The call can come from three different sources: a security center gets a report from a special sensor in the building. If the sensor reports 'Leaving' the security center calls us. We also get calls from kids who are playing practical jokes (mainly during the holidays) as well as from seriously concerned people who notice smoke coming out of the building (mainly after work-hours). The sensor is noisy. It sometimes does not report when everyone is leaving and sometimes reports leaving for no reason. If the fire alarm goes off, that causes people in the building to leave. The fire alarm can go off either because there is a fire or because someone tampers with it. The fire also causes smoke to rise from the building. In winter, fireplaces in the building can also cause the presence of smoke.

Matilda's Type 1 question. What is the relationship between two nodes?

This option allows the user to ask "What information d-separates two selected nodes?" by selecting two nodes X and Y and choosing between the three types of d-separating

*Note that we cannot reproduce the color scheme in this text.

© 2004 by Chapman & Hall/CRC Press LLC

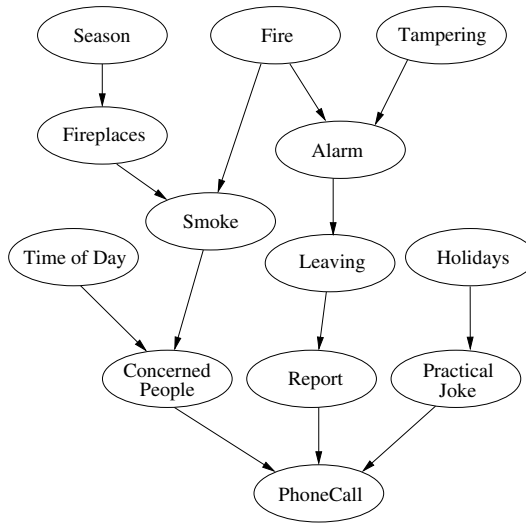


FIGURE 9.4
A BN solution for the fire alarm example.

sets: simple, essential and minimal blocking sets (as described above). Matilda then computes the selected type of d-separating set and highlights it using the blocking symbol.

Example: selected *X=Alarm*, *Y=Phone Call* (Figure 9.5).

A simple blocking set is the set of parents of *Phone Call*: {*Concerned People*, *Report*, *Practical Joke*}. The verbal explanation is: "IF the value of: <Practical Joke>, <Report>, <Concerned People> is known then knowing the value of <Alarm> WILL NOT CHANGE the belief about the value of <Phone Call> and vice versa."

An essential blocking set is the set {*Concerned People*, *Report*}. The verbal explanation is: "IF the value of: <Report>, <Concerned People> is known then knowing the value of <Alarm> WILL NOT CHANGE the belief about the value of <Phone Call> and vice versa."

The possible minimal blocking sets are: {*Fire*, *Report*}, {*Fire*, *Leaving*}, {*Report*, *Concerned People*}, {*Report*, *Smoke*}, {*Concerned People*, *Leaving*}, {*Leaving*, *Smoke*}. The verbal explanation of the first one is: "IF the value of: <Fire>, <Report> is known then knowing the value of <Alarm> WILL NOT CHANGE the belief about the value of <Phone Call> and vice versa."

Matilda's Type 2 question. When does a node become irrelevant?

Here, Matilda visualizes the relationships between one node and the rest of the network. This option allows the user to select a single node *X* and ask for a set of nodes that d-separates ("blocks" in Matilda terminology) this node from the rest of the structure. The number of such sets is potentially exponential, so Matilda highlights

© 2004 by Chapman & Hall/CRC Press LLC

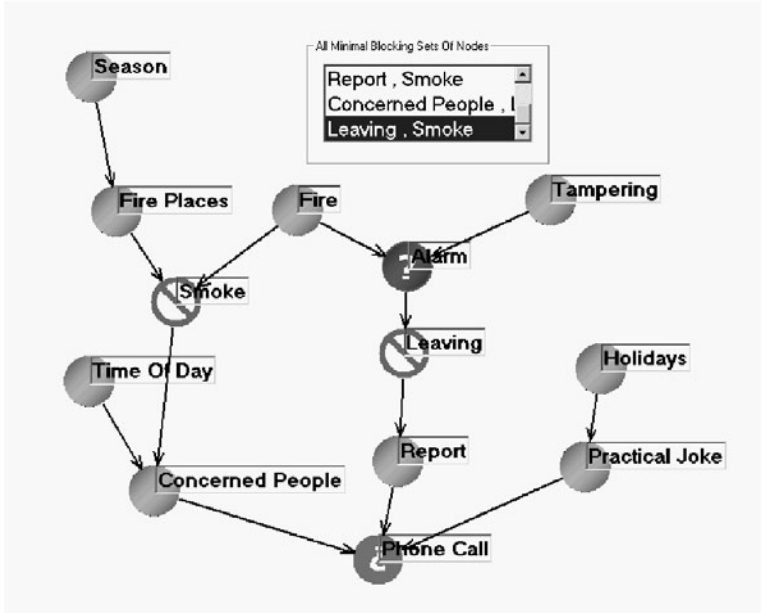


FIGURE 9.5
Matilda's Type 1 visualization of d-separation in the fire alarm example.

only one set, namely the Markov blanket (see §2.2.2).
Example: selected *X*=*Report* (Figure 9.6).

The Markov blanket for this node is the set {*Leaving* (parent), *Phone Call* (child), *Practical Joke*, *Concerned People* (child's parents)}. The verbal explanation is: "IF the value of: <Leaving>, <Phone Call>, <Practical Joke>, <Concerned People> is known then knowing the value of <Report> WILL NOT CHANGE the belief about the value of any other node and vice versa."

Matilda's Type 3 question. Given some information, what happens to the relationships between nodes?

Here, Matilda visualizes the relationships between *sets* of nodes. This option allows the user to select a set of nodes *X* (the query nodes) and a set of nodes *Z* (the prior information) and request the set of all *Y* nodes that are d-separated (blocked) from *X*. Matilda highlights all the nodes *Y* in response.
Example: selected *X*={*Phone Call*}, with prior information for nodes *Z*={*Smoke*, *Fire*} (Figure 9.7).

The nodes that are d-separated from *Phone Call* by *Smoke* and *Fire* are *Fire Places* and *Seasons*. The verbal explanation is "IF the value of: <Fire>, <Smoke> is known then knowing the value of <Phone Call> WILL NOT CHANGE the belief about the value of <Season>, <Fire Places> and vice versa."

In short, for various types of query, Matilda visualizes the relation "X is d-separated

© 2004 by Chapman & Hall/CRC Press LLC

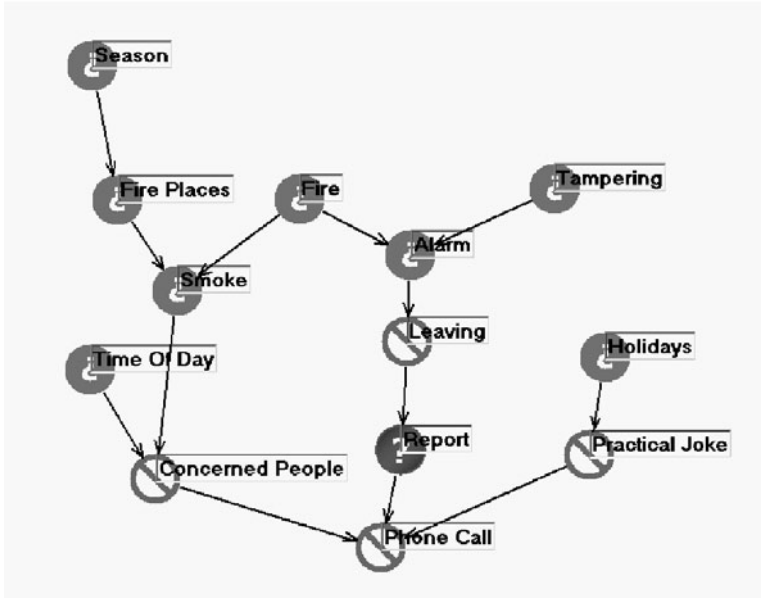


FIGURE 9.6
Matilda's Type 2 visualization of a Markov blanket for *Report*.

from Y given Z." In the first type, the user chooses *X* and *Y*; in response the tool highlights *Z*. In the second question type, the user chooses *X*; in response the tool highlights *Z*. In the third question type, the user chooses *X* and *Z*; in response the tool highlights *Y*. Note that in the first two types of queries the tool identifies the d-separating sets of nodes, whereas in the latter type of query the user is asking the question with regard to a specific d-separating set of nodes.

Case-studies [24] suggest that Matilda is useful in understanding networks as they are being built and in understanding the consequences of different possible design choices. It can be used not just by BN experts, but also by domain experts to validate the network and identify potential problems with the structure. It can be used to investigate an existing network at any stage during development. It is especially helpful in investigating networks built by automated methods, when prior intuitive understanding may be weak.

9.3.2.3 Other relationships

There are other less explicit indications of the correct network structure.

Association relationships occur when knowing a value of one variable provides information about another variable. By Reichenbach's Principle of the Common Cause, some causal nexus must explain the association; although any active (unblocked) path will do this, just the information that there is *some* active path may serve as a beginning in building the causal structure. The absence of an uncondi-

© 2004 by Chapman & Hall/CRC Press LLC

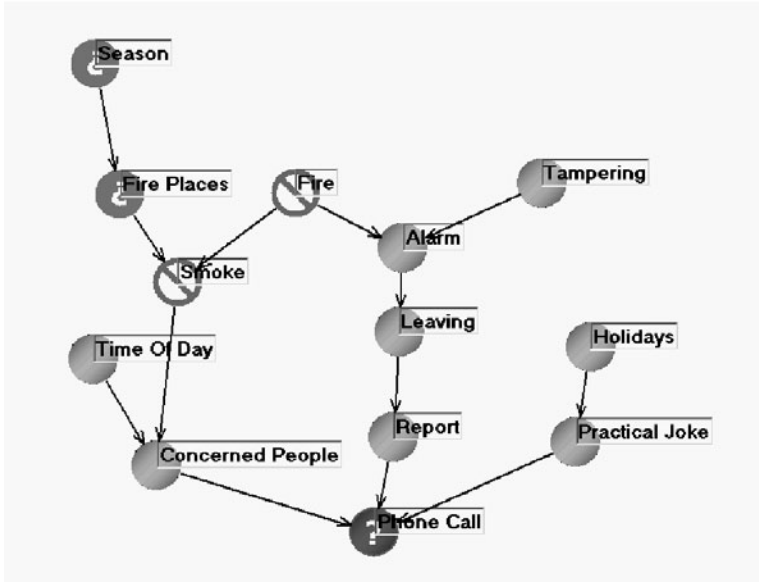


FIGURE 9.7
Matilda's Type 3 visualization.

tional association is also useful to know. Thus, in the fire alarm example of Figure 9.4, there is no marginal dependence between *Time of Day* and *Fire*, which is an implication worth checking. On the other hand, there is an implied association between *Report* and *Smoke* (through their common ancestor *Fire*).

In many domains, there may be a known **temporal ordering** of variables, where one event or value change occurs *before* another. The known ordering may be either total or partial. In either case, the temporal information will restrict the orientation of some of the arcs, assuming you are building a causal network.

9.3.2.4 Combining discrete and continuous variables

The initial work on BNs with continuous variables [217, 251] only allowed variables with linear Gaussian distributions. A method for combining discrete and continuous variables was proposed [168] and implemented in cHugin [210]; this approach did not allow for discrete children of continuous parents. The usual solution is to discretize all continuous variables at some stage. As suggested above, this is best done manually at the moment. In the future, we expect the best discretization methods will be incorporated in Bayesian network tools, so that the process can be automated, given sufficient statistical data.

There has also been some research on extending the inference algorithms to cope with some special cases of continuous variables and inference, and these no doubt will eventually be applied in application tools (e.g., [170]).

© 2004 by Chapman & Hall/CRC Press LLC

9.3.3 Probabilities

The parameters for a BN are a set of conditional probability distributions of child values given values of parents. There is one distribution for each possible instantiation of parent variables; so the bad news is that the task of probability assessment is exponential in the number of parent variables. If there is local structure (see §7.4), of course, the number of parameters to be estimated is reduced.

9.3.3.1 Parameter sources

There are three possible parameter sources.

1. Data

We have previously described some specific methods for learning parameters from domain data (see Chapter 7). General problems with data include: noise, missing values and small samples. These can be overcome to a certain extent by using robust data mining techniques. It is also useful to instrument the domain for collecting data to use in the future, by adaptation of parameters.

2. Domain Experts

The most basic problem is finding suitable experts who have the time and interest to assist with the modeling process. Another difficulty is that humans, including expert humans, almost always display various kinds of bias in estimating probabilities. These include:

- **Overconfidence**, the tendency to attribute higher than justifiable probabilities to events that have a probability sufficiently greater than 0.5. Thus, an event which objectively has a probability of 0.9 will usually be attributed a probability that is somewhat higher (see Figure 9.8). To be sure, expertise itself has been found to have a moderating effect on this kind of miscalibration [239].
- **Anchoring**, the tendency for subsequent estimates to be "weighed down" by an initial estimate. For example, if someone is asked to estimate the average age of coworkers and begins that process by estimating the average age of those in a meeting, a high (or low) age at the meeting will very probably bias the estimate upwards (or downwards) [137].
- **Availability**, that is, assessing an event as more probable than is justifiable, because it is easily remembered or more salient [278].

There is a large, and problematic, literature on assessing these biases and proposals to debias human probability estimates. The latter have met with highly limited success. The best advice we can give, short of an in-depth exploration of the literature (and we do recommend some such exploration, as described in §9.6 below), is to be aware of the existence of such biases, discuss them with the experts who are being asked to make judgments and to take advantage of whatever statistics are available, or can be made available, to test human judgments against a more objective standard.

Two attributes of a good elicitation process are (adapted from Morgan and Henrion [194, pp. 158-159]):

1. The expert should be apprised of what is known about the process, especially the nearly universal tendency to overconfidence and other forms of bias. In

© 2004 by Chapman & Hall/CRC Press LLC

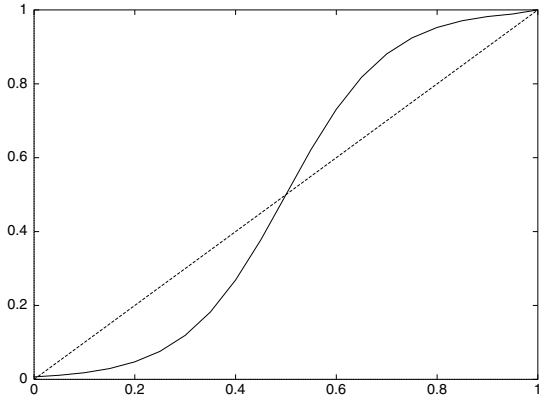


FIGURE 9.8
Overconfidence curve: subjective probability (vertical axis) vs. objective probability (horizontal axis).

order to avoid some of the problems, values should be elicited in random order and the expert *not* given feedback on how the different values fit together until a complete set has been elicited.

2. The elicitation process is not simply one of requesting and recording numbers, but also one of refining the definitions of variables and terms to be used in the model. What values are elicited depends directly upon the interpretation of terms and these should be made as explicit as possible and recorded during the elicitation. This is a part of the process management described earlier in §9.2.4.

3. The Literature

There may be a published body of knowledge about the application domain. One common problem with published statistics is sparseness. For example, in a medical diagnosis domain, the probability of observing a symptom given a disease, $P(Symptom|Disease = T)$, may be available in the medical textbooks ("80% of patients with TB will present with a cough"), but not $P(Symptom|Disease = F)$, the information about the frequency of the symptom occurring when the disease is not present. There is also a bias in what information is available in the literature: the fact of its publication reflects interest. These problems can be moderated by using expert opinion to review the proposed parameterization.

There is a risk involved with combining parameters from different sources, with different biases. It is not necessarily the case that combining sources with multiple biases smoothly averages out to no bias, or to a readily measurable and manageable bias. For an informal example, one author shot baskets at the Exploratorium in San Francisco with some biased goggles. At first, this resulted in missing to the left, but fairly soon the brain accommodated the misdirection and the basketball started hitting the hoop again. As soon as the goggles were removed, however, the shots

© 2004 by Chapman & Hall/CRC Press LLC

missed to the right! The unbiased new input was being combined with an older bias, leading to error.

9.3.3.2 Probability elicitation for discrete variables

For discrete variables, one approach is direct elicitation, where an expert provides a number such as "the probability is 0.7." However, given the problems people have with such estimation, noted above, other elicitation techniques might be considered. People are often better at providing frequencies rather than probabilities, such as "1 in 4" or "1 in 10,000" [92], especially for situations where the probabilities involved are very large or very small. Assessing extreme probabilities directly is difficult, and orders of magnitude assessments might be tried.

Assessing by odds is often useful. For example, given that $Y = y$, a domain expert may report:

"It is three times more likely that variable X *has the value* x_1 *than* x_2*."*

This information gives the equation $P(X = x_1|Y = y) = 3P(X = x_2|Y = y)$, and only one probability has to be elicited.

An alternative approach is to use qualitative assessment, where the domain expert describes the probability in common language, such as "very high" or "unlikely." Such probability elicitation using a scale with numerical and verbal anchors is described in [280]. The verbal cues in that scale were `certain`, `probable`, `expected`, `fifty-fifty`, `uncertain`, `improbable` and `impossible`. Having been translated from Dutch some of these cues are inappropriate, for example "`uncertain`" is ambiguous and could be replaced with "`unlikely`." Because these qualitative phrases can be ambiguous (in fact, this is the problem of **linguistic uncertainty**), they can cause miscommunication, especially where more than one domain expert is involved, unless they are themselves calibrated. It is advisable to do the mapping of verbal levels to actual probabilities (called the **verbal map**) separately from the probability elicitation exercise. The verbal map should be customized to suit the individual expert. We have developed a software tool called VE (for Verbal Elicitor) in conjunction with the Netica BN software (see §B.4.8), which supports the qualitative elicitation of probabilities and verbal maps [109].

An example of VE's main window for eliciting qualitative probabilities is shown in Figure 9.9, together with the window editing the verbal map for this tool. The probabilities associated with each verbal cue are set using a slider bar, and other verbal cues can be added if desired.

A common problem when eliciting probabilities is that an expert may specify probabilities that are **incoherent**, failing to satisfy the probability axioms. For example, suppose the verbal mapping shown in Figure 9.9 is being used and that the expert provides the qualitative assessment of the probabilities for X-ray results given in Table 9.3.

These probabilities are incoherent, since they do not sum to one for each conditioning case. VE provides a function to correct incoherent probabilities automatically; it uses an iterative, linear optimization to find verbal mappings that are coherent and as close as possible to the original map.

© 2004 by Chapman & Hall/CRC Press LLC

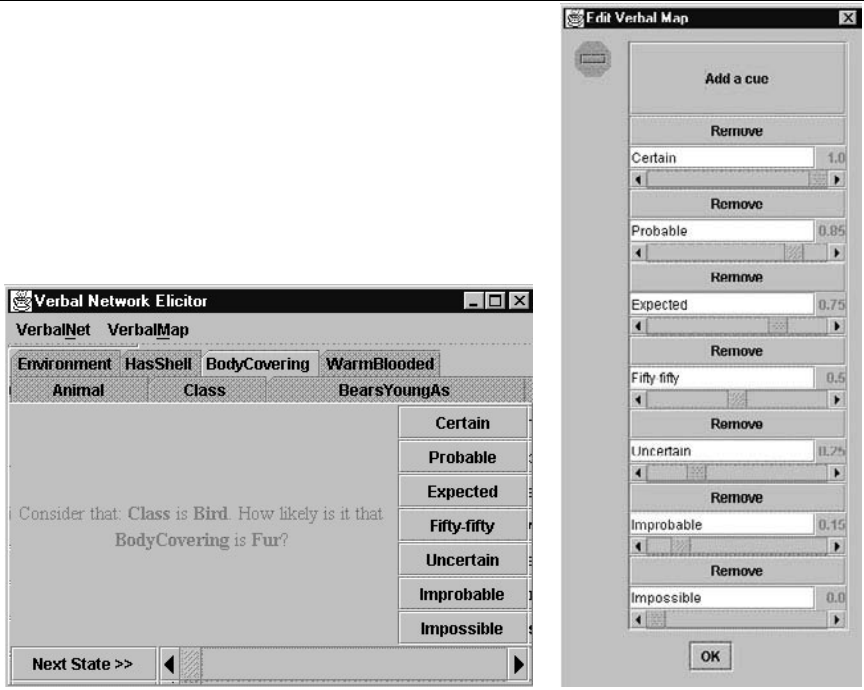


FIGURE 9.9
VE: main elicitation window (left) and the verbal map editing window (right).

9.3.3.3 Probability elicitation for continuous variables

The easiest continuous variables to parameterize are those which are distributed according to a **parametric model**, that is, those which are fully characterized by a limited number of parameters. The most common example is the Normal distribution (also known as the Gaussian or "bell curve"), $N(\mu, \sigma^2)$, which is characterized by just two parameters, the mean μ and the variance σ^2. Normal distributions are used to model: noisy measurement processes (e.g., velocities, the positions of stars); the central tendencies (average values) of almost any process (e.g., average age or income of samples from a population), which is justified by the central limit theorem of probability theory; and, by a kind of metaphorical extension, summative measures of

TABLE 9.3
Incoherent qualitative assessments for the *X-ray* CPT

$P(X = Pos \mid Cancer = T)$ = Probable
$P(X = Neg \mid Cancer = T)$ = Improbable
$P(X = Pos \mid Cancer = F)$ = Uncertain
$P(X = Neg \mid Cancer = F)$ = Certain

© 2004 by Chapman & Hall/CRC Press LLC

a complex of processes when the individual processes are not well understood (e.g., IQ). Another popular parametric distribution is the exponential, which is often used to model life- and death-processes, such as the life span of an electrical part. Still other parametric continuous distributions are the Gamma, Chi-square and F distributions. These, and the discrete parametric distributions (e.g., binomial and Poisson), form much of the material of probability and statistics classes. They are worth learning about since in every case one need only obtain a good estimate of only a few parameters from an expert in order to obtain a good estimate of the entire probability distribution — assuming, of course, that the variable in question is properly modeled by the chosen family of distributions!

If the problem is not as simple as estimating a couple of parameters, like the mean and variance of a normal distribution, then most common is to elicit estimates of key values for the probability density function. Recall from §1.3.2 that if the continuous density function is $f(x)$, then the cumulative distribution function $F(x)$ is

$$F(x) = P(X \leq x) = \int_{x' \leq x} f(x')dx' \tag{9.1}$$

A likely route to estimating the density function is by bi-sectioning it. First, elicit the median, the value at which X is equally likely to be found either above or below. Then elicit the 25th percentile by "bisecting" the region below the median into two further equally likely regions, and then the 75th percentile analogously. This process can be continued until the density has been sufficiently well refined for the problem, or until the expert can no longer make meaningful distinctions. This kind of estimation may be usefully accompanied by the expert simply sketching her or his impression of the shape of the density function; otherwise one might overlook something simple and important, such as whether the density is intended to be unimodal or bimodal, skewed or symmetric, etc.

Having estimated a distribution in this fashion, it may be best and simplest to find a parametric model (with parameter values) which reproduces the estimated distribution reasonably well and use that for your Bayesian network model.

9.3.3.4 Support for probability elicitation

Visual aids are known to be helpful and should be used for probability elicitation (see Figure 9.10). With a **pie chart** the expert aims to size a slice of the "pie" so that a spinner will land in that region with the probability desired. A **histogram** may help the expert to order discrete events by probability. As we mentioned, simple freehand drawings of probability distributions can also be informative.

Lotteries can be used to force estimates of either probabilities or utilities, in techniques going back to Ramsey [231]. Given clear utility values, say dollars, you can elicit someone's estimated probability of an uncertain event E by finding at what point the person is indifferent between two gambles: the first one paying, say, \$100 if E comes true; the second one paying, say, \$1 million if a (free) lottery ticket *Wins*. Since the two gambles are considered equivalued, we have (where N is the number

© 2004 by Chapman & Hall/CRC Press LLC

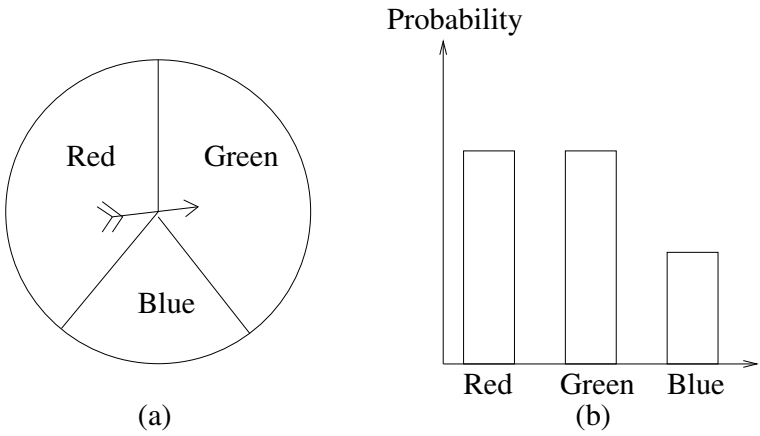


FIGURE 9.10
Examples of visual aids for probability elicitation: (a) pie chart; (b) histogram.

of lottery tickets required to reach indifference):

$$EU(Gamble) = \$100P(E) = EU(Lottery) = \frac{\$1000000}{N} \tag{9.2}$$

Hence, $P(E) = 10000/N$. Lotteries can be used analogously to elicit unclear utilities for an outcome state by manipulating the probability of reaching that state until the expert is indifferent between the proposed gamble and some lottery ticket with known value and probability of winning.

Our VE software tool provides a number of useful automated functions that facilitate probability elicitation. One function normalizes CPTs, allowing users to specify ratios in place of probabilities[†]. For example, if the expert thinks that someone has three times the chance of getting cancer as not getting cancer (say if they are a smoker), they can specify

$$P(Cancer = T|Smoker = T) : P(cancer = F|Smoker = F)$$

as 3:1, which the tool translates into the probabilities 0.75 and 0.25, respectively.

VE can also perform a maximum entropy fill of CPTs, where the remaining probabilities are filled in uniformly with the probability remaining after subtracting supplied probabilities from 1. This means the expert need only provide probabilities for combinations about which s/he is confident. For example, if the variable A has states a_1, a_2 and a_3, and the probability for $A = a_1$ given some combination of values for its parent variables, $P(A = a_1|B = T, C = F)$ is set to 0.95, then $P(A = a_2|B = T, C = F)$ and $P(A = a_3|B = T, C = F)$ will both be set to 0.025 automatically.

[†]Note that some other software packages have this feature — see §B.4.

© 2004 by Chapman & Hall/CRC Press LLC

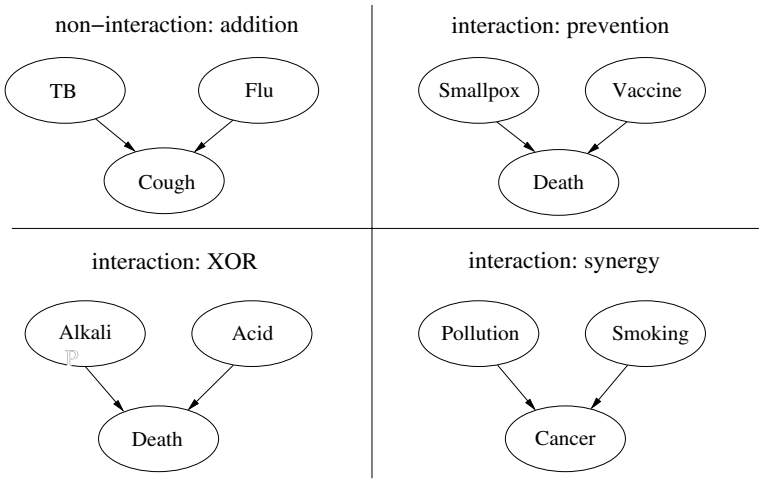


FIGURE 9.11
Different qualitative causal relationships.

9.3.4 Local structure

When parameterizing the relation between parents and a child node, the possibility of there being "local structure" was discussed in Chapter 7 in the context of automated parameter learning. It is also of interest in the elicitation process, of course. There are various *kinds* of causal interaction, such as those displayed in Figure 9.11. A classic example of interaction between causes is **XOR**, where each cause cancels the other out. The alkali/acid case (§7.4.1) is an example of this: *one might ingest alkali, and die; one might instead ingest acid, and die; but if one ingests both alkali and acid together, then one may well not die.*

Other causal interactions include **prevention**, where one causal factor intervenes to stop another, such as a *Vaccine* preventing *Smallpox* leading to *Death*. And again there is the possibility of **synergy**, where the effects are reinforced by the occurrence of both causes beyond the mere addition of the effects independently. All of these relationships can be looked for explicitly during an elicitation process. For example,
Q: *"Given that* Acid *and* Alkali *are independently causes of* Death*, when taken jointly what happens to the risk?"*
A: *"It is decreased."*
Modeling: in this case, the causal interaction is clearly an XOR type.

These kinds of interaction imply that the probabilities associated with one or more of the possible instantiations of the parents are *independent* of the probabilities associated with the other parent instantiations. For example, knowing what happens when you ingest Acid but not Alkali tells you little or nothing about what happens when you ingest both.

Local structure is the opposite situation: there is some structure across the different parent instantiations that allows you to infer some probabilities from the others. We

© 2004 by Chapman & Hall/CRC Press LLC

have already discussed some different models of non-interaction (local structure) in Chapter 7, namely noisy-or, logit models and classification tree models, all of which allow a more compact specification of the CPT under non-interaction. In our original noisy-or example of *Flu*, *TB* and *SevereCough* (in Figures 7.4 and 9.11), this relationship would be identified by negative answers to the questions:
Q: *"Does having* TB *change the way that* Flu *causes a* Severe Cough*?"*
A: *"No."*
Q: *"Similarly, does* Flu *change the way that* TB *causes a* Severe Cough*?"*
A: *"No."*

Assuming that *Flu* and *TB* have been identified as causes of *Severe Cough*, these answers imply that the probability of not having the symptom is just the product of the independent probabilities that every cause present will fail to induce the symptom. (This is illustrated in Table 7.2 and explained in the surrounding text.) Given such a noisy-or model, we only need to elicit three probabilities: namely, the probability that *Flu* will fail to show the symptom of *Severe Cough*, the probability that *TB* will fail to show the symptom and the background probability of not having the symptom.

Local structure, clearly, can be used to advantage in either the elicitation task or the automated learning of parameters (or both).

One method for eliciting local structure is "elicitation by partition" [102, 90, 181]. This involves dividing the joint states of parents into subsets such that each subset shares the conditional probability distribution for the child states. In other words, we **partition** the CPT, with each subset being a **partition element**. The task is then to elicit one probability distribution per partition element. Suppose, for example, that the probability of a high fever is the same in children, but not adults, for both the flu and measles. Then the partition for the parent variables *Flu*, *Measles* and *Age* and effect variable *Fever* would produce two partition elements for adults and one for children.

Note that this elicitation method directly corresponds to the use of classification trees and graphs in automated parameter learning (see §7.4.3). So, one way of doing partitioning is by building that corresponding classification tree by hand. Figure 9.12 illustrates this possibility for a simple network.

9.3.4.1 Divorcing

Another way to reduce the number of parameters is to alter the graph structure; **Divorcing multiple parents** is a useful technique of this type. It is typically applied when a node has many parents (and so a large CPT), and when there are likely groupings of the parents in terms of their effect on the child. Divorcing means introducing an intermediate node that summarizes the effect of a subset of parents on a child.

An example of divorcing is shown in Figure 9.13; the introduction of variable Z divorces parent nodes A and B from the other parents C and D. The method of divorcing parents was used in Munin [9]. Divorcing involves a trade-off between new structural complexity (the introduction of additional nodes and arcs) and parameter simplification. We note that divorcing may also be used for purposes other than reducing the number of parameters, such as dealing with overconfidence [209].

© 2004 by Chapman & Hall/CRC Press LLC

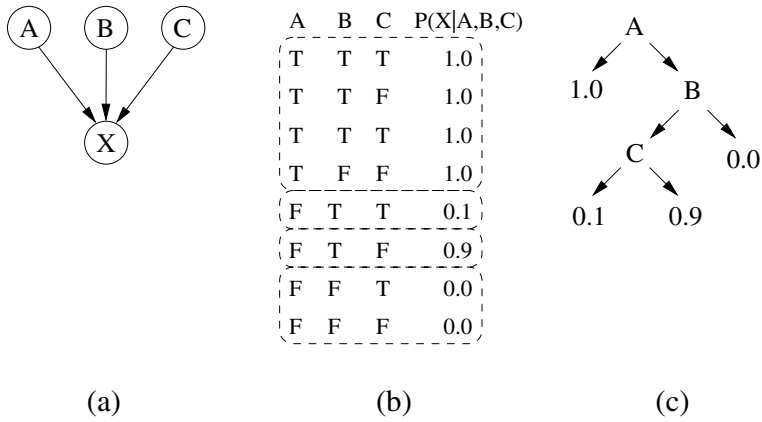


FIGURE 9.12
Local CPT structure: (a) the Bayesian network; (b) the partitioned CPT; (c) classification tree representation of the CPT.

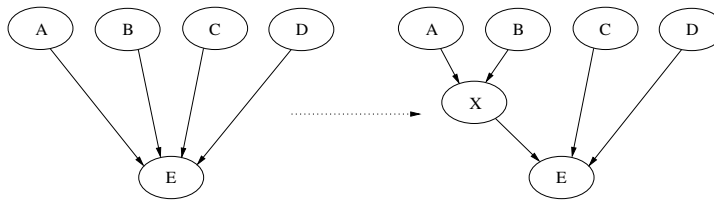


FIGURE 9.13
Divorcing example.

Divorcing example: search and rescue

Part of the land search and rescue problem is to predict what the health of a missing person will be when found. Factors that the domain expert initially considered were external factors, such as the temperature and weather, and features of the missing person, such as age, physical health and mental health.

This example arises from an actual case of Bayesian network modeling [24]. A BN constructed by the domain expert early in the modeling process for a part of the search and rescue problem is shown in Figure 9.14(a). The number of parameters required for the node *HealthWhenFound* is large. However, there is a natural grouping of the parent nodes of this variable into those relating to weather and those relating to the missing person, which suggests the introduction of intermediate nodes. In fact, it turns out that what really matters about the weather is a combination of temperature and weather conditions producing a dangerous wind chill, which in turn may lead to hypothermia. The extent to which the person's health deteriorates due to hypothermia depends on *Age* and the remaining personal factors, which may be summarized by a *ConditionWhenLost* node, hence, the introduction of three mediating variables,

© 2004 by Chapman & Hall/CRC Press LLC

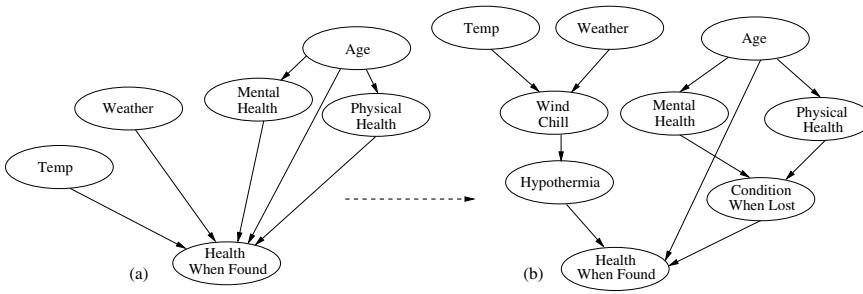


FIGURE 9.14
Search and rescue problem: (a) the original BN fragment; (b) the divorced structure.

WindChill, *Hypothermia* and *ConditionWhenLost*, as shown in Figure 9.14(b). The result is a structure that clearly reduces the number of parameters, making the elicitation process significantly easier[‡].

9.3.5 Variants of Bayesian networks

9.3.5.1 Qualitative probabilistic networks (QPN)

Qualitative probabilistic networks (QPN) [296] provide a qualitative abstraction of Bayesian networks, using the notion of positive and negative influences between variables. Wellman shows that QPNs are often able to make optimal decisions, without requiring the specification of the quantitative part of the BN, the probability tables.

9.3.5.2 Object-oriented BNs (OOBNs)

Object-oriented BNs are a generalization of BNs proposed by Koller and Pfeffer [151]. They facilitate network construction with respect to both structure and probabilities by allowing the representation of commonalities across variables. Most important for probability elicitation, they allow inheritance of priors and CPTs. However, OOBNs are not supported by the main BN software packages (other than Hugin Expert) and hence are not as yet widely used[§].

[‡]This example comes from a case study undertaken when evaluating Matilda [24]. Examination of the initial structure with Matilda led to the expert identifying two missing variables, *Wind Chill* and *Hypothermia*. (Note that these two could be combined in the network shown without any loss of information; this would not work, however, in the larger network.) Subsequent analysis led to the divorce solution.

[§]Other packages do support subnetwork structures, see §B.4.

© 2004 by Chapman & Hall/CRC Press LLC

9.3.6 Modeling example: missing car

We will now run through the modeling process for an example we have used in our BN courses, the missing car problem (Figure 9.15)[¶]. This is a small problem, but illustrates some interesting modeling issues.

The first step is to highlight parts of the problem statement which will assist in identifying the Bayesian network structure, which we have already done in Figure 9.15. For this initial analysis, we have used the annotations:

- possible situations (→ nodes and values)
- **words connecting situations** (→ graphical structure)
- indication of evidence (→ observation nodes)
- `focus of reasoning` (→ query nodes)

The underlined sections suggest five Boolean nodes, shown in the table in Figure 9.15, consisting of two `query` nodes and three observation nodes.

Marked-Up Problem Statement

John and Mary Nguyen arrive home after a night out to find that their second car is not in the garage. Two **explanations** *occur to them:* **either** *the car has been stolen* **or** *their daughter Sam has borrowed the car without permission. The Nguyens know that* **if** *the car was stolen,* **then** *the garage will probably show signs of forced entry. The Nguyens also know that Sam has a busy social life, so* **even if** *she didn't borrow the car, she* **may be** *out socializing.* `Should they be worried about their second car being stolen and notify the police, or has Sam just borrowed the car again?`

Preliminary Choice of Nodes and Values

Type	Description	Name	Values
Query	Has Nguyen's car been stolen?	*CarStolen*	*T,F*
	Has Sam borrowed the car?	*SamBorrowed*	*T,F*
Observation	See 2nd car is missing	*MissingCar*	*T,F*
	See signs of forced entry to garage?	*ForcedEntry*	*T,F*
	Check whether or not Sam is out?	*SamOut*	*T,F*

FIGURE 9.15
Missing car problem: preliminary analysis.

There are two possible **explanations**, or **causes**, for the car being missing, suggesting causal arcs from *CarStolen* and *SamBorrowed* to *MissingCar*. Signs of forced entry are a likely **effect** of the car being stolen, which suggests an arc from *CarStolen* to *ForcedEntry*. That leaves only *SamOut* to connect to the network. In the problem as stated, there is clearly an association between Sam being out and Sam borrowing the car; which way should the arc go? Some students add the arc

[¶] The networks developed by our students are available from the book Web site.

© 2004 by Chapman & Hall/CRC Press LLC

$SamOut \longrightarrow SamBorrowed$ because that is the direction of the inference or reasoning. A better way to think about it is to say "Sam borrowing the car *leads to* Sam being out," which is both a causal and a temporal relationship, to be modeled by $SamBorrowed \longrightarrow SamOut$.

The BN constructed with these modeling choices is shown in Figure 9.16(a). The next step of the KEBN process is to determine the parameters for this model; here we underline text about the "numbers."

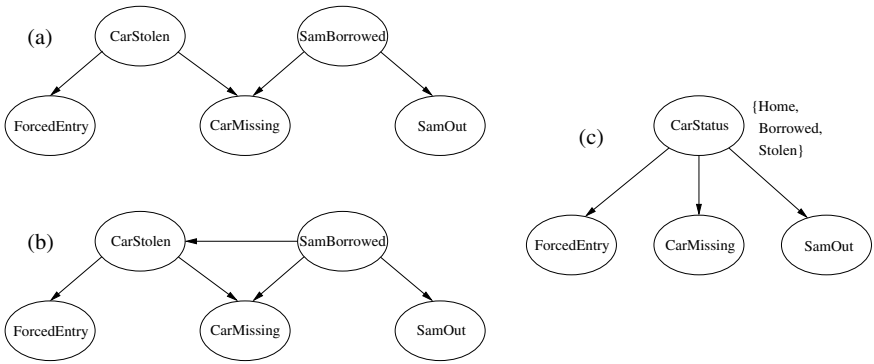


FIGURE 9.16
Alternative BNs for the missing car problem.

Additional problem information

The Nguyens know that the rate of car theft in their area is about 1 in 2000 each day, and that if the car was stolen, there is a 95% chance that the garage will show signs of forced entry. There is nothing else worth stealing in the garage, so it is reasonable to assume that if the car isn't stolen, the garage won't show signs of forced entry. The Nguyens also know that Sam borrows the car without asking about once a week, and that even if she didn't borrow the car, there is a 50% chance that she is out.

There are two root nodes in the model under consideration (Figure 9.16(a)). The prior probabilities for these nodes are given in the additional information:

- $P(CarStolen)$ is the rate of car theft in their area of "1 in 2000," or 0.0005
- $P(SamBorrowed)$ is "once a week," or ≈0.143

The conditional probabilities for the*ForcedEntry* node are also straightforward:

- $P(ForcedEntry = T|CarStolen = T) = 0.95$ (from the "95% chance")
- $P(ForcedEntry = T|CarStolen = F) = 0$

In practice, it might be better to leave open the possibility of there being signs of forced entry even though the car hasn't been stolen (something else stolen, or thieves interrupted), by having a very high probability (say 0.995) rather than the 1;

© 2004 by Chapman & Hall/CRC Press LLC

this removes the possibility that the implemented system will grind to a halt when confronted with a combination of impossible evidence (say, $CarStolen = F$ and $ForcedEntry = T$). There is a stronger case for

- $P(SamOut = T|SamBorrowed = T) = 1$

The new information also gives us

- $P(SamOut = T|SamBorrowed = F) = 0.5$

So the only CPT parameters still to fill in are those for the *MissingCar* node. But the following probabilities are clear:

- $P(MissingCar = T|CarStolen = T, SamBorrowed = F) \approx 1$
- $P(MissingCar = T|CarStolen = F, SamBorrowed = T) \approx 1$

If we want to allow the possibility of another explanation for the car being missing (not modeled explicitly with a variable in the network), we can adopt

- $P(MissingCar = T|CarStolen = F, SamBorrowed = F) = \alpha > 0$

Finally, only one probability remains, namely

- $P(MissingCar = T|CarStolen = T, SamBorrowed = T)$

Alert! Can you see the problem? How is it possible for Sam to have borrowed the car *and* for the car to have been stolen? These are mutually exclusive events!

The first modeling solution is to add an arc $SamBorrowed \longrightarrow CarStolen$ (or vice versa)(see Figure 9.16(b)), and make this mutual exclusion explicit with

- $P(CarStolen = T|SamBorrowed = T) = 0$

In which case, it doesn't matter what numbers are put in the CPT for $P(Missing\ Car = T|CarStolen = T, SamBorrowed = T)$. While this "add-an-arc" modeling solution "works," it isn't very elegant. Although Sam's borrowing the car will prevent it from being stolen, it is equally true that someone's stealing the car will prevent Sam's borrowing it!

A better solution is to go further back in the modeling process and re-visit the choice of nodes. The Nguyens are actually interested in the state of their car, whether it has been borrowed by their daughter, stolen, or is safe at home. So instead of two Boolean query nodes, we should have a single query node *CarStatus* with possible values $\{home, borrowed, stolen\}$. This simplifies the structure to that shown in Figure 9.16(c). A disadvantage of this simplified structure is that it requires additional parameters about other relationships, such as between signs of forced entry and Sam borrowing the car, and the car being stolen and Sam being out.

© 2004 by Chapman & Hall/CRC Press LLC

9.3.7 Decision networks

Since the 1970s there have been many good software packages for decision analysis. Such tools support the elicitation of decisions/actions, utilities and probabilities, building decision trees and performing sensitivity analysis. These areas as well covered in the decision analysis literature (see for example, Raiffa's *Decision Analysis* [229], an excellent book!).

The main differences between using these decision analysis tools and knowledge engineering with decision networks are: (1) the scale — decision analysis systems tend to require tens of parameters, compared to anything up to thousands in KEBN; and (2) the structure, as decision trees reflect straight-forward state-action combinations, without the causal structure, prediction and intervention aspects modeled in decision networks.

We have seen that the KE tasks for ordinary BN modeling are deciding on the variables and their values, determining the network structure, and adding the probabilities. There are several additional KE tasks when modeling with decision networks, encompassing decision/action nodes, utility (or value) nodes and how these are connected to the BN.

First, we must model what decisions can be made, through the addition of one or more decision nodes. If the decision task is to choose only a single decision at any one time from a set of possible actions, only one decision node is required. A good deal can be done with only a single decision node. Thus, a single *Treatment* decision node with options {*medication*, *surgery*, *placebo*, *no-treatment*} precludes consideration of a combination of surgery and medication. However, combinations of actions can be modeled within the one node, for example, by explicitly adding a *surgery-medication* action. This modeling solution avoids the complexity of multiple decision nodes, but has the disadvantage that the overlap between different actions (e.g., medication and surgery-medication) is not modeled explicitly.

An alternative is to have separate decision nodes for actions that are not mutually exclusive. This can lead to new modeling problems, such as ensuring that a "no-action" option is possible. In the treatment example, the multiple decision node solution would entail 4 decision nodes, each of which represented the positive and the negative action choices, e.g., {*surgery*, *no-surgery*}. The decision problem becomes much more complex, as the number of combinations of actions is $2^4 = 16$. Another practical difficulty is that many of the current BN software tools (including Netica) only support decision networks containing either a single one-off decision node or multiple nodes for sequential decision making. That is, they do not compute optimal *combinations* of decisions to be taken at the same time.

The next KE task for decision making is to model the utility of outcomes. The first stage is to decide what the unit of measure ("utile") will mean. This is clearly domain specific and in some cases fairly subjective. Modeling a monetary cost/benefit is usually fairly straightforward. Simply adopting the transformation \$1 = 1 utile provides a linear numeric scale for the utility. Even here, however, there are pitfalls. One is that the utility of money is not, in fact, linear (as discussed in §4.2): the next dollar of income undoubtedly means more to a typical student than to a millionaire.

© 2004 by Chapman & Hall/CRC Press LLC

When the utile describes subjective preferences, things are more difficult. Requesting experts to provide preference orderings for different outcomes is one way to start. Numeric values can be used to fine tune the result. As noted previously, hypothetical lotteries can also be used to elicit utilities. It is also worth noting that the domain expert may well be the wrong person for utility elicitation, either in general or for particular utility nodes. It may be that the value or disvalue of an outcome arises largely from its impact on company management, customers or citizens at large. In such cases, utilities should be elicited from those people, rather than the domain experts.

It is worth observing that it may be possible to get more objective assessments of value from social, business or governmental practice. Consider the question: What is the value of a human life? Most people, asked this question, will reply that it is impossible to measure the value of a human life. On the other hand, most of *those* people, under the right circumstances, will go right ahead and measure the unmeasurable. Thus, there are implicit valuations of human life in governmental expenditures on health, automobile traffic and air safety, etc. More explicitly, the courts frequently hand down judgments valuing the loss of human life or the loss of quality of life. These kinds of measurement should not be used naively — clearly, there are many factors influencing such judgments — but they certainly can be used to bound regions of reasonable valuations. Two common measures used in these kinds of domains are the **micromort** (a one in a million chance of death) and the **QALY**, a quality-adjusted life year (equivalent to a year in good health with no infirmities).

The knowledge engineer must also determine whether the overall utility function consists of different value attributes which combine in an additive way.

Q: *"Are there different attributes that contribute to an overall utility?"*
Modeling: add one utility node for each attribute.

Finally, the decision and utility nodes must be linked into the graph structure. This involves considering the *causal* effects of decisions/actions, and the *temporal* aspects represented by information links and precedence links. The following questions probe these aspects.

Q: *"Which variables can decision/actions affect?"*
Modeling: add an arc from the action node to the chance node for that variable.

Q: *"Does the action/decision itself affect the utility?"*
Modeling: add an arc from the action node to the utility node.

Q: *"What are the outcome variables that there are preferences about?"*
Modeling: add arcs from those outcome nodes to the utility node.

Q: *"What information must be available before a particular decision can be made?"*
OR **Q:** *"Will the decision be contingent on particular information?"*
Modeling: add information arcs from those observation nodes to the decision node.

Q: *"(When there are multiple decisions) must decision D1 be taken before decision D2?"*
Modeling: add precedence arcs from decision node D1 to decision node D2.

© 2004 by Chapman & Hall/CRC Press LLC

Missing car example

Let's work through this for the missing car example from the previous section. Recall that the original description concluded with *"Should they [the Nguyens] be worried about their second car being stolen and notify the police?"* They have to *decide* whether to notify the police, so *NotifyPolice* becomes the decision node. In this problem, acting on this decision doesn't affect any of the state variables; it might lead to the Nyugens obtaining more information about their car, but it will not change whether it has been stolen or borrowed. So there is no arc from the decision node to any of the chance nodes. What about their preferences, which must be reflected in the connections to the utility node?

Additional information about Nguyen's preferences

If the Nguyen's car is stolen, they want to notify the police as soon as possible to increase the chance that it is found undamaged. However, being civic-minded citizens, they don't want to waste the time of the local police if Sam has borrowed it.

This preference information tells us that utility depends on both the decision node and the *CarStatus* node, shown in Figure 9.17 (using the version in Figure 9.16(c)). This means there are four combinations of situations for which preferences must be elicited. The problem description suggests the utility ordering, if not the exact numbers, shown in Table 9.4[‖]. The decision computed by this network as evidence is added sequentially, is shown in Table 9.5. We can see that finding the car missing isn't enough to make the Nguyen's call the police. Finding signs of forced entry changes their decision; however more information — finding out that Sam is out — once again changes their decision. With all possible information now available, they decide not to inform the police.

TABLE 9.4
Utility table for the missing car problem

CarStatus	*Inform Police*	Outcome utility	
		Qualitative	Quantitative
borrowed	*no*	neutral	0
home	*no*	neutral	0
borrowed	*yes*	poor	-5
home	*yes*	poor	-5
stolen	*yes*	bad	-40
stolen	*no*	terrible	-80

[‖] Note that the utilities incorporate unmodeled probabilistic reasoning about getting the car back.

© 2004 by Chapman & Hall/CRC Press LLC

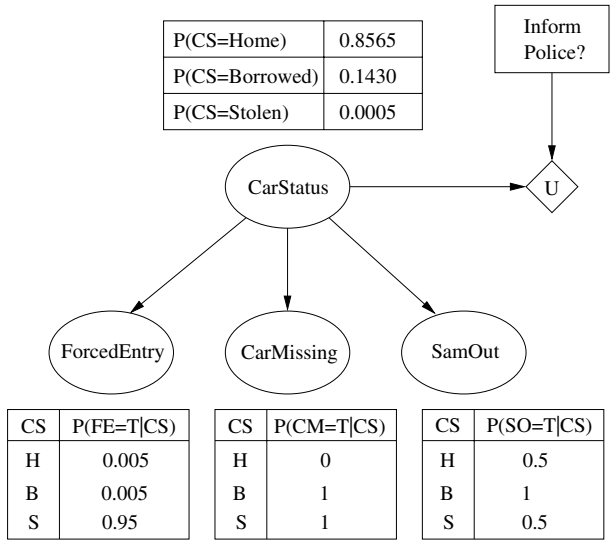


FIGURE 9.17
A decision network for the missing car problem.

TABLE 9.5
Expected utilities of *InformPolice* decision node for the missing car problem as evidence is added sequentially

Evidence	EU(*InformPolice*=Y)	EU(*InformPolice*=N)	Decision
None	-20.01	**-0.04**	No
CarMissing=T	-20.07	**-0.28**	No
ForcedEntry=T	**-27.98**	-31.93	Yes
SamOut=F	-24.99	**-19.95**	No

9.4 Adaptation

Adaptation in Bayesian networks means using some machine learning procedure to modify either the network's structure or its parameters over time, as new data arrives. It can be applied either to networks which have been elicited from human domain experts or to networks learned from some initial data. The motivation for applying adaptation is, of course, that there is some uncertainty about whether or not the Bayesian network is correct. The source of that uncertainty may be, for example, the expert's lack of confidence, or perhaps a lack of confidence in the modeled process itself being stable over time. In any case, part of the discipline of KEBN is the ongoing collection of statistics; so the opportunity of improving deployed networks by adaptation ought to be available.

We have already seen elements of adaptation. In Chapter 7 we saw how the Multi-

© 2004 by Chapman & Hall/CRC Press LLC

nomial Parameterization Algorithm 7.1 requires the specification of priors for the parameters, including an equivalent sample size which implicitly records a degree of confidence in those parameters. Also, §8.6.3 shows how prior probabilities for arc structure can be fed into the learning process. And, indeed, this section relies upon an understanding of those sections. The difference is one of intent and degree: in those chapters we were concerned with specifying some constraints on the learning of a causal model; here we are concerned with modifying a model already learned (or elicited).

9.4.1 Adapting parameters

We describe how to modify multinomial parameters given new complete joint observations. This can be done using the Spiegelhalter-Lauritzen Algorithm 7.1 under the same global and local assumptions about parameter independence. Recall that in that process the prior probabilities for each parameter are set with a particular equivalent sample size. If the parameters have actually been learned with some initial data set, then subsequent data can simply be applied using the same algorithm, starting with the new equivalent sample size and Dirichlet parameters set at whatever the initial training has left. If the parameters have been elicited, then you must somehow estimate your, or the expert's, degree of confidence in them. This is then expressed through the equivalent sample size, the larger the sample size the greater the confidence in the parameter estimates and the slower the change through adapting them with new data.

Suppose for a particular Dirichlet parameter α_i the expert is willing to say she gives a 90% chance to the corresponding probability $p_i = 0.2$ lying within the interval $[.1, .3]$. The expected value (i.e., the mean) of the Dirichlet distribution over state i corresponds to the estimated parameter value:

$$\mu_i = \frac{\alpha_i}{s} = 0.2$$

where $s = \sum_j \alpha_j$ is the equivalent sample size. The variance over state i of the Dirichlet distribution is:

$$\sigma_i^2 = \frac{\alpha_i(s - \alpha_i)}{s^2(s + 1)} = \frac{\mu_i(1 - \mu_i)}{s + 1}$$

The 90% confidence interval of this distribution lies within 1.645 standard deviations of the mean and is

$$\mu_i \pm \sigma_i = \mu_i \pm 1.645\sqrt{\frac{\mu_i(1 - \mu_i)}{s + 1}}$$

We can solve this algebraically for values of α_i and s that yield the mean and 90% confidence interval. A solution is an equivalent sample size $s = 15$ and a parameter value α_i of 3.

Of course, when this procedure is applied independently to the different Dirichlet parameters for a single parent instantiation the results may not be fully consistent. If

© 2004 by Chapman & Hall/CRC Press LLC

the expert reports the above interval for the first state of the binary child variable and $[0.4, 0.6]$ for the second, then the latter will lead to an equivalent sample size of $s = 24$ and $\alpha_2 = 12$. Since the equivalent sample size applies to all of the values of the child variable for a single instantiation, it cannot be both 15 and 24! The sample size must express a common degree of confidence across all of the parameter estimates for a single parent instantiation. So, the plausible approach is to compromise, for example, taking an average of the equivalent sample sizes, and then finding numbers as close to the estimated means for each state as possible. Suppose in this case, for example, we decide to compromise with an equivalent sample size of 20. Then the original probabilities for the two states, 0.2 and 0.5, yield $\alpha = (4, 10)$, which does not work. Normalizing (with round off) would yield instead $\alpha = (6, 14)$.

When parameters with confidence intervals are estimated in this fashion, and are not initially consistent, it is of course best to review the results with the expert(s) concerned.

Fractional updating is what Spiegelhalter and Lauritzen[263] call their technique for adapting parameters when the sample case is missing values, i.e., for incomplete data. The idea is simply to use the Bayesian network as it exists, applying the values observed in the sample case and performing Bayesian propagation to get posterior distributions over the unobserved cases. The observed values are used to update the Dirichlet distributions for those nodes; that is, a 1 is added to the relevant state parameter for the observed variable. The posteriors are used to proportionally update those variables which were unobserved; that is, $p' < 1$ is added to the state parameter corresponding to a value which takes the posterior p'. The procedure is complicated by the fact that a unique parent instantiation may not have been observed, when the proportional updating should be applied across all the possible parent instantiations, weighted by their posterior probabilities. This procedure unfortunately has the drawback of overweighting the equivalent sample size, resulting in an artificially high confidence in the probability estimates relative to new data.

Fading refers to using a time decay factor to underweight older data exponentially compared to more recent data. If we fade the contribution of the initial sample to determining parameters, then after sufficient time the parameters will reflect only what has been seen recently, allowing the adaptation process to track a changing process. A straightforward method for doing this involves a minor adjustment to the update process of Algorithm 7.1 [128, pp. 89-90]: when state i is observed, instead of simply adding 1 to the count for that state, moving from α_i to $\alpha_i + 1$, you first discount all of the counts by a multiplicative decay factor $d \in [0, 1]$. In other words the new Dirichlet distribution becomes $D[d\alpha_1, \ldots, d\alpha_i + 1, \ldots, d\alpha_\tau]$. In the limit, the Dirichlet parameters sum to $\frac{1}{1-d}$, which is called the **effective sample size**.

9.4.2 Structural adaptation

Conceivably, rather than just modifying parameters for an existing structure, as new information comes to light we might want to add, delete or reverse arcs as well. Jensen reports that "no handy method for incremental adaptation of structure has been constructed" [128]. He suggests the crude, but workable, approach of accumu-

© 2004 by Chapman & Hall/CRC Press LLC

lating cases and rerunning structure learning algorithms in batch mode periodically.

In addition to that, the Bayesian approach allows for structural adaptation, at least in principle. If we let what has previously been learned be reflected in our prior probabilities over structures, then new data will update our probability distributions over causal structure. That simply is Bayesian theory. To be sure, it is also Bayesian updating without model selection, since we need to maintain a probability distribution over all of the causal structures that we care to continue to entertain. A straightforward implementation of such inference will almost immediately be overwhelmed by computational intractability.

However, there is a simpler technique available with CaMML for structural adaptation. Since CaMML reports its estimate of the probability of each directed arc at the end of its sampling phase, and since CaMML allows the specification of a prior probability for such arcs before initiating causal discovery, those probabilities can be reused as the prior arc probabilities when performing learning with new data. This is crude in that there is no concept of equivalent sample size operating here; so there is no way to give greater weight to the initial data set over the latter, or vice versa. Nevertheless, it can be an effective way of adapting previously acquired structure.

9.5 Summary

There is an interplay between elements of the KEBN process: variable choice, graph structure and parameters. Both prototyping and the spiral process model support this interplay. Various BN structures are available to compactly and accurately represent certain types of domain features. While no standard knowledge engineering process exists as yet, we have sketched a framework and described in more detail methods to support at least some of the KE tasks. The integration of expert elicitation and automated methods, in particular, is still in early stages. There are few existing tools for supporting the KEBN process, although some are being developed in the research community, including our own development of VE and Matilda.

9.6 Bibliographic notes

An excellent start on making sense of software engineering is Brooks' *Mythical Man-Month* [33]. Sommerville's text is also worth a read [261]. You may also wish to consult a reference on software quality assurance in particular, which includes detailed advice on different kinds of testing [93].

Laskey and Mahoney also adapt the ideas of prototyping and the spiral model to the application of Bayesian networks [166]. Bruce Marcot [182] has described a

© 2004 by Chapman & Hall/CRC Press LLC

KEBN process for the application area of species-environment relations.

One introduction to parametric distributions is that of Balakrishnan and Nevzorov [14]. Mosteller et al.'s *Statistics by Example* [196] is an interesting and readily accessible review of common applications of different probability distributions. An excellent review of issues in elicitation can be found in Morgan and Henrion's *Uncertainty* [194], including the psychology of probability judgments.

9.7 Problems

Problem 1

The elicitation method of partitioning to deal with local structure is said in the text to directly correspond to the use of classification trees and graphs in Chapter 7. Illustrate this for the *Flu*, *Measles*, *Age*, *Fever* example by filling in the partitioned CPT with hypothetical numbers. Then build the corresponding classification tree. What do partition elements correspond to in the classification tree?

Problem 2

Consider the BN that you constructed for your own domain in Problem 5, Chapter 2. (Or if you haven't completed this problem, use an example network from elsewhere in this text, or one that comes with a BN software package.)

1. Identify the type of each node: target/query, evidence/observation, context, controllable.
2. Re-assess your choice of nodes and values, particularly if you discretized a continuous variable, in the light of the discussion in §9.3.1.
3. Label each arc with the relationship that is being modeled, using the relationships described in §9.3.2 or others you might think of.
4. Use Matilda to investigate the dependence and independence relationships in your network.
5. Can you map the probabilities in the CPTs into a qualitative scale, as in §9.3.3.2?

Problem 3

Using some of methods from this chapter, reengineer one of the BN applications you developed for one of the problems from Chapter 5. That is, redo your application, but employ techniques from this chapter. Write up what you have done, with reference to specific techniques used. Also, describe the difference use of these techniques has made in the final BN.

© 2004 by Chapman & Hall/CRC Press LLC

10

Evaluation

10.1 Introduction

Critical to developing Bayesian networks is evaluative feedback. One of the major advantages of the spiral, prototyping development of software is that from the very beginning a workable (if limited) user interface is available, so that end-users can get involved in experimenting with the software and providing ideas for improvement. This is just as true with the knowledge engineering of Bayesian networks, at least when a GUI front-end is available.

In this chapter we discuss some of the formal and semi-formal methods of generating evaluative feedback on Bayesian models. The first three types of evaluation are largely concerned with obtaining feedback from human experts, using an elicitation review process, sensitivity analysis and the evaluation of cases. The last section introduces a number of formal metrics for evaluating Bayesian models using statistical data.

10.2 Elicitation review

An elicitation review is a structured review of the major elements of the elicitation process. It allows the knowledge engineer and domain expert to take a global view of the work done and to check for consistency. This is especially important when working with multiple experts or combining expert elicitation with automated learning methods.

First, the variable and value definitions should be reviewed, encompassing:

1. A clarity test: do all the variables and their values have a clear operational meaning — i.e., are there definite criteria for when a variable takes each of its values? "Temperature is high" is an example that does not pass the clarity test*; this might be refined to "Temperature $\geq$ 30 degrees" which does.
2. Agreement on variable definitions:
    - Are all the relevant variables included? Are they named usefully?

*We have been guilty of exactly this in §3.2.

© 2004 by Chapman & Hall/CRC Press LLC

- Are all states (values) appropriate? Exhaustive and exclusive?
- Are all state values useful, or can some be combined?
- Are state values appropriately named?
- Where variables have been discretized, are the ranges appropriate?

3. Consistency checking of states: are state spaces consistent across different variables? For example, it might cause misunderstanding if a parent variable takes one of the values {*veryhigh, high, medium, low*} and its child takes one of {*extremelyhigh, high, medium, low*}.

The graph structure should be reviewed, looking at the implications of the d-separation dependencies and independencies and at whether the structure violates any prior knowledge about time and causality. A review of local model structure based on partitions or classification trees may also be conducted.

Reviewing the probabilities themselves can involve: comparing elicited values with available statistics; comparing values across different domain experts and seeking explanation for discrepancies; double-checking cases where probabilities are extreme (i.e., at or close to 0 or 1).

It is often useful to do a **model walk-through**, where a completed version of the model is presented to domain experts who have not been involved in the modeling process to date. All components of the model are evaluated. This can be done by preparing a set of cases with full coverage of the BN — i.e., sets of assumed observations with various sets of query nodes, when the reasoning required exercises every part of the network. This performs a similar function to code reviews in the software engineering process.

10.3 Sensitivity analysis

Another kind of evaluation is to analyze how sensitive the network is to changes in parameters or inputs; this is called **sensitivity analysis**. The network outputs may be either the posterior probabilities of the query node(s) or (if we are building a decision network) the choice of action. The changes to be tested may be variations in the evidence provided, or variations in the network parameters — specifically conditional probability tables or utilities. We will look at each of these types of changes in turn.

10.3.1 Sensitivity to evidence

Earlier in §9.3.2.2, we saw how the properties of d-separation can be used to determine whether or not evidence about one variable may influence belief in a query variable. It is also possible to measure this influence. Given a metric for changes in

© 2004 by Chapman & Hall/CRC Press LLC

belief in the query node (which we address just below), we can, for example, rank evidence nodes for either the maximum such effect (depending on which value the evidence node takes) or the average such effect. It is also possible, of course, to rank sets of evidence nodes in the same way, although the number of such sets is exponential in the number of evidence nodes being considered. In either case, this kind of sensitivity analysis provides guidance for the collection of further evidence. An obvious application area for this is medical diagnosis, where there may be multiple tests available; the clinician may like to perform the test that most decreases the uncertainty of the diagnosis.

So, how to quantify this uncertainty? What metric of change should we employ? We would like to drive query node probabilities close to 0 and 1, representing greater certainty. **Entropy** is the common measure of how much uncertainty is represented in a probability mass. The entropy of a distribution over variable X is (cf. Definition 8.2):

$$H(X) = -\sum_{x \in X} P(x) \log P(x) \tag{10.1}$$

For continuous variables, we can simply substitute integration for summation. The goal, clearly, is to minimize entropy: in the limit, entropy is zero if the probability mass is concentrated on a single value.

A second measure used sometimes is **variance**:

$$Var(X) = \sum_{x \in X} P(x - \mu)^2 P(x) \tag{10.2}$$

where μ is the mean, i.e., $\sum_{x \in X} xP(x)$. Variance is the classic measure of the dispersion of X around its mean. The greater the dispersion, the less is known; hence, we again aim to reduce this measure to zero.

TABLE 10.1
Entropy and variance measures for three distributions
(H is computed with natural logs)

P(Z=1)	P(Z=2)	P(Z=3)	P(Z=4)	H(Z)	Var(Z)
1	0	0	0	0.0	0.0
0.25	0.25	0.25	0.25	1.39	1.25
0	0.5	0.5	0	0.69	0.25

Whether using entropy or variance, the metric will actually be computed for the query node *conditional upon* whatever evidence nodes are being tested for sensitivity — that is, for the distribution $P(X|\mathbf{E})$ rather than $P(X)$. To provide a feel for these measures, Table 10.1 compares the entropy and variance for a few distributions over a quaternary variable Z.

The main BN software packages provide a facility to quantify the effect of evidence using some such measure (see §B.4).

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 10.2
Output from Netica's sensitivity to findings function for the Cancer Example, with $Cancer$ as the query node

Node	value of C	min Bel(C)	max Bel(C)	Entropy Reduction	(%)
No evidence. Bel(C=T)=0.012					
C	T	0	1	0.0914	100%
	F	0	1		
X	T	0.001	0.050	0.0189	20.7%
	F	0.950	0.999		
S	T	0.003	0.032	0.0101	11.0%
	F	0.968	0.997		
D	T	0.006	0.025	0.0043	4.7%
	F	0.975	0.994		
P	T	0.001	0.029	0.0016	1.7%
	F	0.971	0.990		
Evidence $X = Pos$. Bel(C=T)=0.050					
C	T	0	1	0.2876	100%
S	T	0.013	0.130	0.0417	14.5%
D	T	0.026	0.103	0.0178	6.2%
P	T	0.042	0.119	0.0064	2.2%
Evidence $X = Pos$,$S = T$. Bel(C=T)=0.129					
C	T	0	1	0.5561	100%
D	T	0.069	0.244	0.0417	7.5%
P	T	0.122	0.3391	0.0026	0.47%
Evidence $X = Pos$,$S = T$,$D = T$. Bel(C=T)=0.244					
C	T	0	1	0.8012	100%
P	T	0.232	0.429	0.0042	0.53%

Lung cancer example

Let us re-visit our lung cancer example from Figure 2.1 to see what these measures can tell us. $Cancer$ is our query node, and we would like to know observations of which single node will most reduce uncertainty about the cancer diagnosis. Table 10.2 shows the output from the Netica software's "sensitivity to findings" function, which provides this information. Each pair of rows tells us, if evidence were obtained for the node in column 1:

- The minimum (column 3) and maximum (column 4) posterior probabilities for $C = T$ or $C = F$ (column 2); and
- The reduction in entropy , both in absolute terms (column 5) and as a percentage (column 6).

The first set of results assumes no evidence has been acquired, with the prior $Bel(C = T) = 0.012$. As we would expect, all the "min" beliefs are less than the current belief and "max" beliefs above. Obtaining evidence for C itself is a degenerate case, with the belief changing to either 0 or 1. Of the other nodes, X

© 2004 by Chapman & Hall/CRC Press LLC

(*XRay*) has the most impact on C, then S, D and P respectively. Netica's output does not tell us which particular observed value actually gives rise to the minimum and maximum new belief; however, this is easy to work out.

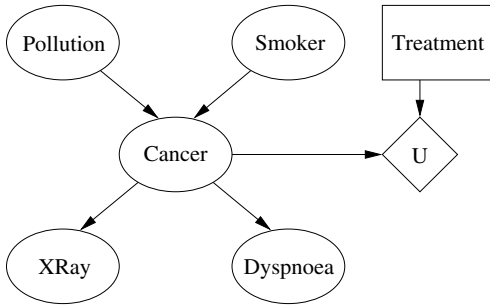
Suppose that the medical practitioner orders the test having the most potential impact, namely an X-ray, which then returns a positive result. The second set of results in Table 10.2 shows the new sensitivity to findings results (with values for $C = F$ omitted). In this example the relative ordering of the remaining nodes is the same — S,D and P — but in general this need not be the case.

These results only consider observations one at a time. If we are interested in the effect of observations of multiple nodes, the results can be very different. To compute the entropy reduction of a pair of observations each possible combination of values of both nodes must be entered and the entropy reductions computed. Netica does not provide for this explicitly in its GUI, but it is easily programmed using the Netica API.

Sensitivity of decisions

In general, rather than estimating raw belief changes in query nodes, we are more interested in what evidence may *change* a decision. In particular, if an observation *cannot* change the prospective decision, then it is an observation not worth making — at least, not in isolation.

In medical diagnosis, the decision at issue may be whether to administer treatment. If the BN output is the probability of a particular disease, then a simple, but ad hoc, approach is to deem treatment appropriate when that probability passes some threshold. Suppose for our cancer example that threshold is 50%. Then any one observation would not trigger treatment. Indeed, after observing a positive X-ray result and learning that the patient is a smoker, the sensitivity to findings tells us that evidence about shortness of breath still won't trigger treatment (since max(Bel(C=T)) is 0.244).



Cancer	Treatment	U
F	Y	−30
F	N	0
T	Y	−50
T	N	−250

FIGURE 10.1
A decision network for the cancer example.

© 2004 by Chapman & Hall/CRC Press LLC

This whole approach, however, is ad hoc. We are relying on an intuitive (or, at any rate, unexplicated) judgment about what probability of cancer is worth acting upon. What we really want is to decide for or against treatment based upon a consideration of the expected value of treatment. The obvious solution is to move to decision networks, where we can work out the expected utility of both observations and treatments. This is exactly the evaluation of "test-act" combinations that we looked at in §4.4, involving the **value of information**.

Figure 10.1 shows a decision network for the extended treatment problem. The effect of different observations on the expected utilities for $Treatment$ are shown in Table 10.3. We only consider the observations that can increase the belief in cancer and hence may change the treatment decision. We can see that evidence about any single node or pair of nodes will not change the decision, because in all cases the expected utility for not treating remains greater than the expected utility for treating the patient. However, once we consider the combinations of three evidence nodes, the decision can change when *X-Ray* = *pos*.

TABLE 10.3
Expected utilities for $Treatment$ in the extended lung cancer problem (highest expected utility in **bold**)

X=pos	S=T	D=T	P=high	EU(Treat=Y)	EU(Treat=N)
No evidence				-30.23	**-2.908**
•				-31.01	**-12.57**
	•			-30.64	**-8.00**
		•		-30.50	**-6.22**
			•	-30.58	**-7.25**
•	•			-32.59	**-32.37**
	•	•		-31.34	**-16.71**
		•	•	-31.22	**-15.19**
•		•		-32.06	**-25.73**
	•		•	-31.00	**-12.50**
•			•	-32.37	**-29.62**
•	•	•		**-34.88**	-60.94
•	•		•	**-33.83**	-47.87
•		•	•	**-34.51**	-56.38
	•	•	•	-32.05	**-25.59**
•	•	•	•	**-36.78**	-84.78

The work we have done thus far is insufficient, however. While we have now taken into account the expected utilities of treatment and no treatment, we have not taken into account the costs of the various observations and tests. This cost may be purely monetary; more generally, it may combine monetary costs with non-financial burdens, such as inconvenience, trauma or intrusiveness, as well as some probability of additional adverse medical outcomes, such as infections caused by examinations or biopsies.

© 2004 by Chapman & Hall/CRC Press LLC

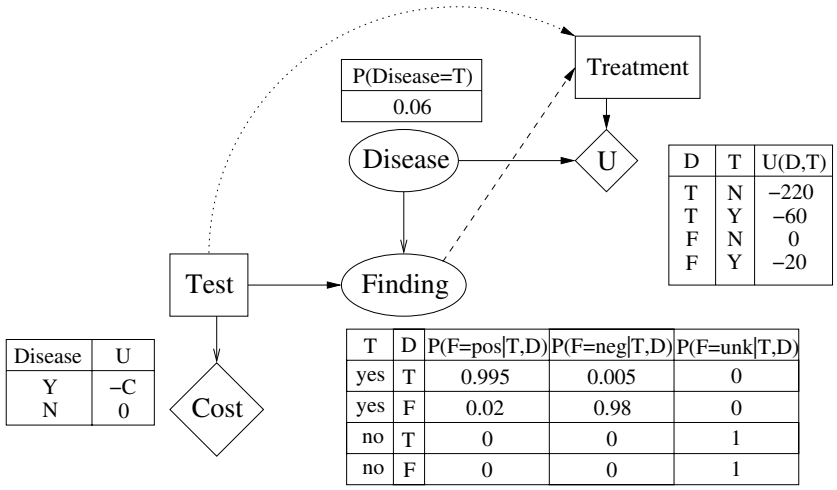


FIGURE 10.2
A decision network for disease treatment.

Let us work through such reasoning with the simple, generic decision network shown of Figure 10.2. Here there is a single disease node and a single test findings node. The utility is a function of the disease and the treatment decision. We would also like to take into account the option for running the test, represented by another decision node, which also has an associated cost, C. (A cost of $C = 5$ is used for the calculations in the remainder of this section.)

If we do not run the test, the probability of the disease is just the prior (0.30) and the expected utilities of the treatment options are:

$$\begin{aligned} EU(Treatment = Y) &= P(Disease = T) \times U(T = Y, D = T) \\ &\quad + P(Disease = F) \times U(T = Y, D = F) \\ &= 0.06 \times -60 + 0.94 \times -20 = -22.4 \\ EU(Treatment = N) &= P(Disease = T) \times U(T = N, D = T) \\ &\quad + P(Disease = F) \times U(T = N, D = F) \\ &= 0.06 \times -220 + 0.94 \times 0 = -13.2 \end{aligned}$$

So, without a test, the decision would be *not* to have treatment.

© 2004 by Chapman & Hall/CRC Press LLC

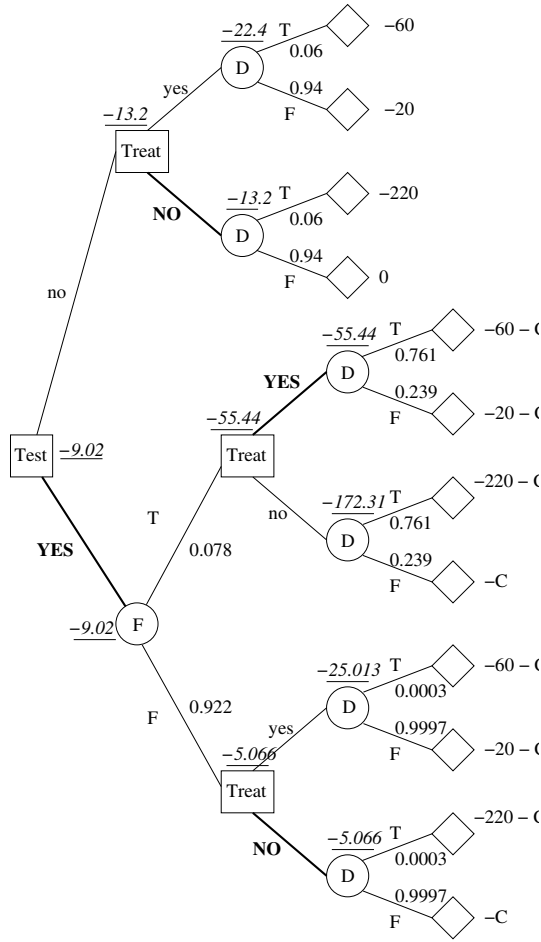


FIGURE 10.3
A decision tree evaluation of the disease test-treatment decision network.

If, on the other hand, we run the test and get a value for the findings (positive or negative), we obtain the expected utilities of Table 10.4. The decision tree evaluation for this problem is shown in Figure 10.3, using the method described in §4.4.3.

Hence the **expected benefit** of running the test, the *increase* in our expected utility, is the difference between the expected utility of running the test and the expected utility of our decision without the test, namely: $EU(Test = Y) - EU(Treatment = N)$ = -9.02 − (-13.20) = 4.18.

What if there are several tests that can be undertaken? In order to determine the best course of action, ideally we would look at all possible combinations of tests, including doing none. This will clearly often be impractical. In such cases, a greedy search may be used, where at each step, the single test with the highest expected

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 10.4
Expected utilities given different test findings (highest EU given in **bold**, indicating decision table entry)

EU	Treatment	
Finding	Y	N
positive	**-55.42**	-172.31
negative	-25.01	**-5.07**

benefit is chosen (if this is greater than no test). The price of practicality is that greedy search can result in a non-optimal sequence, if the outcome of a rejected test would have made another more informative.

10.3.2 Sensitivity to changes in parameters

Sensitivity analysis can also be applied to the parameters, checking them for correctness and whether more precision in estimating them would be useful. Most such sensitivity analyses are one-dimensional, that is, they only vary one parameter at a time. For many decision problems, decisions will be unaffected by the precision of either the model or the input numbers. This phenomenon is known as a **flat maximum**, as it marks the point beyond which small changes in probabilities or utilities are unlikely to affect the final decision [59]. Once a flat maximum is observed, no more effort should be spent refining the model or its inputs. However, even when a network may be insensitive to a change in one of its parameters, it may well still be sensitive to changes in combinations of parameters. As always, testing all possible combinations of parameters is exponentially complex.

It is plausible that the elicitation process can be supported by interactively performing sensitivity analyses of the network during construction, starting with initial rough assessments [57]. A sensitivity analysis indicates which probabilities require a high level of accuracy and which do not, providing a focus for subsequent elicitation efforts.

One can approach sensitivity analysis of BNs either empirically (e.g., [146, 56]) or theoretically (e.g., [167, 150]). The empirical approach examines the effects of varying the parameters on a query node. It requires first identifying reasonable ranges for each parameter, then varying each parameter from its lowest to its highest value while holding all the other variables fixed. The changes in the posterior probability of the query node may be quantified with a measure such as entropy (see above). Or when we are concerned with decision making, we can identify at what point (if any) changes in a parameter can change the decision.

The theoretical approach instead determines a function expressing the posterior probability of the query node in terms of the parameters. For example, Laskey [167] computes a partial derivative of the query node with respect to each parameter. In addition to not requiring additional assessment by the expert, often this function can be computed efficiently by a modification of the standard inference algorithms.

The two approaches can be used together, by first having a theoretical approach

© 2004 by Chapman & Hall/CRC Press LLC

identify parameters with potentially high impact, then reasonable ranges can be assessed for these, allowing one to compute the sensitivity empirically.

For some time there was a belief common in the BN community that BNs are generally insensitive to inaccuracies in the numeric value of their probabilities. So the standard complaint about BNs as a method that required the specification of precise probabilities ("where will the numbers come from?") was met with the rejoinder that "the numbers don't really matter." This belief was based on a series of experiments described in [221], which was further elaborated on in [107]. Clearly, however, the inference that insensitivity is general is wrong, since it is easy to construct networks that are highly sensitive to parameter values (as one of the problems in §10.9 shows). Indeed, more recent research has located some real networks that are highly sensitive to imprecision in particular parameters [146, 279, 41]. The question of how to identify such sensitive parameters, clearly an important one from a knowledge engineering perspective, is a current research topic.

There is as yet only limited support in current BN software tools for sensitivity analysis; see §B.4.

Disease treatment example

Before we leave this topic, let us briefly look at an example of how a decision may be sensitive to changes in parameters, using the generic disease "test-treatment" sequential decision network from Figure 10.2. Figure 10.4 shows how the expected utilities for the test decision (y-axis) vary as we change a parameter (x-axis), for three different parameters.

(a) The test cost, C: While EU(Test=N) does not change, we can see that the test decision will change around the value of C=9.

(b) U(Disease=F,Treatment=Y): In this case, the change in decision from "no" to "yes" occurs between -5 and -6.

(c) The prior, P(Disease=T): Here, the decision changes between 0.03 and 0.04.

10.4 Case-based evaluation

Case-based evaluation runs the BN model on a set of test cases, evaluating the results via expert judgment or some kind of prior knowledge of the domain. Test cases can be used either for **component testing**, where fragments of the BN are checked, or for **whole-model testing**, where the behavior of the full BN is tested. The results of interest for the test will depend on the domain, sometimes being the decision, or a diagnosis, or again a future prediction. Ideally, cases would be generated to test a wide variety of situations, possibly using the **boundary-value analysis** technique of software engineering: choosing inputs at the extreme upper and lower values of

© 2004 by Chapman & Hall/CRC Press LLC

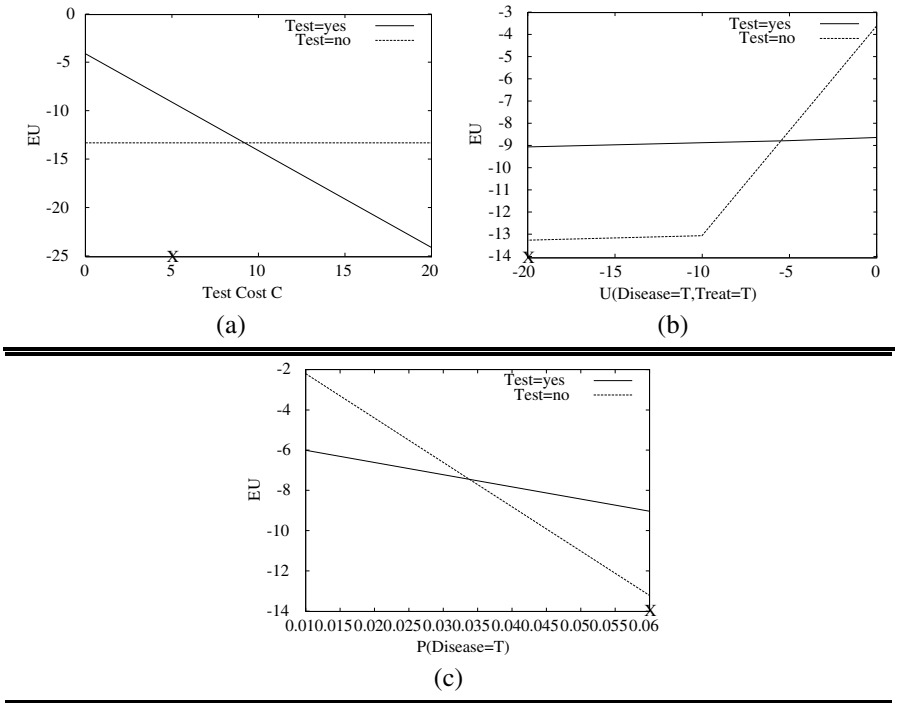


FIGURE 10.4
Sensitivity analysis showing effect of varying a parameter on the test decision, for three parameters: (a) Cost of test, C; (b) U(Disease=T,Treatment=yes); (c) P(Disease=T). Parameter value in model marked with X on the x-axis.

available ranges to ensure reasonable network performance across the largest possible set of conditions. The usability of this approach will depend upon the experts having enough experience to make useful judgments in all such cases.

Case-base evaluation tends to be informal, checking that expert opinion is being properly represented in the network. If a great many cases can be brought together, then more formal statistical methods can be applied. These are discussed immediately below, as validation methods.

10.4.1 Explanation methods

If a BN application is to be successfully deployed and used, it must be accepted first by the domain experts and then by the users[†]. As we have already discussed (§5.5 and §9.3.2.2), the independence-dependence relations encoded in the BN structure are not always obvious to someone who is not experienced with BNs. Also, the

[†] Of course the domain experts may also be the intended users.

© 2004 by Chapman & Hall/CRC Press LLC

outcomes of probabilistic belief updating are often unintuitive. These factors have motivated the development of methods for the **explanation** of Bayesian networks.

Some of these methods focus on the explanation *of* the Bayesian network. Others provide explanations of the conclusions drawn about the domain *using* the Bayesian network — that is, how the inference has led from available evidence to the conclusions. This latter is how our NAG would like to be employed eventually (§5.5). The term "explanation" has also been used to describe the provision of the "most probable explanation"(MPE) (§3.7.1) of the evidence (e.g., [164]). We view that more as an inferential matter, however.

INSITE [271] was an early explanation system that identified the findings that influence a certain variable and the paths through which the influence flows. It then generates visual cues (shading and highlighting paths) and verbal explanations of the probabilistic influences. Explanation methods have been used for teaching purposes. For example, BANTER [97] is a BN tutoring shell, based on INSITE's methods, for tutoring users (such as medical students) in the evaluation of hypotheses and selection of optimal diagnostic procedures. In BANTER, the use of the BN by the system to perform its reasoning is completely invisible to the user. GRAPHICAL-BELIEF [178] is an explanatory system that provides a graphical visualization of how evidence propagates through a BN.

In addition to supporting a deployed system or *being* part of the application itself, such explanation methods can be useful during the iterative evaluation of the KEBN process. They have been shown to support the detection of possible errors in the model [97] and can be used by the domain expert to verify the output of the model [271, 178, 77].

10.5 Validation methods

Validating a Bayesian network means to confirm that it is an accurate representation of the domain or process being modeled. Much of what we have discussed above, concerning comparing the behavior of a network with the judgments of experts, could be considered a kind of validation. However, we will reserve the term for more statistically oriented evaluation methods and specifically for methods suited to testing the correctness of a Bayesian network when a reasonably large amount of data describing the modeled process is available.

First there is a general consideration. The Bayesian networks are (usually) built or learned under an explicitly causal interpretation. But much validation work in the literature concerns testing the probability distribution represented by the network against some **reference distribution**. By a reference distribution (or, reference model, etc.) we mean a known distribution which is being used as the source for sample data; typically, this is a Bayesian network which is being used to artificially generate sample data by simulation. Now, since Bayesian networks within a single Markov equivalence class can be parameterized so as to yield identical probability

© 2004 by Chapman & Hall/CRC Press LLC

distributions, testing their probability distributions against a reference distribution fails to show that you have the right causal model. For example, the Kullback-Leibler divergence (§3.6.5) from the true distribution to the learned distribution will not help distinguish between the different causal models within a single Markov equivalence class. There is no consensus view on how to evaluate learned causal models as distinct from learned probability distributions.

One way of dealing with this problem is to collect experimental data, rather than simply take joint observations across the variables. In that case, we can represent the causal interventions explicitly and distinguish between causal models. There are many reasons why this option may be unavailable, the main one being that collecting such experimental data may be prohibitively expensive or time consuming.

A more readily available option is to analyze the problem and identify a subset of nodes which are characteristically going to be the query nodes in application and another subset which are going to be evidence nodes. Sample data can then be used to see how well the learned network predicts the values of the query nodes when the evidence nodes take the values observed in the sample. This can be done whether the query nodes are answers to diagnostic queries (i.e., causes of the evidence nodes) or are causally downstream from the evidence nodes. As described so far, this evaluation also does not take into account the causal structure being learned. It will often turn out that the restricted subnetwork being examined has few Markov equivalent subnetworks even when the full network does. Alternatively, you can examine causal structure directly, penalizing errors (as we discussed in §8.7.1). In one of our studies, we tested learned models restricted to leaf nodes only against the reference model's leaf nodes; if our method mislearned causal structure it would misidentify leaves, thus leading to a higher reported error [201].

The validation methods we consider here evaluate Bayesian networks against real data, sampled from the real process to be modeled, rather than against data simulated from reference models. Much of the research literature, however, employs artificial data, sampled from a reference model. It is important to be clear about the difference in intent behind these two procedures. If we are testing one machine algorithm against another (e.g., K2 against CaMML), then sampling from an artificially constructed Bayesian network and then seeing whether K2 or CaMML can learn from the sample data something like the original network makes good sense. In that case we are testing *algorithms* against each other. But if we are interested in testing or validating a *model* for a real process, then we presumably do not know in advance what the true model is. If we did, we would already be done. So, the practical validation problem for KEBN is to test some models, constructed by hand or learned by our machine learning algorithms, against real data reporting the history of the process to be modeled. All of the methods below are applicable to evaluating against the real data, except for the Kullback-Leibler divergence (§3.6.5), which is used to motivate our information reward measure.

© 2004 by Chapman & Hall/CRC Press LLC

10.5.1 Predictive accuracy

Predictive accuracy is far and away the most popular technique for evaluating models, whether they are Bayesian networks, classification trees, or regression models. Predictive accuracy is assessed relative to a target variable by:

1. Examining each joint observation in the sample
2. Adding any available evidence for the other nodes (i.e., non-target nodes)
3. Updating the network
4. Taking the predicted value for target variable to be that which then has the highest probability
5. And comparing this with the actual value for that variable in the observation

For example, if a model attempting to relate measured attributes of mushrooms to their edibility reported for a particular mushroom that the probability of it being edible is 0.8, then it would be taken as predicting edibility. Running through the entire sample, we can record the frequency with which the model gets its prediction right. That is its **predictive accuracy**. We could equivalently report its error rate as 100% minus its predictive accuracy.

If the sample data have been used to learn the model in the first place (i.e., they are the training data) — whether learning the causal structure, the parameters or both — then they cannot safely be reused to gauge the model's predictive accuracy. In that case, the model is not being tested on its predictions, but just on how well it has been tuned to the training data. Fit to the training data can be very far from the ability of the model to *predict* new values. When a fixed sample must be used for both training and testing a model, a plausible approach is divide the data into separate training and test sets, for example, in a 90% and 10% split. The model's predictive accuracy on the test set will now provide an estimate of its **generalization accuracy**, how well it can predict genuinely new cases, rather than rewarding overfitting of the training data.

A single such test will only provide a crude idea of predictive accuracy. It amounts, in fact, to a sample of size one of the model's predictive accuracy and so provides no opportunity to judge the variation in the model's predictive accuracy over similar test sets in the future. If the training-test procedure can be repeated enough times, then each predictive accuracy measure will provide a separate sample, and the whole sample can be used to assess the variation in the model's performance. For example, it then becomes possible to perform an orthodox statistical significance test to assess one model's accuracy against another's. There may be no opportunity to collect enough data for such repeated testing. There may nevertheless be enough data for **k-fold cross validation**, which reuses the same data set, randomly generating 90-10 splits k times[‡].

[‡] Standard significance tests appear to be biased for k-fold cross validation and are better replaced by a special paired t-test; see Dietterich [75].

© 2004 by Chapman & Hall/CRC Press LLC

Predictive accuracy is relatively simple to measure, and for that reason may provide your best initial guide to the merit of a model. Unfortunately, predictive accuracy is not your best final guide to a model's merit: there are excellent reasons to prefer other measures. A fundamental problem is that predictive accuracy entirely disregards the confidence of the prediction. In the mushroom classification problem, for example, a prediction of edibility with a probability of 0.51 counts exactly the same as a prediction of edibility with a probability of 1.0. Now, if *we* were confronted with the first prediction, we might rationally hesitate to consume such a mushroom. The predictive accuracy measure does not hesitate. According to standard practice, any degree of confidence is as good as any other if it leads to the same prediction: that is, all predictions in the end are categorical, rather than probabilistic. Any business, or animal, which behaved this way would have a very short life span! In language we have previously introduced (see §9.3.3.1), predictive accuracy pays no attention to the calibration of the model's probabilities. So, we will now consider three alternative metrics which do reward calibration and penalize miscalibration.

10.5.2 Expected value

In recognition of this kind of problem, there is a growing movement in the machine learning community to employ **cost-sensitive classification** methods. Instead of preferring an algorithm or model which simply has the highest predictive accuracy, the idea is to prefer an algorithm or model with the best weighted average cost or benefit computed from its probabilistic predictions. In other words, the best model is that which produces the highest expected utility for the task at hand.

Since classifications are normally done with some purpose in mind, such as selecting a treatment for a disease, it should come as no surprise that utilities should be relevant to judging the predictive model. Indeed, it is clear that an evaluation which ignores utilities, such as predictive accuracy, cannot be optimal in the first place, since it will, for example, penalize a false negative of some nasty cancer no more and no less than a false positive, even if the consequences of the former swamp the latter.

Here is a simple binomial example of how we can use expected value to evaluate a model. Suppose that the utilities associated with categorically predicting $X = T$ or $X = F$ are as shown in Table 10.5. Then the model's score would be

- $P(x_i = T)u_1 + P(x_i = F)u_3$ across those cases where in fact $x_i = T$, and
- $P(x_i = T)u_2 + P(x_i = F)u_4$ otherwise.

Its average score would be the sum of these divided by the number of test cases, which is the best estimate of the expected utility of its predictions, if the model is forced to make a choice about the target class.

This expected value measurement explicitly takes into account the full probability distribution over the target class. If a model is overconfident, then for many cases it will, for example, attach a high probability to $x_i = T$ when the facts are otherwise, and it will be penalized with the negative utility u_2 multiplied with that high probability.

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 10.5
Utilities for binary classification and misclassification

	$x_i = T$	$x_i = F$
$x_i = T$ predicted	$u_1 > 0$	$u_2 < 0$
$x_i = F$ predicted	$u_3 < 0$	$u_4 > 0$

This approach to evaluating predictive models is, in fact, quite paradigmatically Bayesian. The typical Bayesian approach to prediction is not to nominate a highest probability value for a query variable given the evidence, but instead to report the posterior probability distribution over the query node. Indeed, not only do Bayesians commonly hesitate about nominating a predicted class, they even hesitate about nominating a specific model for classifying. That is, the ideal Bayesian approach is to compute a posterior distribution over the entire space of models, and then to combine the individual predictions of the models, weighting them according to their posteriors. This mixed prediction will typically provide better predictions than those of any of the individual models, including that one with the highest posterior probability. This is the hard-core Bayesian method, as advocated, for example, by Richard Jeffrey [123]. But in most situations this ideal method is unavailable, because it is computationally intractable. More typical of scientific practice, for example, is that we come to believe a particular scientific theory (model), which is then applied *individually* to make predictions or to guide interventions in related domains.

The expected value measurement is the best Bayesian guide to evaluating individual models; it is the Bayesian gold standard for predictive evaluation. In effect, it obliges the model to repeatedly gamble on the values of the target variable. The best model will set the optimal odds (probabilities) for the gamble, being neither underconfident nor overconfident. The underlying idea, from Frank Ramsey [231], is to take gambling as a useful metaphor for decision making under uncertainty.

As everything, this method comes at a price: in particular, we must be able to estimate the utilities of Table 10.5. For many applications this will be possible, by the various elicitation methods we have already discussed. But again, in many others the utilities may be difficult or impossible to estimate. This will especially be true when the model being developed has an unknown, unexplored, or open-ended range of potential applications. It may be, for example, that the industrial environment within which the model will be employed is rapidly evolving, so that the future utilities are unknown. Or, in the case of many scientific models, the areas of application are only vaguely known, so the costs of predictive error are only vaguely to be guessed at. So, we now turn to evaluative methods which share with predictive accuracy the characteristic of being independent of the utility structure of the problem. They nevertheless are sensitive to over- and underconfidence, rewarding the well-calibrated predictor.

10.5.3 Kullback-Leibler divergence

There are two fundamental ingredients to gambling success, and we would like our measure to be maximized when they are maximized:

© 2004 by Chapman & Hall/CRC Press LLC

Property 1: Domain knowledge (first-order knowledge), which can be measured by the frequency with which one is inclined to correctly assert $x_i = T$ or $x_i = F$ — i.e., by predictive accuracy. For example, in sports betting the more often you can identify the winning team, the better off you are.

Property 2: Calibration (meta-knowledge), the tendency of the bettor to put $P(x_i = T) = p$ close to the objective probability (or, actual frequency). That betting reward is maximized by perfect calibration is proven as Theorem 6.1.2 in Cover and Thomas's *Elements of Information Theory* [60][§].

With Property 1 comes a greater ability to predict target states; with Property 2 comes an improved ability to assess the probability that those predictions are in error. These two are not in a trade-off relationship: they can be jointly maximized.

If we happen to have in hand the true probability distribution over the target variables, then we can use Kullback-Leibler divergence (from §3.6.5) to measure how different the model's posterior distribution is from the true distribution. That is, we can use

$$KL(p, q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \tag{10.3}$$

This is just the expected log likelihood ratio:

$$KL(p, q) = E_p \log \frac{p(x)}{q(x)} \tag{10.4}$$

Kullback-Leibler divergence is a measure which has the two properties of betting reward in reverse: that is, by minimizing KL divergence you maximize betting reward, and vice versa. Consider, the KL divergence from the binomial distribution $B(p = 0.5)$ for various $B(q)$, as in Figure 10.5. Clearly, this KL divergence is minimized when $q = p$. This is proved for all positive distributions in Technical Notes §10.8. Of course, $q = p$ implies that the model is perfectly calibrated. But it also implies Property 1 above: clearly there is no more domain knowledge to be had (disregarding causal structure, at any rate) once you have exactly the true probability distribution in the model.

KL divergence is the right measure when there is a preferred "point of view" — when the true probability distribution is known, p in equation (10.3), then KL divergence reports how close a model's distribution q is to the generating distribution. This provides an evaluation measure for a model that increases monotonically as the model's distribution diverges from the truth. The drawback to KL divergence is simply that when assessing models for real-world processes, the true model is necessarily unknown. Were it known, we would have no need to collect observational data and measure KL divergence in the first place. In consequence, we want some

[§]This is recognized in David Lewis's "Principal Principle," which asserts that one's subjective probability for an event, conditioned upon the knowledge of a physical probability of that event, should equal the latter [171].

© 2004 by Chapman & Hall/CRC Press LLC

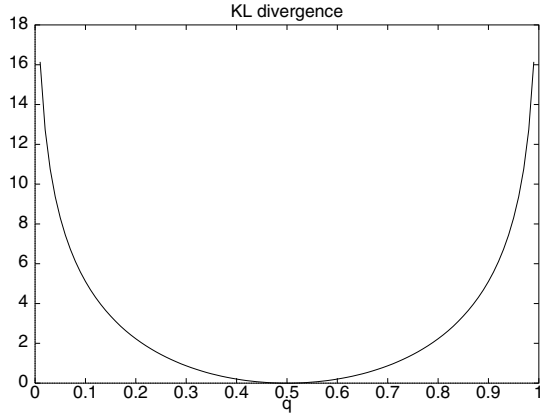


FIGURE 10.5
KL divergence.

other performance measure, sharing with KL the same two properties of being optimized with maximal domain knowledge and calibration, while not depending on some privileged access to the truth.

10.5.4 Information reward

Information reward, due to I.J. Good [95], is just such a measure. Good invented it as a cost neutral assessment of gambling expertise for binomial prediction tasks. Good's definition is:

$$IR_G = \sum_i [1 + \log_2 P(x_i = v)] \tag{10.5}$$

where v is whichever truth value is correct for the case of x_i. Good introduced the constant 1 in order to fix the reward in case of prior ignorance to zero, which he assumed would be represented by predicting $P(x_i = v) = 0.5$. This reward has a clear and useful information-theoretic interpretation (see Figure 10.6): it is one minus the length of a message conveying the actual event in a language efficient for agents with the model's beliefs (i.e., a language that minimizes entropy for them). Note that such a message is infinite if it is attempting to communicate an event that has been deemed to be impossible, that is, to have probability zero. Alternatively, message length is minimized, and information reward maximized, when the true class is predicted with maximum probability, namely 1. This gives us Property 1.

IR_G also has Property 2. Let c be the chance (physical probability, or its stand-in, frequency in the test set) that some test case is in the target class and p the probability estimated by the machine learner. Then the expected IR_G is:

$$[1 + \log_2 p]c + [1 + \log_2 (1 - p)](1 - c)$$

© 2004 by Chapman & Hall/CRC Press LLC

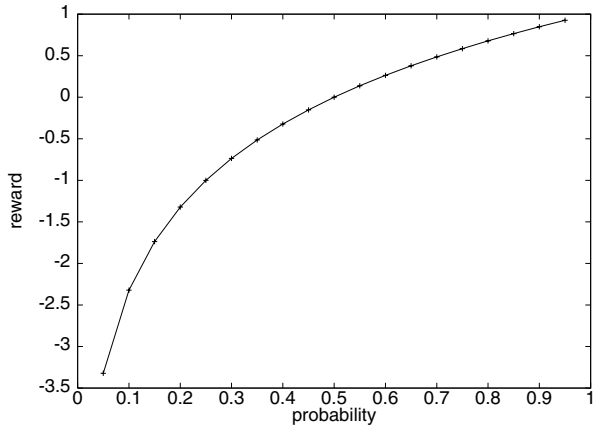


FIGURE 10.6
Good's information reward.

To find the maximum we take the derivative with respect to p and set it to zero:

$$\begin{aligned} 0 &= \frac{d}{dp}\left([1+\log_2 p]c + [1+\log_2(1-p)](1-c)\right) \\ &= \frac{c}{p} + \frac{1-c}{1-p}\frac{d}{dp}(1-p) \\ &= \frac{c}{p} - \frac{1-c}{1-p} \end{aligned}$$

So,

$$c = p$$

which last equation reports that perfect calibration maximizes reward.

All of this has some interesting general implications for machine learning research and evaluation methodology. For one thing, machine learning algorithms might be usefully modified to monitor their own miscalibration and use that negative feedback to improve their calibration performance while also building up their domain knowledge.

In experimental work, we have shown that standard machine learning problems issue differing verdicts as to which algorithm or model is best, when those verdicts are based on information reward instead of predictive accuracy [157]. In other words, reliance upon the predictive accuracy metric is demonstrably causing errors in the judgments of performance showing up in the machine learning literature.

10.5.5 Bayesian information reward

There are two respects in which Good's information reward is deficient, leading to potential errors in evaluating models. The first is that IR_G assumes a uniform prior probability over the target classes. If the base rate for a disease, for example, is far

© 2004 by Chapman & Hall/CRC Press LLC

lower than 50% in a population, say 1%, then a model which reflects no understanding of the domain at all — beyond that base rate — can accumulate quite a high information reward simply by predicting a 1% chance of disease for all future cases. Rather than Good's intended zero reward for ignorance, we have quite a high reward for ignorance. As a *Bayesian* reward function, we propose one which takes the prior into account and, in particular, which rewards with zero any prediction at the prior probability for the target class.

The second deficiency is simply that IR_G is limited to binomial predictions. Bayesian networks are very commonly applied to more complex prediction problems, so we propose the **Bayesian information reward** which applies to multinomial prediction:

$$IR_B = \frac{\sum_i I_i}{n} \tag{10.6}$$

where n is the number of test cases, $I_i = I_i^+$ for the true class and $I_i = I_i^-$ otherwise, and

$$I_i^+ = \log \frac{p_i'}{p_i} \quad \text{(true value)}$$
$$I_i^- = \log \frac{1 - p_i'}{1 - p_i} \quad \text{(false value)}$$

where p_i' is the model's probability for the variable taking the particular value at issue and p_i is the prior probability of the variable taking that value. Note that this version of information reward is directly related to Kullback-Leibler divergence (§3.6.5), where the prior probability takes the role of the reference probability, but with two differences. First, the weighting is done implicitly by the frequency with which the different values arise in the test set, presumably corresponding to the prior probability of the different values. Second, the ratio between the model probability and the prior is inverted, to reflect the idea that this is a *reward* for the model to *diverge* from the reference prior, rather than to converge on it — so long as that divergence is approaching the truth more closely than does the prior. This latter distinction, rewarding divergence toward the truth and penalizing divergence away from the truth, is enforced by the distinction between IR_B for true value and for false values.

This generalizes Good's information reward to cover multinomial prediction by introducing a reward value for values which the variable in the test case fails to take (i.e., I_i^-). In the case of the binomial IR_G this was handled implicitly, since the probability of the alternative to the true value is constrained to be $1 - p_i'$. However, for multinomials there is more than one alternative to the true value and they must be treated individually, as we do above. IR_B has the following meritorious properties:

- If a model predicts any value with that value's prior probability, then the IR_B is zero. That is, ignorance is never rewarded. This raises the question of where the priors for the test cases should come from. The simplest, and usually satisfactory, answer is to use the frequency of the value for the variable within the training cases.

© 2004 by Chapman & Hall/CRC Press LLC

- Property 1 is satisfied: as predictions become more extreme, close to one or zero, for values which are, respectively, true or false, the reward increases asymptotically to one. As incorrect predictions become more extreme, the negative reward increases without bound. Both of these are strictly in accord with the information-theoretic interpretation of the reward.

- Property 2 is satisfied: IR_B is maximized under perfect calibration (see §10.8).

10.6 Summary

Sensitivity analysis can be applied to model parameters (utilities and conditional probability tables) as well as inputs (observations) to determine the impact of large or small changes to decisions and/or query nodes. It is well suited to obtaining feedback from a domain expert and can be used throughout the prototyping development of Bayesian networks. Case-based evaluation is again well suited to obtaining expert judgment on how a particular set of cases is handled by the model. Explanation methods can be applied both to enhance user understanding of models and to support their initial development.

Validation methods are applicable when a reasonable set of observational samples are available for the domain. In evaluating Bayesian networks the causal structure needs to be taken into account explicitly, since the validation metrics do not do so of their own accord. A plausible approach is to examine the probability distribution over a subnetwork of query and evidence nodes. Four validation methods are:

- Predictive accuracy. This is the most commonly used and generally the easiest to compute. It can provide a biased assessment in that it fails to take into account the probabilistic nature of prediction; in particular, it fails to penalize miscalibration.

- Expected value, or cost-sensitive methods. This is the most relevant and effective evaluation metric for any particular domain, so long as the costs associated with misclassification and the benefits for correct classification are available.

- Kullback-Leibler divergence. This is an idealized metric, one which requires that the *true* distribution is already known. It illustrates the virtues which we would like a good metric to have.

- Bayesian information reward. This is a metric which has those virtues, but does not require prior knowledge of the truth. It is suboptimal in that it is entirely insensitive to the utilities of (mis)classification; however, it will deliver a generally useful assessment of the betting adequacy of the model for the domain.

© 2004 by Chapman & Hall/CRC Press LLC

10.7 Bibliographic notes

Coupé's Ph.D. thesis [55] provides a very thorough coverage of the issues involved in undertaking sensitivity analysis with respect to parameters, for both Bayesian networks and decision networks.

For an example of recent work on cost-sensitive classification, see papers by Turney [276] and Provost [224]; there was also some early work by Pearl [214]. Kononenko and Bratko also criticize predictive accuracy for evaluating algorithms and models [155]; furthermore, they too offer a scoring function which takes the prior probabilities of target classes into account. Unfortunately, their metric, while beginning with an information-theoretic intuition, does not have a proper information-theoretic interpretation. See our [110] for a criticism of the Kononenko-Bratko metric (however, please note that this paper's metric itself contains an error; IR_B in the text is a further improvement).

10.8 Technical notes

Kullback-Leibler divergence

We prove that KL divergence is minimized at zero if and only if the model under evaluation is perfectly calibrated (restricting our consideration to strictly positive distributions).

Theorem 10.1 *Let p and q of Equation 10.3 be strictly positive probability distributions. Then $KL(p, q) \geq 0$ and $KL(p, q) = 0$ if and only if $p = q$.*

Proof. (See [25, A.15.1].)
First, a lemma:

$$\sum_{x \in X} p(x) \log \frac{q(x)}{p(x)} \leq 0 \tag{10.7}$$

This is because (since $\forall x[\log x \leq x - 1]$)

$$\begin{aligned} \sum_{x \in X} p(x) \log \frac{q(x)}{p(x)} &\leq \sum_{x \in X} p(x) \left(\frac{q(x)}{p(x)} - 1 \right) \\ &= \sum_{x \in X} p(x) - \sum_{x \in X} q(x) \\ &= 1 - 1 \\ &= 0 \end{aligned}$$

Since,

$$\sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} = - \sum_{x \in X} p(x) \log \frac{q(x)}{p(x)}$$

© 2004 by Chapman & Hall/CRC Press LLC

it follows that

$$\sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \geq 0 \tag{10.8}$$

And if $\sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} = 0$, then by positivity of p, $\log \frac{p(x)}{q(x)}$ must everywhere be zero; hence $p(x) = q(x)$ everywhere.

Bayesian information reward

We provide a restricted proof that Bayesian information reward is maximized when the estimated probabilities are calibrated. The Bayesian information reward for a classification into classes $\{C_1, \ldots, C_k\}$ with estimated probabilities p'_i and prior probabilities p_i, where $i \in \{1, \ldots, n\}$, is

$$IR_B = \frac{\sum_i I_i}{n} \tag{10.9}$$

where $I_i = I_i^+$ for the true class and $I_i = I_i^-$ otherwise, and

$$\begin{aligned} I_i^+ &= \log \frac{p'_i}{p_i} \quad \text{(true value)} \\ I_i^- &= \log \frac{1 - p'_i}{1 - p_i} \quad \text{(false value)} \end{aligned}$$

Theorem 10.2 *Given a batch classification learner (which does not alter its probability estimates after seeing the training data) and a binary classification problem, IR_B is maximal if $p'_i = f$, where f is the frequency of the target class in the test set.*

Proof[¶]. Let S be a test set such that all data items have the same attributes (other than the binary target class value).[‖] To the classifier these items are indistinguishable. S is split into S_1 and S_2 with all items in S_1 belonging to the target class and all items in S_2 belonging to the complement class.

The average IR_B for machine learner M on items S is:

$$IR_B(M) = \sum_{i \in S_1} \frac{\log \frac{p'_i}{p}}{|S|} + \sum_{i \in S_2} \frac{\log \frac{1-p'_i}{1-p}}{|S|} \tag{10.10}$$

Since M is a batch machine learner, it will respond with the same probability p' to each item. So,

$$IR_B(M) = \frac{|S_1| \log \frac{p'}{p}}{|S|} + \frac{|S_2| \log \frac{1-p'}{1-p}}{|S|} \tag{10.11}$$

[¶] With assistance from Lucas Hope.

[‖] If the actual test set is not of this kind, we can partition it so that each subset is. Since the theorem holds for each subset, it will also hold for the union.

© 2004 by Chapman & Hall/CRC Press LLC

The fractions $\frac{|S_1|}{|S|}$ and $\frac{|S_2|}{|S|}$ approximate the true probabilities of being in S_1 and S_2 respectively, so replacing these we get:

$$IR_B(M) \approx f \log \frac{p'}{p} + (1-f) \log \frac{1-p'}{1-p} \tag{10.12}$$

Since we want to find the maximum reward, differentiate with respect to p' and set to 0:

$$\frac{dIR_B(M)}{dp'} = \frac{f}{p'} - \frac{1-f}{1-p'} = 0 \tag{10.13}$$

thus $p' = f$ for maximal IR_B.

10.9 Problems

Problem 1

Prove that IR_B is a proper generalization of IR_G — in other words, prove that IR_B in the binomial case is equivalent to Good's information reward (assuming that the prior probability is uniform).

Problem 2

Start with the decision network for *Julia's manufacturing* problem from §4.9. In case it isn't already, parameterize it so that the decision (between production, marketing, further development) is relatively insensitive to the exact probability of a high vs. low product quality. Then reparameterize it so that the decision is sensitive to product quality. Comment on these results.

Problem 3

In Table 9.4 we gave a utility function for the missing car problem, for the decision network shown in Figure 9.17, with expected utility results in Table 9.5. Investigate the effects on the decision in the following situations:

1. The same relative ordering of outcome preferences is kept, but the quantitative numbers are all made positive.
2. The preference ordering is changed so that most priority is given to not wasting police time.
3. The rate of car theft in the area is much higher, say 1 in 500.

Problem 4

In §10.3.1 we described how it is possible to compute the entropy reduction due to evidence about a pair of observations. Write code to do this using the Netica API.

© 2004 by Chapman & Hall/CRC Press LLC

11

KEBN Case Studies

11.1 Introduction

In this chapter we describe three applications of Bayesian networks. These are, in particular, applications we were involved in developing, and we choose them because we are familiar with the history of their development, including design choices illustrating KEBN (or not), which otherwise must go unreported.

11.2 Bayesian poker revisited

In §5.3, we described a Bayesian decision network for the game of 5-card stud poker. The details of the network structure and betting strategies used by the Bayesian Poker Player (BPP) are a more recent version of a system that has been evolving since 1993. Here we present details of that evolution as an interesting case study in the knowledge engineering of Bayesian networks.

11.2.1 The initial prototype

The initial prototype developed in 1993 [132] used a simple polytree structure. This was largely for pragmatic reasons, as the message passing algorithms for polytrees were simple to implement, and at that time, BN software tools were not widely available*. The original network structure is shown in Figure 11.1(a). The polytree structure meant that we were making the (clearly incorrect) assumption that the final hand types are independent.

In the initial version, the nodes representing hand types were initially given values which divided hands into the nine categories. The size of the model was also limited by compressing the possible values for the *OPP_Action* node to {*pass/call, bet/raise, fold*}. This produced a level of play comparable only to a weak amateur. Since any busted hands, for example, were treated as equal to any other, BPP would bet inappropriately strongly on middling busted hands and inappropriately weakly on

*Early non-commercial tools included IDEAL [266] and Caben [58]; see §B.2 for a brief history of BN software development.

© 2004 by Chapman & Hall/CRC Press LLC

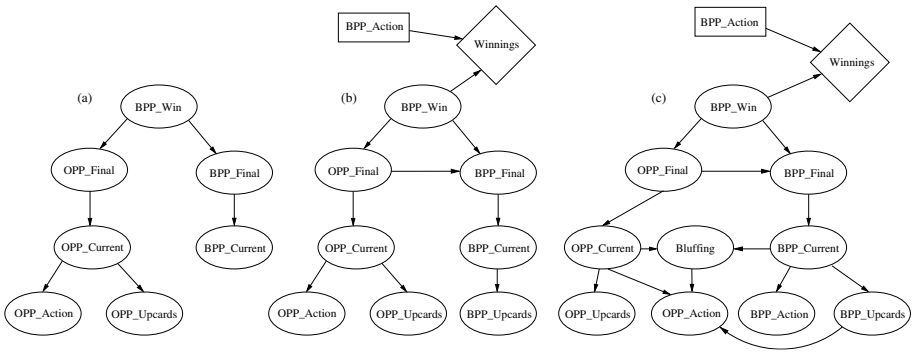


FIGURE 11.1
The evolution of network structures for BPP: (a) polytree used for both 1993 and 1999 version; (b) 2000 decision network version; (c) proposed new structure 2003.

Ace-high hands. The lack of refinement of paired hands also hurt its performance. Finally, the original version of BPP was a hybrid system, using a BN to estimate the probability of winning, and then making the betting decision using randomized betting curves, which were hand constructed. Bluffing was introduced as a random low probability (5%) of bluffing on any individual bet in the final round. The probabilities for the *OPP_Action* node were adapted over time based on observations of opponents' actions. Information about the opponent's betting actions in different situations was used to learn the *OPP_Action* CPT.

11.2.2 Subsequent developments

During 1995, alternative granularities for busted and pair hand types were investigated [274]. At this stage, we introduced an automatic opponent that would estimate its winning probability by dealing out a large sample of final hands from the current situation, and then use simple rules for betting.

In the 1999 version [160] we kept the same polytree structure, but opted for a modest refinement of the hand types, moving from the original 9 hand types to 17, by subdividing busted hands and pairs into low (9 high or lower), medium (10 or J high), queen, king and ace. In this version, the parameters for the betting curves were found using a stochastic greedy search. This version of BPP was evaluated experimentally against: a simple rule-based system; a program which depends exclusively on hand probabilities (i.e., without opponent modeling); and human players with experience playing poker.

A number of individual changes to the system were investigated individually during 2000 [38]. These changes included:

- Expansion of the hand type granularity from 17 to 24, by explicitly including all 9 different pair hands.
- Allowing *OPP_Action* to take all four different action values, {*fold, pass, call, raise*}.

© 2004 by Chapman & Hall/CRC Press LLC

- Adding an arc from *BPP_Final* to *OPP_Final*, to reflect the dependence between them.
- Extending the BN to a decision network, in particular changing the decision method from one based on probability betting curves to one based on expected winnings.
- Improving the bluffing by taking into account the opponent's belief in BPP winning and making bluffing consistent throughout the last round when undertaken.

The network structure for this version, previously shown in Figure 5.5 is shown again in Figure 11.1(b) for easy comparison.

11.2.3 Ongoing Bayesian poker

Finally, as this text is being written, our poker project continues [29]. Possible directions of ongoing research include the following.

Ideally, the program should also take advantage of the difference between BPP's upcards and its full hand. The point is that when one's strong cards are showing on the table, there is no reason to bet coyly; on the contrary, it is advantageous to make opponents pay for the opportunity of seeing any future cards by betting aggressively. On the other hand, when one's strongest card is hidden, aggressive betting can drive opponents out of the game prematurely. This could be done by using the BN model from both BPP and the opponent's side to obtain different user model views of the strength of BPP. When the opponent's view is "strong" but BPP's view is weak, BPP should bet heavily. When both views are strong, BPP should bet more softly to keep the opponent playing.

Just as for BPP, the conservativeness or aggressiveness of an opponent's play, which is learned and captured by recalibrating the matrices relating *OPP Current* and *OPP Action*, does not fully describe the bluffing behavior of the opponent. A plausible extension would be to add an opponent bluffing node which is a parent of *OPP Action* and a child of *OPP Current* and *BPP Current* (since the latter gives rise to BPP's upcards and behavior, even though they are not explicitly represented). Another structural change important for improved learning and play would be to make *OPP Action* a child node of a new *BPP Upcards* node, so that what the opponent observes of BPP's hand would jointly condition his or her behavior. Figure 11.1(c) shows the proposed new network structure.

We are also modifying BPP to play Texas-Hold'em Poker, to allow us to compete against other automated opponents online. These online gaming environments will also require extensions to handle multi-opponent games. In multiple opponent games it will be more important to incorporate the interrelations between what is known of different player's hands and the node representing their final hands.

We also anticipate using a dynamic Bayesian network (DBN) to model more effectively the interrelation between rounds of play; more details of this modeling problem are left as a homework problem (see Problem 5.8).

© 2004 by Chapman & Hall/CRC Press LLC

Finally, we are aware that the lack of non-showdown information is likely to introduce some selection bias into the estimates of the conditional probabilities, but we have not yet attempted to determine the nature of this bias.

11.2.4 KEBN aspects

There are several points to make about this case study, in terms of the overall KEBN process.

All the network structures have been hand-crafted, while the parameters have been either generated (by dealing out large numbers of poker hands) or learnt during playing sessions. This combination of elicitation and learning is fairly typical of BN projects to date, where the amount of expert domain knowledge (and lack of data) means that hand-crafting the structure is the only option.

Our domain expert on this project was one of the authors (who has had considerable poker playing experience), which meant we didn't have to deal with the KE difficulties associated with using an expert who is unfamiliar with the technology.

As we are interested in poker as a challenging application for our BN research agenda, BPP has never been deployed in a "live" environment where BPP and its opponents are playing for money. This has undoubtedly limited the validation and field testing phases of our proposed lifecycle model (see Figure 9.1), and there has certainly been no industrial use.

On the other hand, there *has* been an iterative development cycle, with continual refinement of the versions, following the incremental prototype model. We started with sufficient assumptions to allow the development of a simple but working initial model and gradually increased the complexity, adding variables, increasing the number of values, adding arcs (moving from a polytree to a graph) and extending a BN into a decision network.

The management of versions has been adequate, allowing subsequent versions of BPP to be tested against earlier versions. We did not document changes and the rationale for them as part of a formal KEBN process, but fortunately a series of research reports and papers served that purpose.

11.3 An intelligent tutoring system for decimal understanding

In this section we present a case study in the construction of a BN in an intelligent tutoring system (ITS) application[†], specifically decimal understanding, for the target age range of Grades 5 to 10 [204, 267].

To understand the meaning and size of numbers written using a decimal point involves knowing about place value columns, the ordering of numbers by size and the

[†] There have been a number of other successful ITS systems that use Bayesian networks, e.g., [50, 49, 184, 283].

© 2004 by Chapman & Hall/CRC Press LLC

value of digits. Although this is a topic plagued by misconceptions, very often students don't know they harbour them. Our education domain experts had been working in this area for some time, gathering and analyzing data on students' thinking about decimals. An ITS consisting of computer games involving decimals was being designed and developed as a useful learning tool to supplement classroom teaching. Our domain experts needed an underlying reasoning engine that would enable the ITS to diagnose and target an individual's wrong way of thinking about decimals; here we describe the development of a BN to do this.

We begin with the background domain information that was available at the start of the project (§11.3.1), and then we present the overall architecture of the ITS architecture (§11.3.2). Next we give a detailed description of the both expert elicitation phase, including evaluations (§11.3.3), and the investigation that was undertaken using automated methods (§11.3.4). Finally, we describe results from field trials and draw some conclusions from the case study as a whole.

11.3.1 The ITS domain

Students' understanding of decimal numeration has been mapped using a short test, the Decimal Comparison Test (DCT), where the student is asked to choose the larger number from each of 24 pairs of decimals [269]. The pairs of decimals are carefully chosen so that from the patterns of responses, students' (mis)understanding can be diagnosed as belonging to one of a number of classifications. These classifications have been identified manually, based on extensive research [269, 234, 241, 268]. The crucial aspects are that misconceptions are prevalent, that students' behavior is very often highly consistent and that misconceptions can be identified from patterns amongst simples clues.

About a dozen misconceptions have been identified [269], labeled vertically in Table 11.1. This table also shows the rules the domain experts originally used to classify students based on their response to 6 types of DCT test items (labeled horizontally across the top of the table): H = High number correct (e.g., 4 or 5 out of 5), L = Low number correct (e.g., 0 or 1 out of 5), with '.' indicating that any performance level is observable for that item type by that student class other than the combinations seen above. Most misconceptions are based on false analogies, which are sometimes embellished by isolated learned facts. For example, many younger students think 0.4 is smaller than 0.35 because there are 4 parts (of unspecified size, for these students) in the first number and 35 parts in the second. However, these "whole number thinkers" (LWH, Table 11.1) get many questions right (e.g., 5.736 compared with 5.62) with the same erroneous thinking. So-called 'reciprocal thinking' students (SRN, Table 11.1) choose 0.4 as greater than 0.35 but for the wrong reason, as they draw an analogy between fractions and decimals and use knowledge that 1/4 is greater than 1/35.

The key to designing the DCT was the identification of "item types." An item type is a set of items which a student with any misconception should answer consistently (either all right or all wrong). The definition of item types depends on both the mathematical properties of the item and the psychology of the learners. In practice,

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 11.1
Response patterns expected from students with different misconceptions

Coarse Class	Fine Class	Description	Item type (with sample item)					
			1 0.4 0.35	2 5.736 5.62	3 4.7 4.08	4 0.452 0.45	5 0.4 0.3	6 0.42 0.35
A	ATE	apparent expert	H	H	H	H	H	H
	AMO	money thinker	H	H	H	L	H	H
	AU	unclassified A	H	H	.	.	.	.
L	LWH	whole number thinking	L	H	L	H	H	H
	LZE	zero makes small	L	H	H	H	H	H
	LRV	reverse thinking	L	H	L	H	H	L
	LU	unclassified L	L	H	.	.	.	.
S	SDF	denominator focused	H	L	H	L	H	H
	SRN	reciprocal negative	H	L	H	L	L	L
	SU	unclassified S	H	L	.	.	.	.
U	MIS	misrule	L	L	L	L	L	L
	UN	unclassified	.	.	.	.	.	.

the definition is also pragmatic — the number of theoretically different item types can be very large, but the extent to which diagnostic information should be squeezed from them is a matter of judgement. The fine misconception classifications had been "grouped" by the experts into a coarse classification — `L` (think longer decimals are larger numbers), `S` (shorter is larger), `A` (correct on straightforward items (Types 1 & 2)) and `U` (other). The `LU`, `SU` and `AU` "catch-all" classifications for students who on their answers on Type 1 and 2 items behave like others in their coarse classification, but differ on other item types. These and the `UNs` may be students behaving consistently according to an unknown misconception, or students who are not following any consistent interpretation.

The computer game genre was chosen to provide children with an experience different from, but complementary to, normal classroom instruction and to appeal across the target age range (Grades 5 to 10). The system offers several games, each focused on one aspect of decimal numeration, thinly disguised by a story line.

In the "Hidden Numbers" game students are confronted with two decimal numbers with digits hidden behind closed doors; the task is to find which number is the larger by opening as few doors as possible. Requiring similar knowledge to that required for success on the DCT, the game also highlights the place value property that the most significant digits are those to the left. The game "Flying Photographer" requires students to "photograph" an animal by clicking when an "aeroplane" passes a specified number on a numberline. The "Number Between" game is also played on a number line, but particularly focuses on the density of the decimal numbers; students have to type in a number between a given pair. Finally, "Decimaliens" is a classic shooting game, designed to link various representations of the value of digits in a decimal number. These games address several of the different tasks required of an integrated knowledge of decimal numeration based on the principles of place value. It is possible for a student to be good at one game or the diagnostic test, but

© 2004 by Chapman & Hall/CRC Press LLC

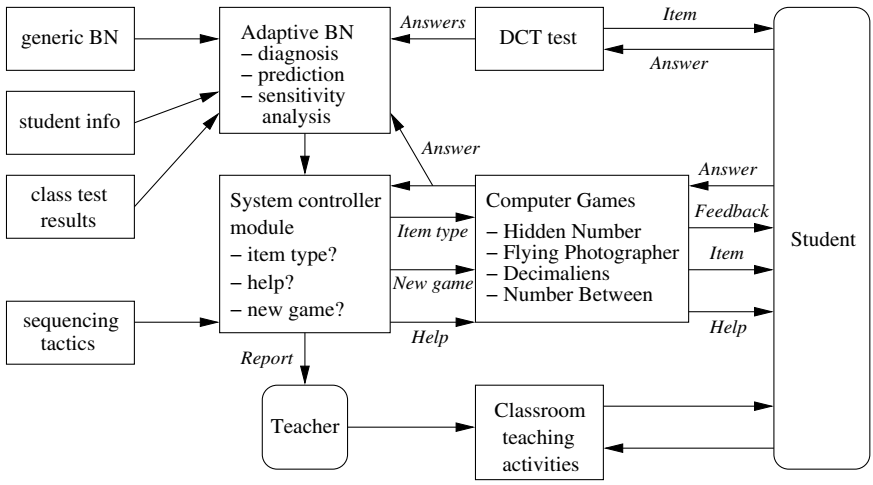


FIGURE 11.2
Intelligent tutoring system architecture.

not good at another; emerging knowledge is often compartmentalized.

11.3.2 ITS system architecture

The high-level architecture of our system is shown in Figure 11.2. The BN is used to model the interactions between a student's misconceptions, their game playing abilities and their performance on a range of test items. The BN is initialized with a generic model of student understanding of decimal numeration constructed using the DCT results from a large sample of students [270]. The network can also be tailored to an individual student using their age or results from a short online DCT. During a student's use of the system, the BN is given information about the correctness of the student's answers to different item types encountered in the computer games.

Given the model, and given evidence about answers to one or more item types, the Bayesian belief updating algorithm then performs diagnosis; calculating the probabilities that a student with these behaviors has a particular misconception. Changes in the beliefs in the various misconceptions are in turn propagated within the network to perform prediction; the updating algorithm calculates the new probabilities of a student getting other item types right or wrong. After each set of evidence is added and belief updating performed, the student model stored within the network is updated, by changing the misconception node priors. Updating the network in this way as the student plays the game (or games) allows changes in students' thinking and skills to be tracked. The identification of the misconception with the highest probability provides the best estimate of the students' current understanding.

The information provided by the BN is, in turn, used by a controller module, together with the specified sequencing tactics (see below), to do the following: se-

© 2004 by Chapman & Hall/CRC Press LLC

lect items to present to the student; or decide whether additional help presentation is required; or decide when the user has reached expertise and should move to another game. The controller module also makes a current assessment of the student available to the teacher and reports on the overall effectiveness of the adaptive system. The algorithm for item type selection incorporates several aspects, based on the teaching model. Students are presented with examples of all item types at some stage during each session. Students meeting a new game are presented with items that the BN predicts they are likely to get correct to ensure they have understood the rules and purposes of the game. If the probabilities of competing hypotheses about the student's misconception classification are close, the system gives priority to diagnosis and proposes a further item to be given to the user. By employing the Netica BN software's "sensitivity to findings" function (see §10.3.1), the user is presented with an item that is most likely to distinguish between competing hypotheses. Finally, the network selects items of varying difficulty in an appropriate sequence for the learner. The current implementation of the system allows comparison of different sequencing tactics: presenting the hard items first, presenting the easy items first and presenting easy and hard items alternately. Note that the terms easy and hard are relative to the individual student and based on the BN's prediction as to whether they will get the item correct.

The ITS also incorporates two forms of teaching into the games: clear feedback and visual scaffolding. For example, in the Flying Photographer, where students have to place numbers on number lines, feedback is given after each placement (see Figure 11.3): if the position is incorrect a sad red face appears at the point, whereas the correct position is (finally) marked by a happy green face. In this game, the visual scaffolding marks intermediate numbers on the interval; for example, when a student has twice failed to place a number correctly on the interval [0,1], the intermediate values 0.1, 0.2, 0.3, ..., 0.9 appear for this and the next item.

11.3.3 Expert elicitation

In this section we describe the elicitation of a fragment of the decimal misconception BN from the education domain experts. This fragment was the first prototype structure built, involving only the diagnosis of misconceptions through the DCT; the game fragments were constructed in later stages of the project. This exposition is given sequentially: the identification of the important variables and their values; the identification and representation of the relationships between variables; the parameterization of the network; and finally the evaluation. In practice, of course, there was considerable iteration over these stages.

11.3.3.1 Nodes

We began the modeling with the main "output" focus for the network, the representation of student misconceptions. The experts already had two levels of classifications for the misconceptions, which we mapped directly onto two variables. The *coarseClass* node can take the values $\{L, S, A, UN\}$, whereas the *fineClass* node, incorporating all the misconception types identified by the experts, can take the 12

© 2004 by Chapman & Hall/CRC Press LLC

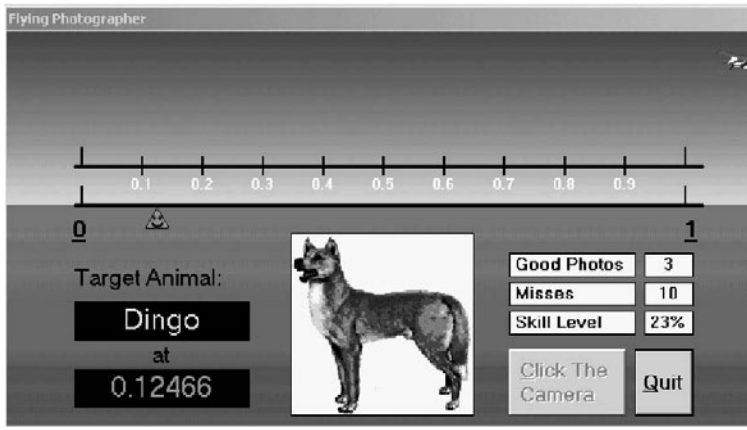


FIGURE 11.3
A screen from the ITS "Flying Photographer" game showing visual feedback (triangle) and scaffolding (marked intervals).

values shown in column 1 of Table 11.1. Note that the experts considered the classifications to be mutually exclusive; at any one time, the student could only hold one misconception. While the experts knew that the same student could hold different misconceptions at different times, they did not feel it was necessary, at least initially, to model this explicitly with a DBN model.

Each different DCT item type was made an observation variable in the BN, representing student performance on test items of those types; student test answers are entered as evidence for these nodes. The following alternatives were considered for the possible values of the item type nodes.

- Suppose the test contains N items of a given type. One possible set of values for the BN item type node is {0,1, ..., N}, representing the number of the items the student answered correctly. The number of items may vary for the different types and for the particular test set given to the students, but it is not difficult to adjust the BN. Note that the more values for each node, the more complex the overall model; if N were large (e.g., > 20), this model may lead to complexity problems.

- The item type node may be given the values {*High, Medium, Low*}, reflecting an aggregated assessment of the student's answers. For example, if 5 such items were presented, 0 or 1 correct would be considered *low*, 2 or 3 would be *medium*, while 4 or 5 would be *high*. This reflects the expert rules classification described above.

Both alternatives were evaluated empirically (see §11.3.3.5 below); however the H/M/L option was used in the final implementation.

© 2004 by Chapman & Hall/CRC Press LLC

11.3.3.2 Structure

The experts considered the coarse classification to be a strictly deterministic combination of the fine classification; hence, the *coarseClass* node was made a child of the *fineClass* node. For example, a student was considered an *L* if and only if it was one of an *LWH*, *LZE*, *LRV* or *LU*. For the DCT test item types, the *coarseClass* node was not necessary; however, having it explicitly in the network helped both the expert's understanding of the network's reasoning and the quantitative evaluation. In addition, including this node allowed modeling situations where all students with the same coarse misconception exhibit the same behavior (e.g., in the Flying Photographer game).

The *type* nodes are observation nodes, where entering evidence for a type node should update the posterior probability of a student having a particular misconception. As discussed in §2.3.1, such diagnostic reasoning is typically reflected in a BN structure where the class, or "cause," is the parent of the "effect" (i.e., evidence) node. Therefore an arc was added from the *fineClass* node to each of the *type* nodes. No connections were added between any of the *type* nodes, reflecting the experts' intuition that a student's answers for different item types are independent, given the subclassification.

A part of the expert elicited BN structure implemented in the ITS is shown in Figure 11.4. This network fragment shows the *coarseClass* node (values $\{L,S,A,UN\}$), the detailed misconception *fineClass* node (12 values), the item *type* nodes used for the DCT, plus additional nodes for the Hidden Number and Flying Photographer games. The bold item type nodes are those corresponding to the DCT test data set and hence used subsequently for evaluation and experimentation.

The "HN" nodes relate to the Hidden Numbers game, with evidence entered for the number of doors opened before an answer was given (*HN_nod*) and a measure of the "goodness of order" in opening doors (*HN_gos*). The root node for the Hidden Number game subnet reflects a player's game ability, specifically their door opening "efficiency" (*HN_eff*). The "FPH" node relate to the Flying Photographer game. The node *FPH_ls* records evidence when students have to place a long number, which is small in size, such as 0.23456. As noted above, LWH students are likely to make an error on this task. The other nodes perform similar functions, with the root node *FPH_pa* reflecting a player's overall game ability.

The BNs for the other two games are separate BNs as student performance on these provides information about abilities other than the 12 misconceptions; however their construction was undertaken using similar principles.

11.3.3.3 Parameters

The education experts had collected data that consisted of the test results and the expert rule classification on a 24 item DCT for over 2,500 test papers from students in Grades 5 and 6. These were then pre-processed to give each student's results in terms of the 6 test item types; 5,5,4,4,3,3 were the number of items of these type 1 to 6 respectively. The particular form of the pre-processing depends on the item type values used: with the 0-N *type* node values, a student's results might be `541233`,

© 2004 by Chapman & Hall/CRC Press LLC

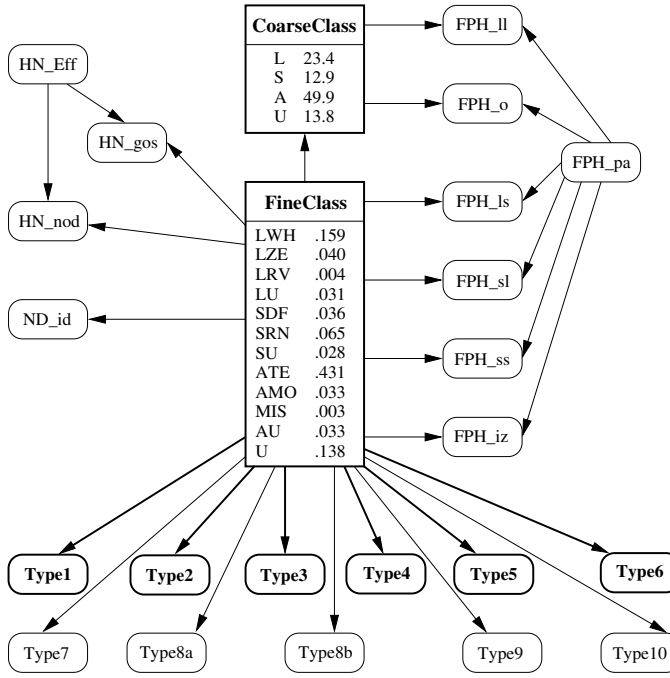


FIGURE 11.4
Fragment of the expert elicited BN currently implemented. Bold nodes are those discussed here.

whereas with the H/M/L values, the same student's results would be represented as *H H L M H H*.

The expert rule classifications were used to generate the priors for the sub-classifications. The priors are found to vary slightly between different classes and teaching methods, and quite markedly between age groups; however these variations are ignored for current purposes. All the CPTs of the item types take the form of

- $P(Type = Value|Classification = X)$

As we have seen from the domain description, the experts expect particular classes of students to get certain item types correct and others wrong. However we do need to model the natural deviations from such "rules," where students make a careless error. We model this uncertainty by allowing a small probability of a careless mistake on any one item. For example, students who are thinking according to the LWH misconception are predicted to get all 5 items of Type 2 correct. If, however, there is a probability of 0.1 of a careless mistake on any one item, the probability of a score of 5 is $(0.9)^5$, and the probability of other scores follows the binomial distribution; the full vector for P(Type2|Subclass=LWH) is (0.59,0.33,0.07,0.01,0.00,0.00) (to two decimal places). When the item type values H/M/L are used, the numbers are accumulated to give the vector (0.92,0.08,0.00) for H, M and L.

© 2004 by Chapman & Hall/CRC Press LLC

The experts considered that this mistake probability was considerably less than 0.1, of the order of 1-2%. We ran experiments with different probabilities for a single careless mistake (pcm=0.03, 0.11 and 0.22), with the CPTs calculated in this manner, to investigate the effect of this parameter on the behavior of the system. These numbers were chosen to give a combined probability for *HIGH* (for 5 items) of 0.99, 0.9 and 0.7 respectively, numbers that our experts thought were reasonable.

Much more difficult than handling the careless errors in the well understood behavior of the specific known misconceptions is to model situations where the experts do not know how a student will behave. This was the case where the experts specified '.' for the classifications *LU*, *SU*, *AU* and *UN* in Table 11.1. We modeled the expert not knowing what such a student would do on the particular item type in the BN by using 0.5 (i.e., 50/50 that a student will get each item correct) with the binomial distribution to produce the CPTs.

11.3.3.4 The evaluation process

During the expert elicitation process we performed the following three basic types of evaluation. First was *case-based evaluation* (see §10.4) , where the experts "play" with the net, imitating the response of a student with certain misconceptions and review the posterior distributions on the net. Depending on the BN parameters, it was often the case that while the incorporation of the evidence for the 6 item types from the DCT test data greatly increased the BN's belief for a particular misconception, the expert classification was not the BN classification with the highest posterior, because it started with a low prior. We found that it was useful to the experts if we also provided the ratio by which each classification belief had changed (although the highest posterior is used in all empirical evaluations). The case-based evaluation also included sequencing, where the experts imitate repeated responses of a student, update the priors after every test and enter another expected test result. The detection of the more uncommon classifications through repetitive testing built up the confidence of the experts in the adaptive use of the BN.

Next, we undertook *comparative evaluation* between the classifications of the BN compared to the expert rules on the DCT data. It is important to note here that the by-hand classification is only a best-guess of what a student is thinking — it is not possible to be certain of the "truth" in a short time frame. As well as a comparison grid (see next subsection), we provided the experts with details of the records where the BN classification differed from that of the expert rules. This output proved to be very useful for the expert in order to understand the way the net is working and to build confidence in the net.

Finally, we performed *predictive evaluation* (see §10.5.1) which considers the prediction of student performance on individual item type nodes rather than direct misconception diagnosis. We enter a student's answers for 5 of the 6 item type nodes, then predict their answer for the remaining one; this is repeated for each item type. The number of correct predictions gives a measure of the predictive accuracy of each model, using a score of 1 for a correct prediction (using the highest posterior) and 0 for an incorrect prediction. We also look at the predicted probability for the actual

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 11.2
Comparison grid: expert rule (vertical) vs expert elicited BN (horizontal) classification; type node states 0-N, pcm=0.11. Desirable re-classifications are *italicized*, while undesirable ones are in **bold**.

	lwh	lze	lrv	lu	sdf	srn	su	ate	amo	mis	au	un
lwh	386	0	0	0	0	0	0	0	0	0	0	0
lze	0	98	0	0	0	0	0	0	0	0	0	0
lrv	**10**	0	0	0	0	0	0	0	0	0	0	0
lu	*6*	*9*	0	54	0	0	0	0	0	0	0	**6**
sdf	0	0	0	0	83	0	**4**	0	0	0	0	0
srn	0	0	0	0	0	159	0	0	0	0	0	0
su	0	0	0	0	*2*	*22*	40	**3**	0	0	0	**2**
ate	0	0	0	0	0	0	0	1050	0	0	0	0
amo	0	0	0	0	0	0	0	0	79	0	0	0
mis	0	0	0	0	0	0	0	0	0	6	0	0
au	**9**	0	0	0	0	0	0	*63*	*8*	0	0	**1**
un	*43*	*6*	0	*15*	*35*	*14*	*11*	*119*	*26*	*2*	0	66

student answer. Both measures are averaged over all students.

We performed these types of evaluation repeatedly throughout the development of the network, which showed the effect changes in structure or parameters may have had on the overall behavior. The iterative process halted when the experts felt the behavior of the BN was satisfactory.

11.3.3.5 Empirical evaluation

Table 11.2 is an example of the comparison grids for the classifications that were produced during the comparison evaluation phase. The vertical list down the first column corresponds to the expert rules classification, while the horizontal list across the top corresponds to the BN classification, using the highest posterior; each entry in the grid shows how many students had a particular combination of classifications from the two methods. The grid diagonals show those students for whom the two classifications are in agreement, while the "desirable" changes are shown in *italics* and undesirable changes are shown in **bold**. Note that we use the term "match," rather than saying that the BN classification was "correct," because the expert rule classification is not necessarily ideal.

Further assessment of these results by the experts revealed that when the BN classification does not match the expert rules classification, the misconception with the second highest posterior often did match. The experts then assessed whether differences in the BN's classification from the expert rules classification were in some way desirable or undesirable, depending on how the BN classification would be used. They came up with the following general principles which provided some general comparison measures.

1. It is **desirable** for expert rule classified LUs to be re-classified as another of the specific Ls, similarly for AUs and SUs, and it was desirable for Us to be

© 2004 by Chapman & Hall/CRC Press LLC

TABLE 11.3
Results showing fine classification summary comparison of various models compared to the expert rules (match, desirable and undesirable changes), together with accuracy of various models predicting student item type answers (using two different measures)

Method	Type values		Match	Des. change	Undes. change	Avg Pred. Accuracy	Avg Pred. Prob.
Expert	0-N	0.22	77.88	20.39	1.72	0.34	0.34
BN		0.11	82.93	15.63	1.44	0.83	0.53
		0.03	84.37	11.86	3.78	0.82	0.70
	H/M/L	0.22	80.47	18.71	0.82	0.89	0.69
		0.11	83.91	13.66	2.42	0.89	0.80
		0.03	90.40	6.48	3.12	0.88	0.83
SNOB	24 DCT		79.81	17.60	2.49		
	0-N		72.06	16.00	11.94		
	H/M/L		72.51	17.03	10.46		
learned	0-N	Avg	95.97	2.36	1.66	0.83	0.74
parameters	H/M/L	Avg	97.63	1.61	0.75	0.89	0.83
CaMML	0-N	Avg	86.51	5.08	8.41	0.83	0.72
constr.	H/M/L	Avg	83.48	8.12	8.34	0.88	0.79
CaMML	0-N	Avg	86.15	5.87	7.92	0.83	0.74
uncons.	H/M/L	Avg	92.63	4.61	2.76	0.89	0.83

re-classified as anything else (because this is dealing with borderline cases that the expert rule really can't say much about).

2. It is **undesirable** for (a) specific classifications (i.e., not those involving any kind of "U") to change, because the experts are confident about these classifications, and (b) for any classification to change to UN, because this is in some sense throwing away information (e.g., LU to UN loses information about the "L-like" behavior of the students).

Many classification comparison grids were obtained through varying the probability of a careless mistake (pcm=0.22, 0.11 and 0.03) and the item type values (0-N vs H/M/L). The percentages for match, desirable and undesirable change were calculated for each grid.

Considerable time was spent determining the following factors that might be causing the differences between the BN and the expert rule classifications. First, the expert rules give priority to the type 1 and type 2 results, whereas the BN model gives equal weighting to all 6 item types. Second, the expert elicited BN structure and parameters reflects both the experts' good understanding for the known fine classifications, and their poor understanding of the behavior of "U" students (LU, SU, AU and UN). Finally, as discussed earlier, some classes are broken down into fine classifications more than others, resulting in lower priors, so the more common classifications (such as ATE and UN) tend to draw in others.

Table 11.3 (Set 1) shows a summary of the expert BN fine classification, varying the type values and probability of careless mistake, in terms of percentage of matches (i.e., on the grid diagonal), desirable changes and undesirable changes and the two prediction measures (see §10.5.1), averaged over all predicted item types, for all students. The experts considered the undesirable change percentages to be quite low, especially considering that they felt some of these can be considered quite justified.

© 2004 by Chapman & Hall/CRC Press LLC

Overall the experts were satisfied that the elicited network performs a good classification of students' misconceptions and captures well the different uncertainties in the experts' domain knowledge. In addition, they were reassured by the fact that its performance appeared quite robust to changes in parameters such as the probability of careless mistakes or the granularity of the evidence nodes.

11.3.4 Automated methods

The next stage of the project involved the application of certain automated methods for knowledge discovery to the domain data, for each main task in the construction process.

1. We applied a classification method to student test data.
2. We performed simple parameter learning based on frequency counts to the expert BN structures; and
3. We applied an existing BN learning program, CaMML [294].

In each case we compared the performance of the resultant network with the expert elicited networks, providing an insight into how elicitation and knowledge discovery might be combined in the BN knowledge engineering process.

11.3.4.1 Classification

The first aspect investigated was the classification of decimal misconceptions. We applied the SNOB probabilistic classification program [289], based on the information theoretic Minimum Message Length (MML) [290] to the data from 2437 testpapers in different forms:

1. The raw data from the 24 DCT items
2. The data pre-processed from 24 items into the 6 item types
   (a) Using the values 0-N
   (b) Using the values H/M/L

Using the SNOB's most probable class for each student, we constructed comparison grids comparing the SNOB classification with the expert rule classification.

Given the raw data of 24 DCT items, SNOB produced 12 classes, 8 of which corresponded closely to the expert classifications (i.e., had most members on the grid diagonal). Two classes were not found (LRV and SU). Of the other 4 classes, 2 were mainly combinations of the AU and UN classifications, while the other 2 were mainly UNs. SNOB was unable to classify 15 students (0.6%). The percentages of match, desirable and undesirable change are shown in Table 11.3 (set 2, row 1). They are comparable with the expert BN 0-N and only slightly worse than the expert BN H/M/L results.

The comparison results using the data pre-processed into 6 item types (values 0-N and H/M/L) were not particularly good.

© 2004 by Chapman & Hall/CRC Press LLC

- For O-N type values, SNOB found only 5 classes (32 students = 1.3% not classified), corresponding roughly to some of the most populous expert classes (LWH, SDF, SRN, ATE and UN), subsuming the other expert classes.
- For H/M/L type values, SNOB found 6 classes (33 students = 1.4% not classified), corresponding roughly to 5 of the most populous expert classes (LWH, SDF, SRN, ATE, UN), plus a class that combined MIS with UN. The match results are shown in Table 11.3 (set 2, rows 2 and 3).

Clearly, summarizing the results of 24 DCT into 6 item types gives a worse classification; one explanation of this was that many pairs of the classes are distinguished by student behavior on just one item type, and SNOB might consider these differences to be noise within one class.

The overall good performance of the classification method shows that automated knowledge discovery methods may be useful in assisting experts to identify suitable values for classification type variables, particularly when extensive analysis of the domain is not available.

11.3.4.2 Parameters

Our next investigation was to learn the parameters from the data, while keeping the expert elicited network structure. The data was randomly divided into five 80%-20% splits for training and testing; the training data was used to parameterize the expert BN structures (using the the Spiegelhalter-Lauritzen Algorithm 7.1 in Netica), while the test data was given to the resultant BN for classification. The match results (averaged over the 5 splits) for the fine classification comparison of the expert BN structures (with the different type values, 0-N and H/M/L) with learned parameters are shown in Table 11.3 (set 3), together with corresponding prediction results (also averaged over the 5 splits).

Clearly, learning the parameters from data, if it is available, gives results that are much closer to the expert rule classification than using the parameters elicited from the experts. The trade-off is that the network no longer makes changes to the various "U" classifications, i.e., it doesn't shift LUs, SUs, AUs and UNs into other classifications that may be more useful in a teaching context. However it does mean that expert input into the knowledge engineering process can be reduced, by doing the parameter learning on an elicited structure.

11.3.4.3 Structure

Our third investigation involved the application of CaMML [294] (see §8.5) to learn network structure. In order to compare the learned structure with that of the expert elicited BN, we decided to use the pre-processed 6 type data; each program was given the student data for 7 variables (the fine classification variable and the 6 item types), with both the 0-N values and the H/M/L values. The same 5 random 80%-20% splits of the data were used for training and testing. The training data was given as input to the structural learning algorithm and then used to parameterize the result networks using Netica's parameter learning method.

© 2004 by Chapman & Hall/CRC Press LLC

We ran CaMML once for each split (a) without any constraints and (b) with the ordering constraint that the classification should be an ancestor of each of the type nodes. This constraint reflects the general known causal structure. Each run produced a slightly different network structure, with some having the *fineClass node* as a root, some not. Two measures of network complexity were used: (i) ratio of arcs/nodes, which varied from 1.4 to 2.2 and (ii) the total number of probabilities in the CPTs, which varied from about 700 to 144,000. The junction-tree cost was not used as a measure, although it probably should have been!

The percentage match results comparing the CaMML BN classifications (constrained and unconstrained, O-N and H/M/L) are also shown in Table 11.3 (sets 4 and 5), together with the prediction results. The undesirable changes include quite a few shifts from one specific classification to another, which is particularly bad as far as our experts are concerned. The variation between the results for each data set 1-5 was much higher than for the variation when learning parameters for the expert BN structure, no doubt reflecting the difference between the network structure learned for the different splits. However we did not find a clear correlation between the complexity of the learned network structures and their classification performance.

Our experts also looked at the learnt structures during this phase of the project, but they did not have an intuitive feel for how these structures were modeling the domain. In particular, the change in the direction of the arc between the classification node and (one or more of) the item type nodes did not reflect the causal model we had introduced them to during the elicitation phase. Also, there were many arcs *between* item type nodes, and they could not explain these dependencies with their domain knowledge. Some time later, one of the experts investigated one of these learnt structures using Matilda (see §9.3.2.2), as part of Matilda's evaluation process. By using this KEBN support tool, the expert gained some understanding of the dependencies captured in the seemingly non-intuitive structure. Overall, however, the lack of an adequate explanation for the learnt structures, together with the unacceptable undesirable re-classifications, meant that none of the learnt structures were considered for inclusion in the implemented system.

11.3.5 Field trial evaluation

All components of the complete system were field-tested. The games were trialed with individual students holding known misconceptions and their responses and learning have been tracked [187]. This has refined the design of the games and of the visual scaffolding and led to the decision to provide the feedback and visual scaffolding automatically.

The complete system has also been field tested with 25 students in Grades 5 and 6, who had persistent misconceptions after normal school instruction [84, 105]. Students worked with a partner (almost always with the same misconception) for up to 30 minutes, without adult intervention. The observer recorded their conversations, which were linked to computer results and analyzed to see where students learned or missed learning opportunities and how cognitive conflict was involved. Long term conceptual change was measured by re-administering the DCT about three weeks

© 2004 by Chapman & Hall/CRC Press LLC

later. Students played in pairs so that the observer could, without intervention, monitor their thinking as revealed by their conversations as they played the games.

Ten students tested as experts on the delayed post-test, indicating significant progress. Seven students demonstrated improvement while eight retained their original misconception. There were some instances where students learned from the visual scaffolding of the help screens, but active teacher intervention seems required for most students to benefit fully from these. Very frequently, students learned by observing and discussing with their partners, but they did not always learn the same things at the same time. This means that the computer diagnosis was not necessarily meaningful for both students so that the item type selection may not perform as designed for either student. This disadvantage needs to be weighed against the benefits of working with a partner.

Feedback provided by the games provoked learning in two ways. In some instances students added new information to their conceptual field, without addressing misconceptions (e.g., learned that 0 in the tenths column makes a number small, without really changing basic whole number thinking). In other instances the feedback provoked cognitive conflict and sometimes this was resolved within the session, resulting in a significant change from a misconception to expertise, maintained at the delayed post-test. The item type selection was set to alternate between "easy" and "hard" items for these field trials but this experiment indicated that it gave too many easy items. The real-time updated diagnosis by the system of the student's thinking patterns was (generally) consistent with the observer's opinion. Discrepancies between classifications and the delayed post-test were tracked to known limitations of the DCT, which could not diagnose an unusual misconception prevalent in that class.

On balance, our experts consider the ITS to be a useful supplement to class instruction.

11.3.6 KEBN aspects

The combination of elicitation and automated methods used in this case study was only possible because we had both the involvement of experts with a detailed understanding of the basis of the misconceptions and how they relate to domain specific activities and an extensive data set of student behavior on test items in the domain.

Building the student model from misconceptions (cf. [258, 98]), rather than in terms of gaps in correct pieces of domain knowledge (cf. [50]), is a little unusual and is only viable because of the nature of the domain and the extensive research on student understanding in the domain discussed in §11.3.1.

This application is relatively complex in that the BN is used in several ways, namely for diagnosis, prediction and value of information.

The automated techniques were able to yield networks which gave quantitative results comparable to the results from the BN elicited from the experts. However, the experts preferred to use the elicited BN in the implemented system, based on their qualitative evaluation of the undesirable re-classifications.

The quantitative results do provide a form of 'validation' of the learning techniques and suggests that automated methods can reduce the input required from do-

© 2004 by Chapman & Hall/CRC Press LLC

main experts. In any case, these results build confidence in the elicited BNs. It also supports the reciprocal conclusion regarding the validity of manual construction when there is enough expert knowledge but no available data set.

In addition, we have seen that the use of automated techniques can provide opportunities to explore the implications of modeling choices and to get a feel for design tradeoffs — some examples of this were reported above in both the initial elicitation stage and the discovery stage (e.g., 0-N *vs.* H/M/L).

The actual process of undertaking a quantitative evaluation was tedious, requiring programming in the BN software API to run sets of experiments and collecting and collating results. Much data massaging was required, with all the automated methods requiring different input data formats and producing different output.

While the evaluation consisted of the standard set of case-based, comparative and predictive evaluations, domain specific criteria (such as the division into desirable and undesirable classification changes) were developed during the investigation process and the experts refined their requirements.

The implemented system contains a mechanism for fully recording sessions (with both user inputs and the BNs outputs), which could be invaluable for ongoing evaluation and maintenance of the system. However the system will be deployed via the distribution of CD-ROM to interested math teachers, and there is no incentive or mechanism for them to send this data back to us, the system developers.

11.4 Seabreeze prediction

In this case study[‡] we prototyped the use of Bayesian networks (BNs) for improved weather prediction, applying them to the prediction of seabreezes [142]. This comparatively simple domain problem was chosen as a "proof of concept" for meteorological applications and taking advantage of weather monitoring set up for the Sydney 2002 Olympics. In the study we compared an existing Bureau of Meteorology (BOM) rule-based system developed by our domain experts to an expert elicited BN and others learned by two data mining programs, TETRAD II [264] (§6.3.2) and CaMML [294] (see §8.5). All of the Bayesian networks significantly outperformed the rule-based system.

11.4.1 The seabreeze prediction problem

Seabreezes occur because of the unequal heating and cooling of neighboring sea and land areas. From August through December cooler sea water temperatures are predominant off the New South Wales coast, while solar radiation increases to its maximum level. The contrast between sea and land surface temperatures peaks in

[‡] Adapted from [142]. With permission.

© 2004 by Chapman & Hall/CRC Press LLC

mid-afternoon. As warmed air rises over the land, a local circulation current begins, drawing cool air in from the sea. The ascending warm air returns seaward (see Figure 11.5), increasing the momentum of the cycle and spreading the effect over a greater area.

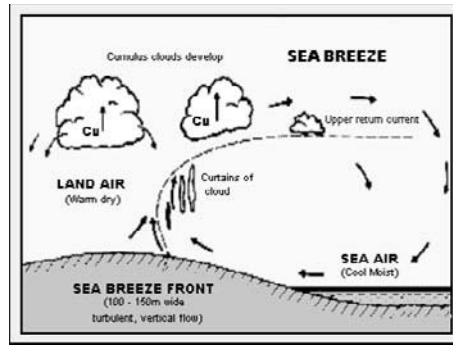


FIGURE 11.5
Seabreeze development cycle.

If the wind currents are weak, a seabreeze will usually commence soon after the land temperature exceeds that of the sea. The seabreeze will increase in strength and reach farther inland as the temperature differential increases. Maximum wind speed (14 to 16 knots) is usually reached several hours after the temperature has peaked. A moderate to strong prevailing offshore wind will delay or, if greater than about 20 knots, prevent a sea breeze from developing [15]. A light to moderate prevailing offshore wind at 900 metres (known as the gradient level) will reinforce seabreezes, which can cause them to reach 30 knots around Sydney. The seabreeze process is also affected by time of day, prevailing weather, seasonal changes and geography [15, 114].

The pre-existing BOM seabreeze forecasting system was the simple rule-based system of Figure 11.6. Its predictions are generated from wind forecasts produced from large-scale weather pattern models. According to BOM, this rule-based system correctly predicts a seabreeze approximately two-thirds of the time.

11.4.2 The data

The BOM provided us with 30MB of data from October 1997 to October 1999, with about 7% of cases having missing attribute values. The data came from three types of sensor sites in the Sydney area. Automatic Weather Stations (AWS) provided ground level wind speed (ws) and direction (wd) at 30 minute intervals. Olympic sites provided ground level wind speed (ws), direction (wd), gust strength, temperature, dew temperature and rainfall. Weather balloon data from Sydney airport (collected at

© 2004 by Chapman & Hall/CRC Press LLC

If	wind component of forecast timeslice is offshore
and	wind component is less than 23 knots
and	part of the forecast timeslice falls in the afternoon,
then	a seabreeze is likely to occur

FIGURE 11.6
Existing rule-based system pseudo code.

5am and 11pm daily) provided vertical readings for gradient-level wind speed (gws), direction (gwd), temperature and rainfall. The variables of predictive interest are: ground level wind speed and direction (ws, wd) and gradient level wind speed and direction (gws, gwd), since a **seabreeze** is defined as occurring when and only when the gradient wind direction is offshore and the ground level wind onshore.

11.4.3 Bayesian network modeling

Regardless of how the Bayesian models of seabreezes were constructed, whether with causal discovery programs or by expert elicitation, we used Netica to parameterize them and to condition them for predictive testing. Netica learns parameters by counting combinations of variable occurrences in the data, using Algorithm 7.1. The number of cases available for the AWS and Olympic sites was large enough to obtain precise parameter estimates, but the airport site data was fairly sparse, leading to weaker predictive results.

11.4.3.1 Expert elicitation

The first technique we used for network construction was elicitation, using meteorologists at the BOM, leading to the causal model of Figure 11.7 (d). The links report causal relationships between the wind to be predicted and the current wind, the time of day and the month of the year.

11.4.3.2 Causal discovery

We applied both TETRAD II and CaMML (introduced, respectively, in Chapters 6 and 8) to the learning of seabreeze models, using all three data sets to find causal structures before parameterizing them with Netica.

As a constraint-based learner, using the PC Algorithm 6.2, TETRAD II does not always specify the direction of a link between nodes. However, it is possible to specify a temporal ordering of variables. Therefore, we generated two networks with TETRAD II for each data set: the first directly from the data and the second including the partial temporal ordering suggested by the expert elicited models. For the AWS data, this resulted in the models of Figure 11.7 (b) and (c). Since CaMML applies a Bayesian metric which aims at finding the highest posterior probability causal model, with all arcs directed, we opted to supply no prior domain knowl-

© 2004 by Chapman & Hall/CRC Press LLC

edge to CaMML. For that reason we also decided to run TETRAD II without any prior domain knowledge, so that we could obtain a legitimate experimental comparison of TETRAD II's performance with that of CaMML. Comparison using identical data sets was somewhat hampered by TETRAD II's inability to accept raw data sets greater than 32,000 cases. This limit was circumvented by generating the correlation matrix from data sets and passing this to TETRAD II (which can operate either from the raw data or from such a correlation matrix).

11.4.3.3 Comparison of TETRAD II and CaMML networks

Without the prior temporal information, TETRAD II produced networks which implied some directed cycles. Whether or not using prior information, TETRAD II networks left many arcs undirected. Therefore, we manually selected directions to resolve the ambiguities and inconsistencies based on knowledge gained during the expert elicitation stage. CaMML networks were often identical those discovered by TETRAD II models (after manual repair), but also they often differed by having more arcs than those of TETRAD II.

The Bayesian networks generated for the airport variables are shown in Figure 11.7. The machine learned networks are fairly similar, although MML tends to find more causal connections than TETRAD II (a result consistent with [65]). The expert network is notably different, discounting any influence of current wind speed and direction measurements.

11.4.4 Experimental evaluation

Here we consider the performance of the different tools in the sea breeze prediction task. First, we compare the BNs with the existing rule-based predictor provided by BOM, and then we compare the different BNs against each other in a variety of ways. The basic result is that the Bayesian networks as a class clearly outperform the rule-based system, while they were effectively indistinguishable from each other, whether they were generated by elicitation or by either causal discovery program. These results held whether the performance metric used was predictive accuracy or information reward. Here we report the results for predictive accuracy only; further details can be found in [143].

11.4.4.1 Test methods

Most of the results reported here used repeated 80-20 splits of data from 1997 and 1998, with 80% of the data applied for training and 20% for testing. The data splits were done randomly 15 times and confidence intervals computed; in general, differences in accuracy of more than 10% were found to be statistically significant at the 0.05 level, which is some indication that the differences are real.

© 2004 by Chapman & Hall/CRC Press LLC

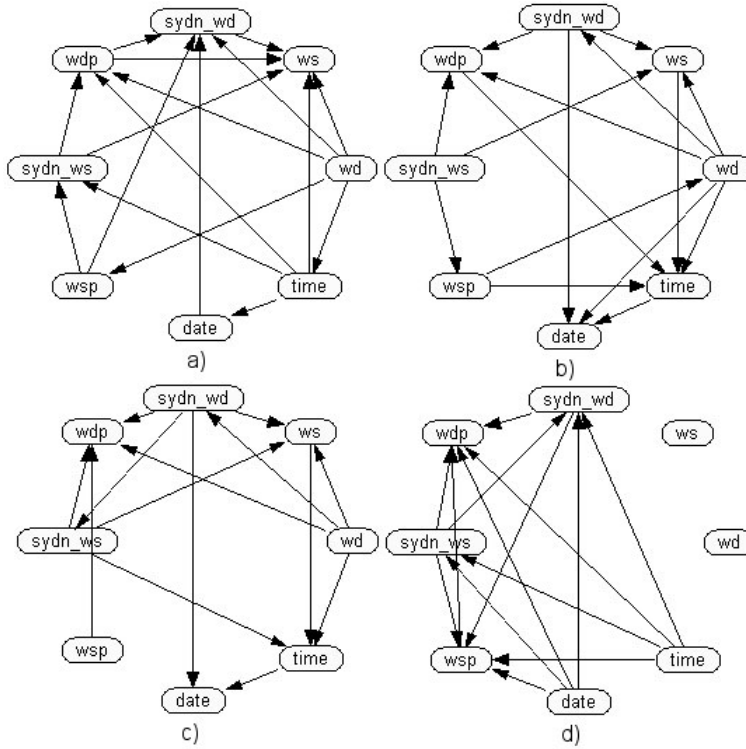


FIGURE 11.7

BN for airport data: (a) CaMML; (b) TETRAD II without prior information; (c) TETRAD II with prior temporal ordering; (d) expert elicitation.

11.4.4.2 Predictive accuracy of the BOM rule-based system

The BOM had estimated that the predictive accuracy of their rule-based system (RB) was around 67 %, but more detailed statistics were unavailable. Our test results were roughly consistent with this. In our testing, we assumed that the prevailing gradient wind pattern would stay constant for the period of the test, which is an approximation whose accuracy declines with the lookahead time.

Weather balloons providing usable data were generally launched from Sydney airport twice a day, at approximately 5am and 11pm. Predictions were made with a lookahead time in increments of three hours. Predictions were generated for each of the AWS sites. The mean accuracy of coastal and inland seabreezes over the different AWS sites is reported in Figure 11.8 (for 5am based predictions only; the 11pm based predictions are similar). A more interesting and difficult task than predicting the mere existence of seabreezes is the prediction of wind direction, which is reported every 30 minutes by the AWS. The RB system was easily adapted to this task and the results are again in Figure 11.8, reported as coastal and inland wind direction

© 2004 by Chapman & Hall/CRC Press LLC

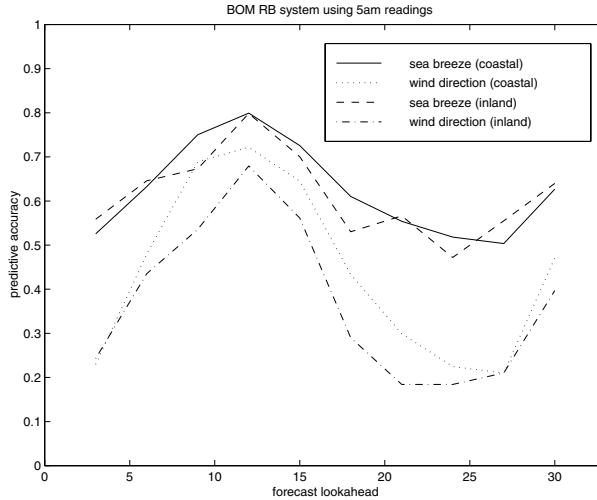


FIGURE 11.8
Comparison of RB predictive accuracy for all sites using 5am readings across lookahead times (in hours).

average accuracies.

As might be expected, the predictive accuracy of the system varied in a rough sine pattern, with a maximum reached at about 4pm, at an accuracy of 80%. Wind direction accuracy was 10 to 20% lower. When comparing the geographic location of individual AWS sites with the predictive accuracy, we can see that sites on the coastline outperform those inland. Presumably, a seabreeze on the coast would not always be of sufficient strength to penetrate inland, making prediction harder.

11.4.4.3 Predictive accuracy of the Bayesian networks

We tested four Bayesian networks for predictive accuracy (see Figure 11.7), one discovered by MML, two by TETRAD II and one elicited from experts. The wind direction predictive accuracy results for these four BNs, together with the BOM RB system, are given in Figure 11.9. It is clear that, with the exception of the earliest 3 hour prediction, the differences between the BNs are not statistically significant, while the simple BOM RB system is clearly underperforming all of the Bayesian networks.

In general, the Bayesian network predictive performance was maximal (up to 80% accuracy) at lookahead times which were multiples of 12 hours, corresponding to late afternoon or early morning. Clearly, there is a strong periodicity to this prediction problem. In the future, such periodicity could be explicitly incorporated into models using MML techniques (e.g., in selecting parameters for a sine function). The predictive differences between BN models appear not to be interesting here. The real

© 2004 by Chapman & Hall/CRC Press LLC

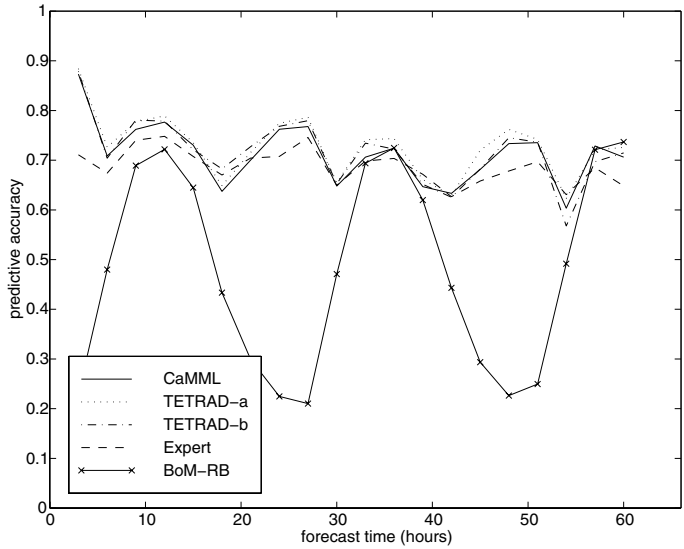


FIGURE 11.9
Wind direction predictive accuracy for airport-type networks.

differences were in usability. The expert network took considerable effort to elicit and build; while TETRAD II either required additional prior information (temporal constraints) or else hand-made posterior edits in comparison with CaMML.

11.4.4.4 Incremental parameter learning

The training method examined thus far has the drawback that both the structure of the model and its parameters are learned in batch mode, with predictions generated from a fixed, fully specified model. Since weather systems change over time, we thought a better approach might be to learn the causal structure in batch mode with 1997 data, but to reparameterize the Bayesian network incrementally using 1998 data, applying fading (see §9.4.1) through a time decay factor so as to favor more recent over older data. The (unnormalized) weight applied to data for incremental updating of the network parameters was optimized by a greedy search, which resulted in a weight applied of $e^{-t0.05}$ where t is the number of sample cases since the data were measured. This is an example of adaptation (see §9.4).

Figure 11.10 shows the average performance of the MML Bayesian networks when incrementally reparameterized. The improvement in predictive accuracy for the AWS and Olympic data sets is statistically significant at the 0.05 level, despite the fact that the average scores reported here are themselves presumably suboptimal, since the predictions made early in the year use parameters estimated from small data sets. Incremental reparameterization in those two cases has also had the curious effect of smoothing out the sine wave variation in predictive performance.

© 2004 by Chapman & Hall/CRC Press LLC

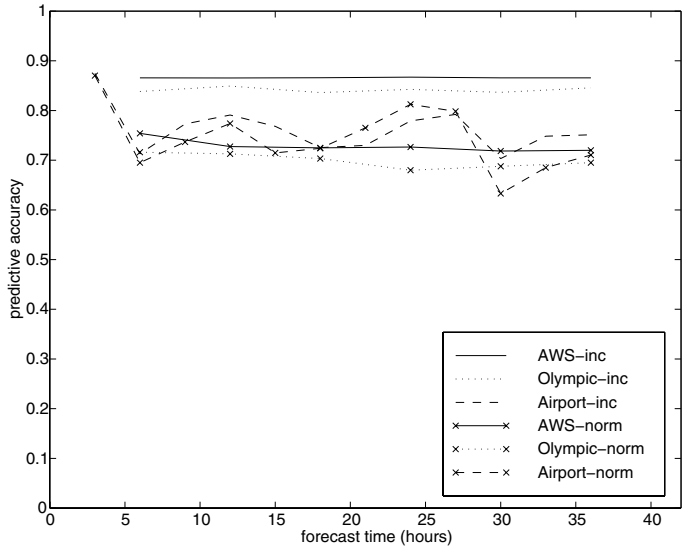


FIGURE 11.10
Comparison of incremental and normal training methods for MML networks.

11.4.5 KEBN aspects

In this seabreeze study we combined a number of KEBN activities. We compared elicited with discovered causal models of seabreezes. We used a large weather data base for automated structure and parameter learning and also for evaluating the learned and elicited Bayesian networks. We also employed parameter adaptation, using fading to get parameters to track changes in weather patterns.

The seabreeze study provides a useful example of the application of Bayesian network technology in data mining problems. There are two basic approaches to deploying Bayesian networks, namely eliciting Bayesian networks from domain experts and using machine learning programs to learn them from data. The initial rule-based predictive system, built in-house by BOM, was shown to be inferior to all of the Bayesian networks developed in this study. The Bayesian network elicited from the domain experts performed on a par with those generated automatically by data mining. Nevertheless, data mining with Bayesian networks shows itself to be a very promising alternative, performing as well as the elicited network and so offering a good alternative for similar problems where human expertise is unavailable. Furthermore, the adaptive parameterization outperformed the static Bayesian networks however they were generated and provides one model for combining elicitation with automated learning.

© 2004 by Chapman & Hall/CRC Press LLC

11.5 Summary

The three case studies demonstrate most of the main knowledge engineering features presented in the previous two chapters. The poker case study illustrates the spiral model of prototype development. All the poker structures were hand-crafted, and their evaluation was based in experimental use against both automated and human opponents. In the ITS case study both the structure and parameters were initially elicited from experts, with automated methods used to validate aspects of the model, using data that was only available for the DCT part of the system. Much more data was available for the final seabreeze prediction case study, which allowed both parameter and structure learning to be performed, as well as parameter adaptation. Consequently, the domain experts had a smaller role in this case study, compared to the first two, and evaluation consisted only of validation methods based on the data.

Our experiences in these, and other, projects were useful in the development of our ideas for the KEBN process presented in this text. KEBN, in turn, should feed into our future Bayesian network projects, resulting in better focused application efforts.

© 2004 by Chapman & Hall/CRC Press LLC

Appendix A

Notation

This appendix provides a list of notational symbols and acronyms, together with page numbers where they are defined or first used.

$MB(h,e)$	degree of increased belief	p. 5
$MD(h,e)$	degree of increased disbelief	p. 5
$CF(h,e)$	certainty factor	p. 5
$A \perp\!\!\!\perp B$	A is independent of B	p. 7
$A \perp\!\!\!\perp B \mid C$	A is independent of B given C	p. 7
x_i	ith state of variable X	p. 9
Ω_X	state space of variable X	p. 9
$\neg(A \perp\!\!\!\perp C \mid B)$	A is dependent upon B given C	p. 41
α	normalizing constant	p. 54
$\lambda(x)$	likelihood	p. 54
$\pi(X)$	parameter in message-passing algorithm	p. 57
$\lambda(X)$	parameter in message-passing algorithm	p. 57
$\pi_X(Y)$	message sent from parent Y to child X	p. 58
$\lambda_X(Y)$	message sent from child Y to parent X	p. 58
$E_{U_i \setminus X}$	evidence connected to U_i	p. 58
$E_{Y_j \setminus X}$	evidence connected to Y_j	p. 58
$u_1 \ldots u_n$	an instantiation of $U_1 \ldots U_n$	p. 58
P'	estimated distribution P'	p. 72
$X \leftarrow X + 1$	assignment (used in algorithms)	p. 73
$C(\mathbf{E})$	measure of conflict between evidence $\mathbf{E}$	p. 78
$max_{X_i \in P\mathbf{X}}(f(X_i))$	returns the X_i that gives maximum $f(.)$	p. 103
X_i^t	node for variable X_i at the tth time-slice	p. 105
$\mathbf{E}_{\{1,t\}}$	evidence nodes from first to tth time-slice	p. 107
$\widehat{Bel}(X)$	predicted belief for X	p. 109
a_{ij}	linear coefficient for $X_j \rightarrow X_i$	p. 155
μ_i	mean of X_i	p. 155
σ_i	standard deviation of X_i	p. 155
p_{ij}	path coefficient for $X_j \rightarrow X_i$	p. 155
r_{ij}	sample correlation between X_j and X_i	p. 156
Φ_k	active path	p. 157
$v(\Phi_k)$	valuation of an active path	p. 157
θ_i	parameterization of model i	p. 165
$\rho_{XY \cdot \mathbf{S}}$	partial correlation between X and Y, S fixed	p. 167
S_{XY}	sample covariance between X and Y	p. 171
S_X	sample standard deviation of X	p. 171
$r_{XY \cdot Z}$	sample partial correlation	p. 171

© 2004 by Chapman & Hall/CRC Press LLC

$D[\alpha_1, \ldots, \alpha_i, \ldots, \alpha_\tau]$	Dirichlet distribution with τ parameters	p. 179
$\vec{\theta}$	the vector $< \theta_1, \ldots, \theta_\tau >$	p. 179
$\pi(X_k)$	parent set of X_k	p. 198
Φ_k	instantiations of parent set of X	p. 199
α_{kjl}	number of matching sample cases	p. 199
$H(X)$	entropy of X	p. 204
$H(X_i, \pi(i))$	mutual information between X and $\pi(X)$	p. 204
$\phi(i)$	an instantiation of $\pi(X_i)$	p. 204
$N(0, \sigma_j)$	Normal (Gaussian) distribution	p. 210
χ^2	the χ^2 distribution	p. 218
IR_G	Good's information reward	p. 280
IR_B	Bayesian information reward	p. 282

Acronyms

BN	Bayesian network	p. 29
CPT	conditional probability table	p. 32
I-map	Independence-map	p. 33
D-map	Dependence-map	p. 33
LS	logic sampling	p. 72
LW	likelihood weighting	p. 74
KL	Kullback-Leibler divergence	p. 76
MPE	most probable explanation	p. 78
EU	expected utility	p. 104
EB	expected benefit	p. 104
DBN	dynamic Bayesian network	p. 104
DDN	dynamic decision network	p. 110
BPP	Bayesian poker player	p. 124
EW	expected winnings	p. 127
NAG	Nice Argument Generator	p. 127
EM	expectation maximization	p. 185
MML	minimum message length	p. 201
MDL	minimum description length	p. 201
TOM	totally ordered model	p. 209
GA	genetic algorithm	p. 211
KEBN	Knowledge Engineering with Bayesian Networks	p. 225
VE	the Verbal Elicitor software package	p. 243
QPN	qualitative probabilistic networks	p. 250
OOBN	object-oriented Bayesian networks	p. 250
ITS	intelligent tutoring system	p. 290
BOM	Bureau of Meteorology	p. 305
AWS	automatic weather station	p. 306

© 2004 by Chapman & Hall/CRC Press LLC

Appendix B

Software Packages

B.1 Introduction

The rapid development of Bayesian network research over the past 15 years has been accompanied by a proliferation of BN software tools. These tools have been built to support both these research efforts and the applications of BNs to an ever-widening range of domains. This appendix is intended as a resource guide to those software tools.

We incorporate Kevin Murphy's listing of software packages for Bayesian networks and graphical models (§B.3), which has been built up and maintained over a number of years*. We also note other Web sites for BN software and other resources.

Next, we describe some of the major software packages — those with the most functionality, or with a particular feature of interest — in more detail. Most of these packages have a long list of features that we cannot even list here, so our survey is by no means exhaustive. We endeavor to point out particular features that relate to issues we have raised earlier in this text. Note that we have personal experience, through teaching, research projects or application development, with the following software: BNT, BUGS, CABeN, CaMML, Hugin, IDEAL, Netica, SMILE and TETRAD II.

In general, all the packages with GUIs include the advanced GUI features (e.g., menu options, short-cut icons, drag-and-drop, online help) that have become the norm in recent years. In this resource review, we will generally ignore GUI aspects, unless there is some feature that stands out. Instead, we will concentrate on aspects of the functionality.

We make no attempt to give any sort of ranking of these packages; so our survey presents the packages alphabetically. The Murphy listing notes whether the products are free or commercial but available in a restricted form. Otherwise we do not make any comments on the cost of commercial products.

*Our thanks to Kevin Murphy for giving us permission to use this listing.

© 2004 by Chapman & Hall/CRC Press LLC

B.2 History

The development of the first BN software, beyond algorithm implementation, occurred concurrently with the surge of BN research in the late 1980s. Hugin [6] was developed at the University of Aalborg in Denmark (see §B.4.5 below) and went on to become a commercial product now widely used. The Lisp-base IDEAL (**I**nfluence **D**iagram **E**valuation and **A**nalysis in **L**isp) test-bed environment was developed at Rockwell [266][†].

Another early BN inference engine was CABeN (a **C**ollection of **A**lgorithms for **Be**lief **N**etworks) [58], which contains a library of routines for different stochastic simulation inference algorithms. Lumina Decision Systems, Inc., was founded in 1991 by Max Henrion and Brian Arnold, which produce Analytica. The development of Netica, now produced by Norsys Ltd, was started in 1992.

B.3 Murphy's Software Package Survey

http://www.ai.mit.edu/~murphyk/Software/BNT/bnsoft.html

This survey is given in Tables B.1, B.2 and B.3. The first two tables gives the software package name, its producers and where it is available online. If the software is commercial but the company has links with particular institutions or BN researchers, those are also noted. Table B.3 covers basic feature information such as technical information about source availability, platforms, GUI and API, very high level functionality such as types of nodes supported (i.e., discrete and/or continuous), whether decision networks are supported, whether undirected graphs are supported. It includes whether the software allows learning (parameters and/or structure), describes the main inference algorithms (if this information is available) and indicates whether the software is free or commercial. The details of the meaning of each column are given in Figure B.1.

Google's list of tools is available at:

http://directory.google.com/Top/Computers/Artificial_Intelligence/Belief_Networks/Software/

The Bayesian Network Repository contains examples of BNs, plus datasets for learning them:

http://www.cs.huji.ac.il/labs/compbio/Repository/

[†] One of the authors used it for her Ph.D. research.

© 2004 by Chapman & Hall/CRC Press LLC

Src Is the source code included? **N=no**. If yes, what language? **J** = Java, **M** = Matlab, **L** = Lisp, **C**, **C++**, **R**, **A** = APL.

API Is an application program interface included?

N means the program cannot be integrated into your code, i.e., it must be run as a standalone executable. **Y** means it can be integrated.

Exec The **executable** runs on: **W** = Windows (95/98/2000/NT), **U** = Unix, **M** = Mac-Intosh, **-** = Any machine with a compiler.

GUI Is a **G**raphical **U**ser **I**nterface included? **Y**=Yes,**N**=No.

D/C Are continuous-valued nodes supported (as well as discrete)? **G** = (conditionally) Gaussians nodes supported analytically, **Cs** = continuous nodes supported by sampling, **Cd** = continuous nodes supported by discretization, **Cx** = continuous nodes supported by some unspecified method, **D** = only discrete nodes supported.

DN Are decision networks/influence diagrams supported? **Y**=Yes,**N**=No.

Params Does the software functionality include parameter learning? **Y**=Yes,**N**=No.

Struct Does the software functionality include structure learning? **Y**=Yes,**N**=No.

CI means **Y**, using conditional independency tests (see §6.3)

K2 means **Y**, using Cooper & Herskovits' K2 algorithm (see §8.2)

D/U What kind of graphs are supported? **U** = only undirected graphs, **D** = only directed graphs, **UD** = both undirected and directed, **CG** = chain graphs (mixed directed/undirected).

Inf Which inference algorithm is used? (See Chapter 3)

JT = **Junction Tree**, **VE** = **variable (bucket) elimination**, **PT** = **Pearl's polytree**, **E** = **Exact inference** (unspecified), **MH** = **Metropolis Hastings**, **MC** = **Markov chain Monte Carlo (MCMC)**, **GS** = **Gibbs sampling**, **IS** = **Importance sampling**, **S** = **Sampling**, **O** = **Other** (usually special purpose), **++** = **Many** methods provided, **?** = Not specified, **N** = None, the program is only designed for structure learning from completely observed data.

NB: Some packages support a form of sampling (e.g., **likelihood weighting**, **MDMC**), in addition to their exact algorithm; this is indicated by **(+S)**.

Free Is a free version available? **O**=Free (though possibly only for academic use), **$** = Commercial (although most have free versions which are restricted in various ways, e.g., the model size is limited, or models cannot be saved, or there is no API.)

FIGURE B.1
Description of features used in Murphy's BN software survey, in Table B.3.

© 2004 by Chapman & Hall/CRC Press LLC

TABLE B.1
Software packages: name, Web location and developers (part I)

Name	Web Location	Authors
Analytica	http://www.lumina.com	Lumina (Henrion)
Bassist	http://www.cs.Helsinki.FI/research/fdk/bassist	U. Helsinki
Bayda	http://www.cs.Helsinki.FI/research/cosco/Projects/NONE/SW/	U. Helsinki
BayesBuilder	http://www.mbfys.kun.nl/snn/Research/bayesbuilder/	Nijman (U. Nijmegen)
BayesiaLab	http://www.bayesia.com	Bayesia Ltd
Bayesware Discoverer	http://www.bayesware.com	Bayesware (Open Univ., UK)
B-course	http://b-course.cs.helsinki.fi	U. Helsinki
BN power constructor	http://www.cs.ualberta.ca/~jcheng/bnpc.htm	Cheng (U.Alberta)
BNT	http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html	Murphy (prev U.C.Berkeley, now MIT)
BNJ	http://bndev.sourceforge.net/	Hsu (Kansas)
BucketElim	http://www.ics.uci.edu/~irinar	Rish (U.C.Irvine)
BUGS	http://www.mrc-bsu.cam.ac.uk/bugs	MRC/Imperial College
Business Navigator 5	http://www.data-digest.com	Data Digest Corp
CABeN	http://www-pcd.stanford.edu/cousins/caben-1.1.tar.gz	Cousins et al. (Wash. U.)
CaMML	http://www.datamining.monash.edu.au/software/camml	Wallace, Korb (Monash U.)
CoCo+Xlisp	http://www.math.auc.dk/~jhb/CoCo/information.html	Badsberg (U. Aalborg)
CIspace	http://www.cs.ubc.ca/labs/lci/CIspace/	Poole et al. (UBC)
Deal	http://www.math.auc.dk/novo/deal	Bottcher et al.
Ergo	http://www.noeticsystems.com	Noetic systems
First Bayes	http://www.shef.ac.uk/~st1ao/1b.html	U. Sheffield
GDAGsim	http://www.staff.ncl.ac.uk/d.j.wilkinson/software/gdagsim/	Wilkinson (U. Newcastle)
GMRFsim	http://www.math.ntnu.no/~hrue/GMRFsim/	Rue (U. Trondheim)
GeNIe/SMILE	http://www.sis.pitt.edu/~genie	U. Pittsburgh (Druzdzel)
GMTk	http://ssli.ee.washington.edu/~bilmes/gmtk/	Bilmes (UW), Zweig (IBM)
gR	http://www.r-project.org/gR	Lauritzen et al.

© 2004 by Chapman & Hall/CRC Press LLC

TABLE B.2
Software packages: name, Web location and developers (part II)

Name	Web Location	Authors
Grappa	http://www.stats.bris.ac.uk/~peter/Grappa/	Green (Bristol)
Hugin	http://www.hugin.com	Hugin Expert (U. Aalborg, Lauritzen/Jensen)
Hydra	http://software.biostat.washington.edu/statsoft/MCMC/Hydra	Warnes (U.Wash.)
Ideal	http://yoda.cis.temple.edu:8080/ideal/	Rockwell (Srinivas)
Java Bayes	http://www.cs.cmu.edu/~javabayes/Home/	Cozman (CMU)
MIM	http://www.hypergraph.dk/	HyperGraph Software
MSBNx	http://research.microsoft.com/adapt/MSBNx/	Microsoft
Netica	http://www.norsys.com	Norsys (Boerlage)
PMT	http://people.bu.edu/vladimir/pmt/index.html	Pavlovic (BU)
PNL	http://www.ai.mit.edu/ murphyk/Software/PNL/pnl.html	Eruhimov (Intel)
Pulcinella	http://iridia.ulb.ac.be/pulcinella/Welcome.html	IRIDIA
RISO	http://sourceforge.net/projects/riso	Dodier (U.Colorado)
TETRAD	http://www.phil.cmu.edu/tetrad/	CMU Philosophy
UnBBayes	http://sourceforge.net/projects/unbbayes/	?
Vibes	http://www.inference.phy.cam.ac.uk/jmw39/	Winn & Bishop (U. Cambridge)
Web Weaver	http://snowhite.cis.uoguelph.ca/faculty_info/yxiang/ww3/	Xiang (U.Regina)
WinMine	http://research.microsoft.com/~dmax/WinMine/tooldoc.htm	Microsoft
XBAIES 2.0	http://www.staff.city.ac.uk/~rgc/webpages/xbpage.html	Cowell (City U.)

© 2004 by Chapman & Hall/CRC Press LLC

TABLE B.3
Murphy's feature comparison of software packages

Name	Src	API	Exec	GUI	D/C	DN	Params	Struct	D/U	Infer	Free
Analytica	N	Y	WM	Y	G	Y	N	N	D	S	$
Bassist	C++	Y	U	N	G	N	Y	N	D	MH	O
Bayda	J	Y	WUM	Y	G	N	Y	N	D	?	O
BayesBuilder	N	N	W	Y	D	N	N	N	D	?	O
BayesiaLab	N	N	-	Y	Cd	N	Y	Y	CG	JT,G	$
Bayesware	N	N	WUM	Y	Cd	N	Y	Y	D	?	$
B-course	N	N	WUM	Y	Cd	N	Y	Y	D	?	O
BNPC	N	Y	W	Y	D	N	Y	CI	D	?	O
BNT	M/C	Y	WUM	N	G	Y	Y	Y	UD	S,E(++)	O
BNJ	J	Y	-	Y	D	N	N	Y	D	JT,IS	O
BucketElim	C++	Y	WU	N	D	N	N	N	D	VE	O
BUGS	N	N	WU	Y	Cs	N	Y	N	D	GS	O
BusNav	N	N	W	Y	Cd	N	Y	Y	D	JT	$
CABeN	C	Y	WU	N	D	N	N	N	D	S(++)	O
CaMML	N	N	U	N	Cx	N	Y	Y	D	N	O
CoCo+Xlisp	C/L	Y	U	Y	D	N	Y	CI	U	JT	O
CIspace	J	N	WU	Y	D	N	N	N	D	VE	O
Deal	R	-	-	Y	G	N	N	Y	D	N	O
Ergo	N	Y	WM	Y	D	N	N	N	D	JT(+S)	$
First Bayes	A	N	W	Y	-	N	N	N	–	O	O
GDAGsim	C	Y	WUM	N	G	N	N	N	D	E	O
GeNIe/SMILE	N	Y	WU	Y	D	Y	N	N	D	JT(+S)	O
GMRFsim	C	Y	WUM	N	G	N	N	N	U	MC	O
GMTk	N	Y	U	N	D	N	Y	Y	D	JT	O
gR	R	-	-	-	-	-	-	-	-	-	O
Grappa	R	Y	-	N	D	N	N	N	D	JT	O
Hugin	N	Y	WU	Y	G	Y	Y	CI	CG	JT	$
Hydra	J	Y	-	Y	Cs	N	Y	N	UD	MC	O
Ideal	L	Y	WUM	Y	D	Y	N	N	D	JT	0
Java Bayes	J	Y	WUM	Y	D	Y	N	N	D	JT,VE	O
MIM	N	N	W	Y	G	N	Y	Y	CG	JT	$
MSBNx	N	Y	W	Y	D	Y	N	N	D	JT	O
Netica	N	Y	WUM	Y	G	Y	Y	N	D	JT	$
PMT	M/C	Y	-	N	D	N	Y	N	D	O	O
PNL	C++	Y	-	N	D	N	Y	Y	UD	JT	O
Pulcinella	L	Y	WUM	Y	D	N	N	N	D	?	O
RISO	J	Y	WUM	Y	G	N	N	N	D	PT	O
TETRAD IV	N	N	WU	Y	Cx	N	Y	CI	UD	N	O
UnBBayes	J	Y	-	Y	D	N	N	Y	D	JT	O
Vibes	J	Y	WU	Y	Cx	N	Y	N	D	?	O
Web Weaver	J	Y	WUM	Y	D	Y	N	N	D	?	O
WinMine	N	N	W	Y	Cx	N	Y	Y	UD	N	O
XBAIES 2.0	N	N	W	Y	G	Y	Y	Y	CG	JT	O

© 2004 by Chapman & Hall/CRC Press LLC

B.4 BN software

In this section we review some of the major software packages for Bayesian and decision network modeling and inference. The information should be read in conjunction with the feature summary in Table B.3. The additional aspects we consider are as follows.

Development: Any background information of the developers or the history of this software.

Technical: Further information about platforms or products (beyond the summary given in Table B.3).

Node Types: Relating to discrete/continuous support (see §9.3.1.4).

CPTs: Support for elicitation (§9.3.3), or local structure (§7.4, §9.3.4).

Inference: More details about the inference algorithm(s) provided, and possible user control over inference options (Chapter 3). Also whether computes MPE and P(E) (§3.7).

Evidence: Whether negative and likelihood evidence are supported, in addition to specific evidence (§3.4).

Decision networks: Information about decision network evaluation (if known), whether expected utilities for all policies are provided, or just decision tables (§4.3.4). Whether precedence links between decision nodes are determined automatically if not specified by the knowledge engineer (§4.4). Also, whether value of information is supported directly (§4.4.4).

DBNs: Whether DBN representation and/or inference is supported (§4.5).

Learning: What learning algorithms are used (Chapters 6–8).

Evaluation: What support, if any, for evaluation? (Chapter 10). In particular, sensitivity analysis (§10.3) and statistical validation methods (§10.5).

Other features: Functionality not found in most other packages.

B.4.1 Analytica

Lumina Decision Systems, Inc.
26010 Highland Way, Los Gatos, CA 95033
http://www.lumina.com

Development: As noted earlier in the history outline, Lumina Decision Systems, Inc., was founded in 1991 by Max Henrion and Brian Arnold. The emphasis in Analytica is on using influence diagrams as a statistical decision support tool. Analytica does not use Bayesian network terminology, which can lead to difficulties in identifying aspects of its functionality.

© 2004 by Chapman & Hall/CRC Press LLC

Technical: Analytica 2.0 GUI is available for Windows and Macintosh. The Analytica API (called the Analytica Decision Engine) is available for windows 95/98 or NT 4.0 and runs in any development environment with COM or Automation support.

CPTs: Analytica supports many continuous and discrete distributions, and provides a large number of mathematical and statistical functions.

Inference: Analytica provides basic MDMC sampling, plus median Latin hypercube (the default method) and random Latin hypercube and allows the sample size to be set. The Analytica GUI provides many ways to view the results of inference, through both tables and graphs: statistics, probability bands, probability mass (the standard for most other packages), cumulative probability and the actual samples generated by the inference.

Evidence: Specific evidence can only be entered for variables previously set up as "input nodes."

DBNs: Analytica provides dynamic simulation time periods by allowing the user to specify both a list of time steps and which variables change over time. Note: Analytica does not use DBN terminology or show the "rolled-out" network.

Evaluation: Analytica provides what it calls "importance analysis," which is an absolute rank-order correlation between the sample of output values and the sample for each uncertain input. This can be used to create so-called importance variables. Analytica also provides a range of sensitivity analysis functions, including "whatif" and scatterplots.

Other features: Analytica supports the building of large models by allowing the creation of a hierarchical combination of smaller models, connected via specified input and output nodes.

B.4.2 BayesiaLab

BAYESIA
6, rue Lonard de Vinci - BP0102,53001 Laval Cedex, France
http://www.bayesia.com

Technical: BayesiaLab GUI is available for all platforms supporting JRE. There is also a product "BEST," for using BNs for diagnosis and repair.

Node Types: Continuous variables must be discretized. When learning variables from a database, BayesiaLab supports equal distance intervals, equal frequency intervals and a decision tree discretization that chooses the intervals depending on the information they contribute to a specified target variable.

CPTs: BayesiaLab supports entry of the CPT through normalization, completing entries, offering multiple entries and will also generate random entries.

Inference: Inference in BayesiaLab is done in what they call "validation" mode (compared to the modeling mode for changing the network). A junction tree algorithm is the default used, with a MDMC Gibbs sampling algorithm available.

© 2004 by Chapman & Hall/CRC Press LLC

Evidence: Only specific evidence supported (entered through the so-called "monitor").

Learning: BayesiaLab does parameter learning using a version of the Spiegelhalter & Lauritzen parameterization algorithm. It has three methods for structural learning (which they call "association discovery"): SopLEQ, which uses properties of equivalent Bayesian networks and two versions of Taboo search. BayesiaLab also provides clustering algorithms for concept discovery.

Evaluation: BayesiaLab provides a number of functions (with graphical display support) for evaluating a network, including the strength of the arcs (used also by the automated graph layout), the amount of information brought to the target node, the type of probabilistic relation, generation of analysis reports, causal analysis (based on an idea of "essential graphs," showing arcs that can't be reversed without changing the probability distribution represented) and arc reversal. It supports simulation of "What-if" scenarios and provides sensitivity analysis (through their so-called "adaptive questionnaires"), lift curves and confusion matrices.

Other features: BayesiaLab supports hidden variables, importing from a database, and export a BN for use by their troubleshooting product, BEST.

B.4.3 Bayes Net Toolbox (BNT)

Kevin Murphy
MIT AI lab,#200 Technology Square, Cambridge, MA 02139
http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html

Development: This package was developed during Kevin Murphy's time at U.C. Berkeley as a Ph.D. student, where his thesis [198] addressed DBN representation, inference and learning. He also worked on BNT while an intern at Intel.

Technical: The Bayes Net Toolbox is for use only with Matlab, a widely used and powerful mathematical software package. Its lack of a GUI is made up for by Matlab's visualization features. This software is distributed under the Gnu Library General Public License.

CPTs: BNT supports the following conditional probability distributions: multinomial, Gaussian, Softmax (logistic/sigmoid), Multi-layer perceptron (neural network), Noisy-or, Deterministic.

Inference: BNT supports many different exact and approximate inference algorithms, for both ordinary BNs and DBNs, including all the algorithms described in this text.

DBNs: The following dynamic models can be implemented in BNT: Dynamic HMMs, Factorial HMMs, coupled HMMs, input-output HMMs, DBNs, Kalman filters, ARMAX models, switching Kalman filters, tree-structured Kalman filters, multiscale AR models.

© 2004 by Chapman & Hall/CRC Press LLC

Learning: BNT parameter learning methods are: (1) Batch MLE/MAP parameter learning using EM (different M and E methods for each node type); (2) Sequential/batch Bayesian parameter learning (for tabular nodes only).

Structure learning methods are: (1) Bayesian structure learning, using MCMC or local search (for fully observed tabular nodes only); (2) Constraint-based structure learning (IC/PC and IC*/FCI).

B.4.4 GeNIe

Decision Systems Laboratory, University of Pittsburgh
B212 SLIS Building, 135 North Bellefield Avenue, Pittsburgh, PA 15260, USA
http://www.sis.pitt.edu/~genie/

Development: Developed by Druzdzel's decision systems group, GeNIe's support of decision networks, in addition to BNs, reflects their teaching and research interests in decision support and knowledge engineering. GeNIe 1.0 was released in 1998, and GeNIe 2.0 is due for release in mid-2003.

Technical: GeNIe (**G**raphical **N**etwork **I**nterface) is a development environment for building decision networks, running under Windows. SMILE (Structural Modeling, Reasoning and Learning Engine) is its portable inference engine, consisting of a library of C++ classes currently compiled for Windows, Solaris and Linux. GeNIe is an outer shell to SMILE. Here we focus on describing GeNIe.

CPTs: Supports chance nodes with General, Noisy OR/MAX and Noisy AND distribution, as well as graphical elicitation of probabilities.

Inference: GeNIe's default BN inference algorithm is the junction tree clustering algorithm; however a polytree algorithm is also available, plus several approximate algorithms that can be used if the networks get too large for clustering (logic sampling, likelihood weighting, self importance and heuristic importance sampling, backwood sampling). GeNIe 2.0 provides more recent state-of-the-art sampling algorithms, AIS-BN [43] and EPIS-BN [304].

Evidence: Only handles specific evidence.

Decision networks: GeNIe offers two decision network evaluation algorithms: a fast algorithm [249] that provides only the best decision and a slower algorithm that use an inference algorithm to evaluate the BN part of the network, then computes the expected utility for all possible policies.

If the user does not specify the temporal order of the decision nodes, it will try to infer it using causal considerations; otherwise it will decide an order arbitrarily. To simplify the displayed model, GeNIe does not require the user to create temporal arcs, inferring them from the temporal order among the decision nodes.

Viewing results: The value node will show the expected utilities of all combinations of decision alternatives. The decision node will show the expected

© 2004 by Chapman & Hall/CRC Press LLC

utilities of its alternatives, possibly indexed by those decision nodes that precede it.

GeNIe provides the expected value of information, i.e., the expected value of observing the state of a node before making a decision.

Evaluation: GeNIe supports simple sensitivity analysis in graphical models, through the addition of a variable that indexes various values for parameters in question. GeNIe computes the impact of these parameter values on the decision results (showing both the expected utilities and the policy). Using the same index variable, GeNIe can display the impact of uncertainty in that parameter on the posterior probability distribution of any node in the network.

Other features: GeNIe allows submodels and a tree view. It can handle other BN file formats (Hugin, Netica, Ergo). GeNIe provides integration with MS Excel, including cut and paste of data into internal spreadsheet view of GeNIe, and supports for diagnostic case management.

GeNIe also supports what they call "**relevance reasoning**" [172], allowing users to specify nodes that are of interest (so-called target nodes). Then when updating computations are performed, only the nodes of interest are guaranteed to be fully updated; this can result in substantial reductions in computation.

B.4.5 Hugin

Hugin Expert, Ltd
Niels Jernes Vej 10, 9220 Aalborg East, Denmark
http://www.hugin.com

Development: The original Hugin shell was initially developed by a group at the Aalborg University, as part of an ESPRIT project which also produced MUNIN system [9]. Hugin's development continued through another Lauritzen-Jensen project called ODIN. Hugin Expert was established to start commercializing the Hugin tool. The close connection between Hugin Expert and the Aalborg research group has continued, including co-location and personnel moving between the two. This has meant that Hugin Expert has consistently contributed to and taken advantage of the latest BN research. In 1998 Hewlett-Packard purchased 45% of Hugin Expert and established a new independent company called Dezide, which bases their product ("dezisionWorks") on Hugin. Further, Hugin is also a reseller of dezisionWorks under the name "Hugin Advisor".

Technical: The Hugin API is called the "Hugin Decision Engine." It is available for the languages C++, Java and as an ActiveX-server and runs on the operating systems: Sun Solaris (Sparc and x86), HP-UX, Linux and Windows. Versions are available for single and double-precision floating-point operations. The Hugin GUI (called "Hugin") is available for Sun Solaris (sparc, x86) Windows and Linux red-hat.

© 2004 by Chapman & Hall/CRC Press LLC

Hugin also offers "Hugin Advisor" for developing trouble shooting applications and "Hugin Clementine" for integrating Hugin's learning with datamining in SPSS's Clementine system.

Node Types: Good support for continuous variable modeling, and combining discrete and continuous nodes, following on from research in this area [210].

CPTs: CPTs can be specified with expressions as well as through manual entry. The CPTs don't have to sum to one; entries that don't sum to one are normalized.

Inference: The basic algorithm is the junction tree algorithm, with options to choose between variations. The junction tree may be viewed. There is the option to vary the triangulation method, and another to turn on compression (of zeros in the junction tree) (see Problem 3, Chapter 3). An approximate version of the junction tree algorithm is offered, where all probabilities less than a specified threshold are made zero (see Problem 5, Chapter 3).

In addition Hugin GUI computes P(E), the data conflict measure [130, 145], described in §3.7.2, and the MPE.

Evidence: Specific, negative and virtual evidence are all supported.

Decision networks: Hugin requires the existence of a directed path including all decision variables. It gives the expected utility of each decision option in the decision table.

Learning: The parameter learning is done with EM learning (see §7.3.2.2); Spiegelhalter & Lauritzen sequential learning (adaptation) and fading are also supported. Structure learning is done using the PC algorithm (see §6.3.2).

Other features: Supports object-oriented BNs (see §9.3.5.2).

B.4.6 JavaBayes

Fabio Gagliardi Cozman
Escola Politécnica, University of São Paulo
http://www.cs.cmu.edu/~javabayes/Home/
http://www.pmr.poli.usp.br/ltd/Software/javabayes/ (recent versions)

Development: JavaBayes was the first BN software produced in Java and is distributed under the GNU License.

Other features: JavaBayes provides a set of parsers for importing Bayesian networks in several proposed so-called "interchange" formats.

JavaBayes also offers Bayesian **robustness analysis** [62], where sets of distributions are associated to variables: the size of these sets indicates the "uncertainty" in the modeling process. JavaBayes can use models with sets of distributions to calculate intervals of posterior distributions or intervals of expectations. The larger these intervals, the less robust are the inferences with respect to the model.

© 2004 by Chapman & Hall/CRC Press LLC

B.4.7 MSBNx

Microsoft
http://research.microsoft.com/adapt/MSBNx/

CPTs: MSBNx supports the construction of the usual tables, as well as local structure in the form of context-sensitive independence (CSI)[30], (see §9.3.4), and classification trees (see §7.4.3).

Inference: A form of junction tree algorithm is used.

Evidence: Supports specific evidence only.

Evaluation: MSBNx can recommend what evidence to gather next. If given cost information, MSBNx does a cost-benefit analysis; otherwise it makes recommendations based on an entropy-based value of information measure (note: prior to 2001, this was a KL-divergence based measure).

B.4.8 Netica

Norsys Software Corp.
3512 W 23^{rd} Ave., Vancouver, BC,Canada V6S 1K5
http://www.norsys.com

Development: Netica's development was started in 1992, by Norsys CEO Brent Boerlage, who had just finished a Masters degree at the University of British Columbia, where his thesis looked at quantifying and displaying "link strengths" in Bayesian networks [23]. Netica became commercially available in 1995 and is now widely used.

Technical: The Netica API is available for languages C and Java, to run on Mac OSX, Sun Sparc, Linux and Windows. The GUI is available for Mac and Windows. There is also a COM interface for integrating the GUI with other GUI applications and Visual Basic programming.

Node Types: Netica can learn node names from variable names in a data file (called a case file). Netica discretizes continuous variables but allows control over the discretization.

CPTs: There is some support for manual entry of probabilities, with functions for checking that entries sum to 100 (Netica has a default option to use numbers out of 100, rather than probabilities between 0 and 1), automatically filling in the final probability and normalizing. Equations can also be used to specify the CPT, using a large built-in library of functions and continuous and discrete probability distributions, and there is support for noisy-or, noisy-and, noisy-max and noisy-sum nodes.

Inference: Netica's inference is based on the elimination junction tree method (see §3.10). The standard compilation uses a minimum-weight search for a good elimination order, while an optimized compilation option searches for the best

© 2004 by Chapman & Hall/CRC Press LLC

elimination order using a combination of minimum-weight search and stochastic search. Both the junction tree and the elimination order may be viewed. Netica also reports both the probability of the most recent evidence, and the probability of all evidence currently entered, and provides the MPE and its probability (but not for networks containing decision nodes). Netica can generate random samples by junction tree or logic sampling.

Evidence: Netica supports specific (which they call "positive"), negative and likelihood evidence. Multiple likelihood evidence may be incorporated for the same nodes. Netica also handles sets of evidence (cases) by case files and direct database access.

Decision networks: Netica infers a temporal order for decision network, if it can. DN evaluation gives the expected utilities for a one-off decision, but only the decision table for sequential decision making.

DBNs: Netica supports DBN specification and roll-out.

Learning: Netica supports parameter learning only. It uses the Spiegelhalter & Lauritzen parameterization algorithm, allows missing values, and allows the specification of a weighting to the original probabilities, providing a form of adaptation. Netica can also do EM learning and gradient descent learning, to handle large amounts of missing data, or latent (unobserved) variables. It also supports fading, with the user able to specify a factor from 0 (no new learning) to 1 (removes all previous learning).

Evaluation: Netica supports sensitivity to findings, as described earlier in §10.3.1. It also provides a number of measures for statistical validation including a count form of predictive accuracy, a confusion matrix, the error rate, scoring rule results, logarithmic loss and quadratic loss, spherical pay off, calibration results and a "times surprised" table (indicating when the network was confident of its beliefs but was wrong).

B.5 Bayesian statistical modeling

B.5.1 BUGS

MRC/Imperial College
http://www.mrc-bsu.cam.ac.uk/bugs

BUGS (**B**ayesian inference **U**sing **G**ibbs **S**ampling) provides Bayesian analysis of complex statistical models using Markov Chain Monte Carlo (MCMC). It's most advanced version is WinBUGS, which is available in a free educational version. Other versions are available for unix systems (Sun, SGI and Linux). The Web pages provide thorough documentation for the different versions of the program. A BUGS email discussion list is also maintained.

© 2004 by Chapman & Hall/CRC Press LLC

The Markov Chain Monte Carlo (MCMC) methods available include univariate Gibbs sampling (BUGS 0.6) and a more sophisticated univariate Metropolis sampler (WinBUGS). Applications include generalized linear mixed models, latent variable models, modeling measurement error, handling missing data and Bayesian updating.

BUGS provides a language for specifying a Bayesian network. A compiler then processes the model and data and sets up the sampling distributions required for the Gibbs sampling. Finally, appropriate sampling algorithms are implemented to simulate values of the unknown quantities in the model.

B.5.2 First Bayes

Dept. of Probability and Statistics
University of Sheffield
http://www.shef.ac.uk/~st1ao/1b.html

First Bayes is program to assist people in learning elementary Bayesian Statistics. It is meant to *supplement* other educational material and is not a self-contained tutoring system. It runs under MS-Windows. First Bayes is offered free to anyone interested in teaching or learning Bayesian Statistics, provided it is not used for profit.

B.6 Causal discovery programs

B.6.1 Bayesware Discoverer

Bayesware Ltd.
http://www.bayesware.com

Bayesware Discoverer is a new commercial causal discovery tool running on the MS Windows platforms. The program uses the Cooper and Herskovits formula of §8.2 to construct a Bayesian metric, adjusted to deal with incomplete data by employing probability intervals. It has a "Wizard-like" interface to assist.

This software is a commercial version of the Bayesian Knowledge Discoverer, developed at the Knowledge Media Institute of The Open University (UK), based on the work of Ramoni and Sebastiani (e.g., [230]). A limited educational version is available free.

B.6.2 CaMML

Chris Wallace, Kevin Korb
School of Computer Science, Monash University, Victoria 3800 Australia

http://www.datamining.monash.edu.au/software/camml/index.shtml

© 2004 by Chapman & Hall/CRC Press LLC

CaMML (Causal discovery via MML) was described in Chapter 8 in some detail. It is freely available as an executable download for Linux from the above Web site. There are two different versions: CaMML-L, which learns linear Gaussian models, and CaMML, which learns discrete causal models. Both use the Metropolis sampling search algorithm. These versions have a fairly crude ASCII command-line interface. There is a project to reimplement CaMML inside the Monash CDMS (Core Data Mining Software) project, providing CaMML with a GUI interface and data visualization capabilities. When ready, this will be available from the above Web site, as will any future developments from the CaMML project.

B.6.3 TETRAD

Peter Spirtes, Clark Glymour and Richard Scheines
Dept. of Philosophy, Carnegie Mellon University
http://www.phil.cmu.edu/tetrad/

TETRAD II was the first commercially available causal discovery program. Its PC algorithm is described in Chapter 6. It is no longer available for purchase. TETRAD III adds Gibbs sampling to the functionality of TETRAD II, but retains the command-line interface. TETRAD IV has a graphical user interface, running under the Java Runtime Environment. TETRAD IV currently supports four causal discovery algorithms:

- The PC Algorithm
- A genetic algorithm search (for linear dag models)
- The CCD Algorithm (linear cyclic models)
- The FCI Algorithm (dag models with latent variables)

TETRAD IV supports maximum likelihood parameter estimation for both discrete and linear models. It also has support for building and simulating time series models. TETRAD III and TETRAD IV are available for free download.

B.6.4 WinMine

David Max Chickering
Microsoft Research, Redmond, Washington 98052
http://research.microsoft.com/~dmax/WinMine/tooldoc.htm

WinMine is a set of causal discovery programs for Windows 2000/NT/XP [47]. The majority of the programs are command-line executables that can be run in scripts. It includes GUIs for viewing learned or modeled Bayesian networks and for displaying classification trees ("decision trees"). It is freely downloadable, if it is not going to be used for commercial purposes.

WinMine can learn Bayesian networks from sample data. It supports prior information in the form of partial variable orderings and forbidden arcs. There is also support for evaluating the learned models.

© 2004 by Chapman & Hall/CRC Press LLC