Advances in Computer Vision and Pattern Recognition

Luis Enrique Sucar

Probabilistic Graphical Models

Principles and Applications

Springer

Advances in Computer Vision and Pattern Recognition

More information about this series at http://www.springer.com/series/4205

Luis Enrique Sucar

Probabilistic Graphical Models

Principles and Applications

Springer

Luis Enrique Sucar
Instituto Nacional de Astrofísica,
Óptica y Electrónica (INAOE)
Santa María Tonantzintla
Puebla
Mexico

ISSN 2191-6586 ISSN 2191-6594 (electronic)
Advances in Computer Vision and Pattern Recognition
ISBN 978-1-4471-6698-6 ISBN 978-1-4471-6699-3 (eBook)
DOI 10.1007/978-1-4471-6699-3

Library of Congress Control Number: 2015939664

Springer London Heidelberg New York Dordrecht

Printed on acid-free paper

Springer-Verlag London Ltd. is part of Springer Science+Business Media (www.springer.com)

To my family, Doris, Edgar and Diana,
for their unconditional love and support

Foreword

Probabilistic graphical models (PGMs), and their use for reasoning intelligently under uncertainty, emerged in the 1980s within the statistical and artificial intelligence reasoning communities. The Uncertainty in Artificial Intelligence (UAI) conference became the premier forum for this blossoming research field. It was at UAI-92 in San Jose that I first met Enrique Sucar—both of us graduate students—where he presented his work on relational and temporal models for high-level vision reasoning. Enrique's impressive research contributions to our field over the past 25 years have ranged from foundational work on objective probabilities, to developing advanced forms of PGMS such as temporal and event Bayesian networks, to the learning of PGMs, for example his more recent work on Bayesian chain classifiers for multidimensional classification.

Probabilistic graphical models are now widely accepted as a powerful and mature technology for reasoning under uncertainty. Unlike some of the ad hoc approaches taken in early experts systems, PGMs are based on the strong mathematical foundations of graph and probability theory. They can be used for a wide range of reasoning tasks including prediction, monitoring, diagnosis, risk assessment and decision making. There are many efficient algorithms for both inference and learning available in open-source and commercial software. Moreover, their power and efficacy has been proven through their successful application to an enormous range of realworld problem domains. Enrique Sucar has been a leading contributor in this establishment of PGMs as practical and useful technology, with his work across a wide range of application areas. These include medicine, rehabilitation and care, robotics and vision, education, reliability analysis and industrial applications ranging from oil production to power plants.

The first authors to drawn upon the early research on Bayesian networks and craft it into compelling narratives in the book form were Judea Pearl in *Probabilistic Reasoning in Intelligent Systems* and Rich Neapolitan in *Probabilistic Reasoning in Expert Systems*. This monograph from Enrique Sucar is a timely addition to the body of literature following Pearl and Neapolitan, with up-to-date coverage of a broader range of PGMs than other recent texts in this area: various

classifiers, hidden Markov models, Markov random fields, Bayesian networks and its dynamic, temporal and causal variants, relational PGMs, decision graphs and Markov decision process. It presents these PGMs, and the associated methods for reasoning (or inference) and learning, in a clear and accessible manner, making it suitable for advanced students as well as researchers or practitioners from other disciplines interested in using probabilistic models. The text is greatly enriched by the way Enrique has drawn upon his extensive practical experience in modelling with PGMs, illustrating their use across a diverse range of real-world applications from bioinformatics to air pollution to object recognition. I heartily congratulate Enrique on this book and commend it to potential readers.

Melbourne, Australia Ann E. Nicholson
May 2015

Preface

Overview

Probabilistic graphical models have become a powerful set of techniques used in several domains. This book provides a general introduction to probabilistic graphical models (PGMs) from an engineering perspective. It covers the fundamentals of the main classes of PGMs: Bayesian classifiers, hidden Markov models, Bayesian networks, dynamic and temporal Bayesian networks, Markov random fields, influence diagrams, and Markov decision processes; including representation, inference, and learning principles for all the techniques. Realistic applications for each type of model are also covered in the book.

Some key features are:

- The main classes of PGMs are presented in a single monograph under a unified framework.
- The book covers the fundamental aspects: representation, inference, and learning for all the techniques.
- It illustrates the application of the different techniques in practical problems, an important feature for students and practitioners.
- It includes some of the latest developments in the field, such as multidimensional Bayesian classifiers, relational graphical models, and causal models.
- Each chapter has a set of exercises, including suggestions for research and programming projects.

Motivating the application of probabilistic graphical models to real-world problems is one of the goals of this book. This requires not only knowledge of the different models and techniques, but also some practical experience and domain knowledge. To help the professionals in different fields gain some insight into the use of PGMs for solving practical problems, the book includes many examples of the application of the different types of models in a wide range of domains, including:

- Computer vision.
- Biomedical applications.

- Industrial applications.
- Information retrieval.
- Intelligent tutoring systems.
- Bioinformatics.
- Environmental applications.
- Robotics.
- Human–computer interaction.
- Information validation.
- Caregiving.

Audience

This book can be used as a text book for an advanced undergraduate or a graduate course in probabilistic graphical models for students of computer science, engineering, physics, etc. It could also serve as a reference book for professionals that want to apply probabilistic graphical models in different areas, or anyone who is interested in knowing the basis of these techniques.

It does not have specific prerequisites, although some background in probability and statistics is recommended. It is assumed that the reader has a basic knowledge of mathematics at the high school level, as well as a certain background in computing and programming. The programming exercises require some knowledge and experience with any programming language, such as C, C++, JAVA, Matlab, etc.

Exercises

Each chapter (except the introduction) includes a set of exercises. Some of these exercises are questions and problems designed to reinforce the understanding of the concepts and techniques presented in the chapter. There are also a few suggestions for research or programming projects (marked with "***") in each chapter, which could be used as projects for a course.

Organization

The book is divided into four parts. The first part provides a general introduction and motivation for PGMs, and reviews the required background in probability and graph theory. The second part describes the models which do not consider decisions or utilities: Bayesian classifiers, hidden Markov models, Markov random fields, Bayesian networks, and dynamic and temporal Bayesian networks. The third part

starts with a brief introduction to decision theory, and then describes the models which support decision making, including decision trees, influence diagrams, and Markov decision processes. Finally, the fourth part presents two extensions to the *standard* PGMs, one is relational probabilistic graphical models and the other causal models.

The *dependency relations* between the chapters are shown in Fig. 1. An arc from chapter X to chapter "Y", $X \rightarrow Y$, indicates that chapter X is required (or at least recommended) for understanding chapter Y. This graphical representation of the book gives a lot of information, in an analogous way to the graphical models that we will cover later.

From Fig. 1, we can deduce different ways of reading this book. First it is recommended that you read the introduction and the fundamental Chaps. 2 and 3. Then you can study relatively independently the different models in Part II: classification (Chap. 4), hidden Markov models (Chap. 5), Markov random fields (Chap. 6), and Bayesian networks (Chaps. 7–9). Before reading about learning Bayesian networks (Chap. 8), it is necessary to read Chap. 7—representation and inference; and both chapters are required before going into dynamic and temporal Bayesian networks.

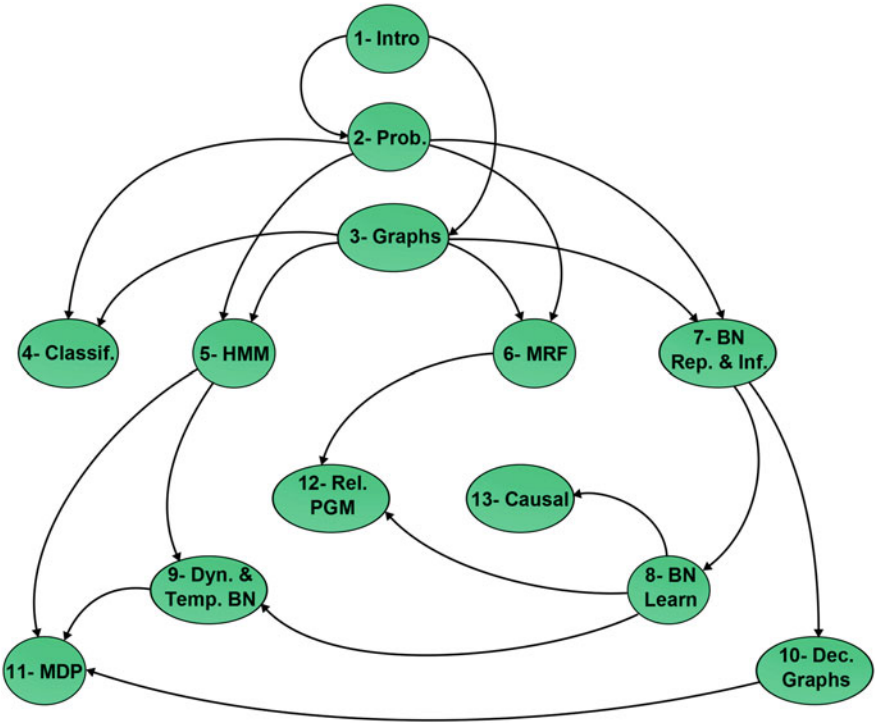


Fig. 1 This figure represents the structure of the book as a directed acyclic graph, showing which chapters are prerequisites for other chapters

The topics in Part III and IV require some of the chapters in Part II. For Chap. 10, which covers decision trees and influence diagrams, you should at least read the first chapter on Bayesian networks. For Chap. 11, which covers sequential decision making, it is recommended that you have covered hidden Markov models and dynamic and temporal Bayesian networks. Relational PGMs (Chap. 12) are based on Markov random fields and Bayesian networks; so Chaps. 6 and 8 are required. Finally, the causal models included in Chap. 13 are based on Bayesian networks including the learning techniques.

If there is not enough time in a course to cover all the book, there are several alternatives. One is to focus on probabilistic models without considering decisions or the more advanced extensions, covering Parts I and II. Another alternative is to focus on decision models, including Part I, the necessary prerequisites from Part II, and Part III. Or you can design your course a la carte, only respecting the dependencies in the graph. However, if you have the time and desire, I suggest you read all the book in order. Enjoy!

Puebla, Mexico
February 2015

Luis Enrique Sucar

Acknowledgments

This book grew out of a course that I have been teaching for several years to graduate students. It initiated as a course in Uncertain Reasoning at Tec de Monterrey in Cuernavaca, and became a course on Probabilistic Graphical Models when I moved to INAOE, Puebla in 2006. During these years, my students have been the main motivation and the source of inspiration for writing this book. I will like to thank them all for their interest, questions, and frequent corrections to my notes. This book is dedicated to all my students, past, present, and future.

I will like to acknowledge those students with whom I have collaborated, usually in a small part, for their bachelor, master, or Ph.D. thesis. Some of the novel aspects of this book and most of the application examples originated form their work. I thank them all, and will mention just a few whose work was more influential for this manuscript: Gustavo Arroyo, Héctor Hugo Avilés, Leonardo Chang, Ricardo Omar Chávez, Francisco Elizalde, Hugo Jair Escalante, Lindsey Fiedler, Giovani Gómez, Carlos Hernández, Pablo Hernández, Yasmín Hernández, Pablo Ibargüengoytia, Roger Luis-Velásquez, Miriam Martínez, José Antonio Montero, Julieta Noguez, Annette Morales, Joaquín Pérez-Brito, Miguel Palacios, Mallinali Ramírez, Alberto Reyes, Andrés Rodríguez, Elías Ruiz, Gerardo Torres-Toledano, and Julio Zaragoza. Special thanks to Lindsey Fiedler who has helped me with all the figures in the book and has revised the English.

I will also like to thank my collaborators, the common research projects and technical discussions have enriched my knowledge of many topics, and have helped me write this manuscript. In particular, I will like to mention my colleagues and friends: Juan Manuel Ahuactzin, Olivier Aycard, Concha Bielza, Roberto Ley Borrás, Cristina Conati, Javier Díez, Hugo Jair Escalante, Duncan Gillies, Jesús González, Edel García, Jesse Hoey, Pablo Ibargüengoytia, Pedro Larrañaga, Ron Leder, Jim Little, José Luis Marroquín, Oscar Mayora, Manuel Montes, Eduardo Morales, Enrique Muñoz de Cote, Julieta Noguez, Felipe Orihuela, Luis Pineda, David Poole, Alberto Reyes, Carlos Ruiz, Sunil Vadera, and Luis Villaseñor. I appreciate the comments that Edel, Felipe, and Pablo have made of some earlier versions of this book. Thanks to Ann Nicholson for providing an excellent Foreword.

I want to acknowledge the support from my institution, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), which provides an excellent environment for doing research and teaching, and has given me all the facilities to dedicate part of my time to write this book.

Last, but not least, I will like that thank my family. My parents, Fuhed† and Aida, who promoted the desire to learn and to work hard, and supported my studies. In particular, my father, who wrote several wonderful books, and gave me the inspiration (and probably the genes) for writing. My brother Ricardo, and my sisters, Shafía and Beatriz, who have always supported and motivated my dreams. And especially to my wife Doris and my children, Edgar and Diana, who have suffered the long hours I have dedicated to the book instead of them, and whose love and support is what keeps me going.

Contents

Acronyms

ADD	Algebraic Decision Diagram
AI	Artificial Intelligence
BAN	Bayesian network Augmented Naive Bayes classifier
BCC	Bayesian Chain Classifier
BCCD	Bayesian Constraint-based Causal Discovery
BN	Bayesian Network
CBN	Causal Bayesian Network
CMI	Conditional Mutual Information
CPT	Conditional Probability Table
CRF	Conditional Random Field
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
DBNC	Dynamic Bayesian Network Classifier
DD	Decision Diagram
DDN	Dynamic Decision Network
DT	Decision Tree
EC	Expected Cost
EM	Expectation–Maximization
FN	False Negative
FP	False Positive
GRM	Gibbs Random Field
HMM	Hidden Markov Model
ICM	Iterative Conditional Modes
ID	Influence Diagram
ILP	Inductive Logic Programming
KB	Knowledge Base
LIMID	Limited Memory Influence Diagram
MAG	Maximal Ancestral Graph
MAP	Maximum a Posteriori
MB	Markov Blanket
MBC	Multidimensional Bayesian network Classifier

MC	Markov Chain
MDL	Minimum Description Length
MDP	Markov Decision Process
MLN	Markov Logic Network
MN	Markov Network
MPE	Most Probable Explanation
MRF	Markov Random Field
NBC	Naïve Bayes Classifier
PAG	Parental Ancestral Graph
PGM	Probabilistic Graphical Model
PL	Pseudolikelihood
POMDP	Partially Observable Markov Decision Process
PRM	Probabilistic Relational Model
RPGM	Relational Probabilistic Graphical Model
SNBC	Semi-Naïve Bayesian Classifier
TAN	Tree Augmented Naive Bayes classifier
TEN	Temporal Event Network
TN	Temporal Node
TNBN	Temporal Nodes Bayesian Network

Notations

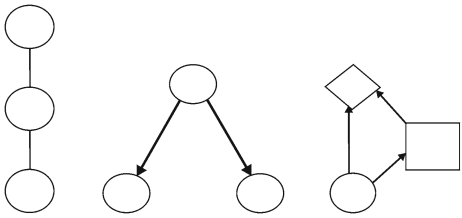
T	True
F	False
A, B, C, …	Propositions (binary variables)
$\neg A$	Not A (negation)
$A \wedge B$	A and B (conjunction)
$A \vee B$	A or B (disjunction)
$A \rightarrow B$	B if A (implication)
$A \leftrightarrow B$	A if B and B if A (double implication)
$X \in A$	X is an element of A
$\forall(X)$	Universal quantifier: for all X
$\exists(X)$	Existential quantifier: exists an X
$C \cup D$	Union of two sets
$C \cap D$	Intersection of two sets
Ω	Sample space
X	A random variable
x	A particular value of a random variable, $X = x$
$\mathbf{X}$	A vector of random variables, $\mathbf{X} = \mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_\mathrm{N}$
$\mathbf{x}$	A particular realization of vector $\mathbf{X}, \mathbf{x} = \mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_N$
$X_{1:T}$	Vector of variable X from t = 1 to t = T , $X_{1:T} = X_1, X_2, \ldots, X_T$
$P(X = x)$	Probability of X being in state x; for short $P(x)$
$P(\mathbf{X} = \mathbf{x})$	Probability of $\mathbf{X}$ being in state $\mathbf{x}$; for short $P(\mathbf{x})$
$P(x, y)$	Probability of x and y
$P(x \vee y)$	Probability of x or y
$P(x \mid y)$	Conditional probability of x given y
$P(x) \sim y$	The probability of x is *proportional* to y, that is $P(x) = k \times y$
$\mathbf{P}(\mathbf{X})$	Cumulative distribution function of a discrete variable X
$P(X)$	Probability function of a discrete variable X
$F(X)$	Cumulative distribution function of a continuous variable X
$f(X)$	Probability density function of a continuous variable X
$I(X, Y, Z)$	X independent of Z given Y
$G(V, E)$	Graph G with set of vertices V and set of edges E

$Pa(X)$	Parents of node X in a directed graph
$Nei(X)$	Neighbors of node X in a graph
$n!$	Factorial of $n, n! = n \times (n-1) \times (n-2) \times \ldots 1$
$\binom{n}{r}$	Combinations of r from n, $\binom{n}{r} = \frac{n!}{r!(n-r)!}$
$exp(x)$	Exponential of $x, exp(x) = e^x$
$\|X\|$	The dimension or number of states of a discrete variable X
μ	Mean
σ^2	Variance
σ	Standard deviation
$N(\mu, \sigma^2)$	Normal distribution with mean μ and standard deviation σ
$I(m)$	Information
$H(M)$	Entropy
$E(X)$	Expected value of a random variable X
$ArgMax_x F(X)$	The value of X for which the function F is maximum

Part I
Fundamentals

The first chapters of the book include a general introduction to probabilistic graphical models, and provide the theoretical foundations required for the rest of the book: probability theory and graph theory.

Chapter 1
Introduction



1.1 Uncertainty

For achieving their goals, intelligent agents, natural or artificial, have to select a course of actions among many possibilities. That is, they have to make decisions based on the information they can obtain from their environment, their previous knowledge, and their objectives. In many cases, the information and knowledge is incomplete or unreliable, and the results of their decisions are not certain, that is, they have to make *decisions under uncertainty*. For instance: a medical doctor in an emergency must act promptly even if she has limited information on the patient's state; an autonomous vehicle that detects what might be an obstacle in its way must decide if it should turn or stop without being certain about the obstacle's distance, size, and velocity; or a financial agent needs to select the best investment according to its vague predictions on the expected return of the different alternatives and its clients' requirements.

One of the goals of artificial intelligence is to develop systems that can reason and make decisions under uncertainty. Reasoning under uncertainty presented a challenge to early intelligent systems, as traditional paradigms were not well suited for managing uncertainty.

1.1.1 Effects of Uncertainty

Early artificial intelligence systems were based on classical logic, in which knowledge can be represented as a set of logic clauses or rules. These systems have two important properties, *modularity* and *monotonicity*, which help to simplify knowledge acquisition and inference.

A system is modular if each *piece* of knowledge can be used independently to arrive at conclusions. That is, if the premises of any logical clause or rule are true, then we can assert its conclusion without needing to consider other

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_1

elements in the knowledge base. For example, if we have the rule, $\forall X, stroke(X) \rightarrow impairedarm(X)$, then if we know that *Mary* had a stroke we know that she has an impaired-arm.

A system is monotonic if its knowledge always increases monotonically: that is, any deduced fact or conclusion is maintained even if new facts are known by the system. For example, if there is a rule such as $\forall X, bird(X) \rightarrow flies(X)$, then if *Tweety* is a bird, we can assert that she flies.

However, if we have uncertainty these two properties are not true in general. In medical systems there is usually uncertainty about the diagnosis of a patient, so if a person suffers a stroke, her arm might not be impaired; it depends on the part of the brain affected by the stroke. Similarly, not all birds fly, so if we later learn that *Tweety* is a penguin, we will need to *retract* the conclusion that she flies.

The loss of these two properties makes a system that has to reason under uncertainty more complex. In principle, the system has to take into account all available knowledge and facts when deriving a conclusion, and must be able to change its conclusions when acquiring new data.

1.2 A Brief History

From an artificial intelligence perspective, we can consider the following stages in the development of uncertainty management techniques:

Beginnings (1950s and 1960s)—artificial intelligence (AI) researchers focused on solving problems such as theorem proving, games like chess, and the "blocks world" planning domain, which do not involve uncertainty, making it unnecessary to develop techniques for managing uncertainty. The symbolic paradigm dominated AI in the beginnings.

Ad hoc techniques (1970s)—the development of expert systems for realistic applications such as medicine and mining, required the development of uncertainty management approaches. Novel ad hoc techniques were developed for specific expert systems, such as MYCIN's certainty factors [15] and Prospector's pseudo-probabilities [3]. Later it was shown that these techniques had a set of implicit assumptions which limited their applicability [5]. Also in this period, alternative theories were proposed to manage uncertainty in expert systems, including fuzzy logic [17] and the Dempster-Shafer theory [14].

Resurgence of probability (1980s)—probability theory was used in some initial expert systems, however, it was later discarded as its application in naive ways implies a high computational complexity (see Sect. 1.3). New developments, in particular Bayesian networks [11], make it possible to build complex probabilistic systems in an efficient manner, starting a new era for uncertainty management in AI.

Diverse formalisms (1990s)—Bayesian networks continued and were consolidated with the development of efficient inference and learning algorithms. Meanwhile, other techniques such as fuzzy and non-monotonic logics were considered as alternatives for reasoning under uncertainty.

Probabilistic graphical models (2000s)—several techniques based on probability and graphical representations were consolidated as powerful methods for representing, reasoning, and making decisions under uncertainty, including Bayesian networks, Markov networks, influence diagrams, and Markov decision processes, among others.

1.3 Basic Probabilistic Models

Probability theory provides a well-established foundation for managing uncertainty, therefore it is natural to use it for reasoning under uncertainty. However, if we apply probability in a naive way to complex problems, we are soon deterred by computational complexity.

In this section we will show how we can model a problem using a naive probabilistic approach based on a *flat* representation; and then how we can use this representation to answer some probabilistic queries. This will help to understand the limitations of the basic approach, motivating the development of probabilistic graphical models.[1]

Many problems can be formulated as a set of variables, $X_1, X_2, \ldots, X_n$ such that we know the values for some of these variables while the others are unknown. For instance, in medical diagnosis, the variables might represent certain symptoms and the associated diseases; usually we know the symptoms and we want to find the most probable disease(s). Another example could be a financial institution developing a system to help decide the amount of credit given to a certain customer. In this case the relevant variables are the attributes of the customer, i.e., age, income, previous credits, etc.; and a variable that represents the amount of credit to be given. Based on the customer attributes we want to determine, for instance, the maximum amount of credit that is safe to give to the customer. In general there are several types of problems that can be modeled in this way, such as diagnosis, classification, and perception problems, among others.

Under a probabilistic framework we can consider that each attribute of a problem is a random variable, such that it can take a certain value from a set of values.[2] Let us consider that the set of possible values is finite; for example, $X = \{x_1, x_2, \ldots, x_m\}$ might represent the m possible diseases in a medical domain. Each value of a random variable will have a certain probability associated in a context; in the case of X it could be the probability of each disease within certain population (what is called the *prevalence* of the disease), i.e.; $P(X = x_1)$, $P(x = x_2)$, ..., for short $P(x_1)$, $P(x_2)$, ...

[1] This and the following sections assume that the reader is familiar with some basic concepts of probability theory; a review of these and other concepts is given in Chap. 2.

[2] Random variables are formally defined later on.

If we consider two random variables, X, Y, then we can calculate the probability of X taking a certain value and Y taking a certain value, that is, $P(X = x_i \wedge Y = y_j)$ or just $P(x_i, y_j)$; this is called the joint probability of X and Y. The idea can be generalized to n random variables, where the joint probability is denoted as $P(X_1, X_2, \ldots, X_n)$. We can think of $P(X_1, X_2, \ldots, X_n)$ as a function that assigns a probability value to all possible combinations of values of the variables $X_1, X_2, \ldots, X_n$.

Therefore, we can represent a domain as:

1. A set of random variables, $X_1, X_2, \ldots, X_n$.
2. A joint probability distribution associated to these variables, $P(X_1, X_2, \ldots, X_n)$.

Given this representation, we can answer some queries with respect to certain variables in the domain, such as:

Marginal probabilities: the probability of one of the variables taking a certain value. This can be obtained by summing over all the other variables of the joint probability distribution. In other words, $P(X_i) = \sum_{\forall X \neq X_i} P(X_1, X_2, \ldots, X_n)$. This is known as *marginalization*. Marginalization can be generalized to obtain the marginal joint probability of a subset of variables by summing over the rest.

Conditional probabilities: by definition the conditional probability of X_i given that we know X_j is $P(X_i \mid X_j) = P(X_i, X_j)/P(X_j)$, $P(X_j) \neq 0$. $P(X_i, X_j)$ and $P(X_j)$ can be obtained via marginalization, and from them we can obtain conditional probabilities.

Total Abduction or MPE: given that a subset (E) of variables is known, abduction consists in finding the values of the rest of variables (J) that maximize their conditional probability given the evidence, $maxP(J \mid E)$. That is, $ArgMax_J[P(X_1, X_2, \ldots, X_n)/P(E)]$.

Partial abduction or MAP: in this case there are three subsets of variables: the evidence, E, the query variables that we want to maximize, J, and the rest of the variables, K, such that we want to maximize $P(J \mid E)$. This is obtained by marginalizing over K and maximizing over J, that is, $ArgMax_J\big[\sum_{X \in K} P(X_1, X_2, \ldots, X_n)/P(E)\big]$.

Additionally, if we have data from the domain of interest, we might obtain a *model* from this data, that is, estimate the joint probability distribution of the relevant variables.

Next we illustrate the basic approach with a simple example.

1.3.1 An Example

We will use the traditional *golf* example to illustrate the basic approach. In this problem we have five variables: *outlook, temperature, humidity, windy, play*. Table 1.1 shows some data for the golf example; all variables are discrete so they can take a

Table 1.1 A sample data set for the golf example

Outlook	Temperature	Humidity	Windy	Play
Sunny	High	High	False	No
Sunny	High	High	True	No
Overcast	High	High	False	Yes
Rain	Medium	High	False	Yes
Rain	Low	Normal	False	Yes
Rain	Low	Normal	True	No
Overcast	Low	Normal	True	Yes
Sunny	Medium	High	False	No
Sunny	Low	Normal	False	Yes
Rain	Medium	Normal	False	Yes
Sunny	Medium	Normal	True	Yes
Overcast	Medium	High	True	Yes
Overcast	High	Normal	False	Yes
Rain	Medium	High	True	No

value from a finite set of values, for instance Outlook could be sunny, overcast, or rain. We will now illustrate how we can calculate the different probabilistic queries mentioned before for this example.

First, we will simplify the example using only two variables, Outlook and Temperature. From the data in Table 1.1 we can obtain the joint probability of Outlook and Temperature as depicted in Table 1.2. Each entry in the table corresponds to the joint probability $P(Outlook, Temperature)$, for example, $P(Outlook = S, Temp. = H) = 0.143$.

Let us first obtain the marginal probabilities for the two variables. If we sum per row (marginalizing Temperature), then we obtain the marginal probabilities for Outlook, $P(Outlook) = [0.357, 0.286, 0.357]$; and if we sum per column we obtain the marginal probabilities for Temperature, $P(Temperature) = [0.286, 0.428, 0.286]$. From these distributions, we obtain that the most probable Temperature is M and the most probable values for Outlook are S and R.

Table 1.2 Joint probability distribution for Outlook and Temperature

Outlook	Temp.		
	H	M	L
S	0.143	0.143	0.071
O	0.143	0.071	0.071
R	0	0.214	0.143

Now we can calculate the conditional probabilities of Outlook given Temperature and vice versa. For instance:

$$\begin{aligned} P(Temp. \mid Outlook = R) &= P(Temp. \wedge Outlook = R)/P(Outlook = R) \\ &= [0, 0.6, 0.4] \\ P(Outlook \mid Temp. = L) &= P(Outlook \wedge Temp. = L)/P(Temp. = L) \\ &= [0.25, 0.25, 0.5] \end{aligned}$$

Given these distributions, the most probable Temperature given that the Outlook is Rain is Medium, and the most probable Outlook given that the Temperature is Low is Rain.

Finally, the most probable combination of Outlook and Temperature is {*Rain*, *Medium*}, which in this case can be obtained directly from the joint probability table.

Although it is possible to compute the different probabilistic queries for this small example, this approach becomes impractical for complex problems with many variables, as the size of the table and the direct computation of marginal and conditional probabilities grow exponentially with the number of variables in the model.

Another disadvantage of this naive approach is that to have good estimates for the joint probabilities from data, we will require a very large database if there are many variables in the model. A rule of thumb is that the number of instances (records) is at least 10 times the number of possible combination values for the variables, so if we consider 50 binary variables, it will require at least 10×2^{50} instances!

Finally, the joint probability table does not say much about the problem to a human; so this approach also has cognitive limitations.

The problems seen with the basic approach are some of the motivations for the development of probabilistic graphical models.

1.4 Probabilistic Graphical Models

Probabilistic graphical models (PGMs) provide a framework for managing uncertainty based on probability theory in a computationally efficient way. The basic idea is to consider only those independence relations that are valid for a certain problem, and include these in the probabilistic model to reduce complexity in terms of memory requirements and also computational time. A natural way to represent the dependence and independence relations between a set of variables is using graphs, such that variables that are directly dependent are connected, and the independence relations are implicit in this dependency graph.

A probabilistic graphical model is a compact representation of a joint probability distribution, from which we can obtain marginal and conditional probabilities. It has several advantages over a flat representation:

- It is generally much more compact (space).
- It is generally much more efficient (time).
- It is easier to understand and communicate.
- It is easier to learn form data or to construct based on expert knowledge.

A probabilistic graphical model is specified by two aspects: (i) a graph, $G(V, E)$, that defines the structure of the model; and (ii) a set of local functions, $f(Y_i)$, that define the parameters, where Y_i is a subset of X. The joint probability is obtained by the product of the local functions:

$$P(X_1, X_2, \ldots, X_N) = K \prod_{i=1}^{M} f(Y_i) \tag{1.1}$$

where K is a normalization constant (it makes the probabilities sum to one).

This representation in terms of a graph and a set of local functions (called *potentials*) is the basis for inference and learning in PGMs:

Inference: obtain the marginal or conditional probabilities of any subset of variables Z given any other subset Y.

Learning: given a set of data values for X (that can be incomplete) estimate the structure (graph) and parameters (local functions) of the model.

We can classify probabilistic graphical models according to three dimensions:

1. Directed or undirected
2. Static or dynamic
3. Probabilistic or decisional

The first dimension has to do with the type of graph used to represent the dependence relations. Undirected graphs represent symmetric relations, while directed graphs represent relations in which the direction is important. Given a set of random variables with the corresponding conditional independence relations, it is not possible to represent all the relations with one type of graph [11]; thus, both types of models are useful.

The second dimension defines if the model represents a set of variables at a certain point in time (static) or across different times (dynamic). Probabilistic models only include random variables, while decisional models also include decision and utility variables.

The most common classes of PGMs and their type according to the previous dimensions are summarized in Table 1.3.

All these types of models will be covered in detail in the following chapters, and also some extensions that consider more expressive models (relational probabilistic graphical models) or represent causal relations (causal Bayesian networks).

Table 1.3 Main types of probabilistic graphical models

Type	**D**irected/**U**ndirected	**S**tatic/**D**ynamic	**P**rob./**D**ecisional
Bayesian classifiers	D/U	S	P
Markov chains	D	D	P
Hidden Markov models	D	D	P
Markov random fields	U	S	P
Bayesian networks	D	S	P
Dynamic Bayesian networks	D	D	P
Influence diagrams	D	S	D
Markov decision processes (MDPs)	D	D	D
Partially observable MDPs	D	D	D

1.5 Representation, Inference, and Learning

There are three main aspects for each class of probabilistic graphical model, representation, inference, and learning.

The representation is the basic property of each model, and it defines which entities constitute it and how these are related. For instance, all PGMs can be represented as graphs that define the structure of the model and by local functions that describe its parameters. However, the type of graph and the local functions vary for the different types of models.

Inference consists in answering different probabilistic queries based on the model and some evidence. For instance, obtaining the posterior probability distribution of a variable or set of variables given that other variables in the model are known. The challenge is how to do this efficiently.

To construct these models there are basically two alternatives: to build it "by hand" with the aid of domain experts or to induce the model from data. The emphasis in recent years has been to induce the models based on machine learning techniques, because it is difficult and costly to do it with the aid of experts. In particular obtaining the parameters for the models is usually done based on data, as humans tend to be *bad* estimators of probabilities.

An important property of these techniques from an application point of view is that they tend to separate the inference and learning techniques from the model. That is, as in other artificial intelligence representations such as logic and production rules, the reasoning mechanisms are general and can be applied to different models. As a result, the techniques developed for probabilistic inference in each class of PGM can be applied directly for different models in a variety of applications.

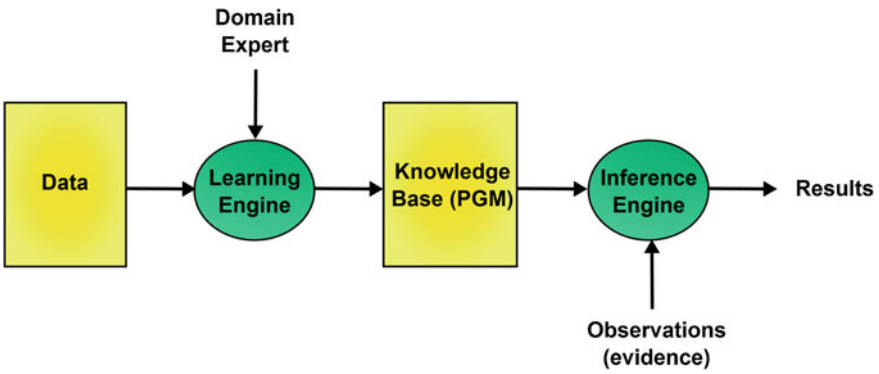


Fig. 1.1 A schematic representation of the general paradigm followed by the different classes of PGMs, in which there is a clear separation between the learning and inference engines, which are generic, and the knowledge base, which depends on the particular application

This idea is illustrated schematically in Fig. 1.1. Based on data or expert knowledge or a combination of both, the knowledge base—in this case a probabilistic graphical model, is built using the learning engine. Once we have the model, we can use it to do probabilistic reasoning through the inference engine; based on the observations and the model, the inference engine derives the results. The learning and inference engines are generic for a class of PGMs, so they can be applied for modeling and reasoning in different domains.

For each type of PGM presented in the book, we will first describe its representation and then present some of the most common inference and learning techniques.

1.6 Applications

Most real-world problems imply dealing with uncertainty and usually involve a large number of factors or variables to be considered when solving them. Probabilistic graphical models constitute an ideal framework to solve complex problems with uncertainty, so they are being applied in a wide range of domains such as:

- Medical diagnosis and decision making.
- Mobile robot localization, navigation, and planning.
- Diagnosis for complex industrial equipment such as turbines and power plants.
- User modeling for adaptive interfaces and intelligent tutors.
- Speech recognition and natural language processing.
- Pollution modeling and prediction.
- Reliability analysis of complex processes.
- Modeling the evolution of viruses.
- Object recognition in computer vision.

- Information retrieval.
- Energy markets.

Different types of PGMs are more appropriate for different applications, as will be shown in the following chapters when we present application examples for each class of PGM.

1.7 Overview of the Book

The book is divided into four parts.

Part I provides the mathematical foundations for understanding the models and techniques presented in the following chapters. Chapter 2 presents a review of some basic concepts in probability and information theory which are important for understanding probabilistic graphical models. Chapter 3 gives an overview of graph theory, with emphasis on certain aspects that are important for representation and inference in probabilistic graphical models; including, among others, cliques, triangulated graphs, and perfect orderings.

Part II covers the different types of probabilistic models that only have random variables, and do not consider decisions or utilities in the model. This is the largest part and it includes the following types of PGMs:

- Bayesian classifiers
- Markov chains and hidden Markov models
- Markov random fields
- Bayesian networks
- Dynamic Bayesian networks and temporal networks

A chapter is dedicated to each type of model (except for Bayesian networks which is divided into two chapters), including representation, inference, and learning; and practical application examples.

Part III presents those models that consider decisions and utilities, and as such are focused on aiding the decision maker to take the optimal actions under uncertainty. This part includes two chapters. The first chapter covers modeling techniques for when there are one or few decisions, including decision trees and influence diagrams. The second chapter covers sequential decision problems, in particular Markov decision processes.

Part IV considers alternative paradigms that can be thought as extensions to the traditional probabilistic graphical models. It includes two chapters. The first chapter is dedicated to relational probabilistic models, which increase the representational power of *standard* PGMs, by combining the expressive power of first-order logic with the uncertain reasoning capabilities of probabilistic models. The second chapter introduces causal graphical models that go beyond representing probabilistic dependencies, to express cause and effect relations.

1.8 Additional Reading

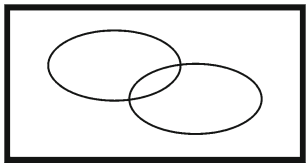
In this book we present a broad perspective on probabilistic graphical models. There are few other books that have a similar coverage. One is by Koller and Friedman [7], which presents the models under a different structure with less emphasis on applications. Another is by Lauritzen [8], which has a more statistical focus. Bayesian programming [1] provides an alternative approach to implement graphical models based on a programming paradigm.

There are several books that cover one or few types of models in greater depth, such as: Bayesian networks [2, 10, 11], decision graphs [6], Markov random fields [9], Markov decision processes [13], relational probabilistic models [4] and causal models [12, 16].

References

1. Bessiere, P., Mazer, E., Ahuactzin, J.M., Mekhnacha, K.: Bayesian Programming. CRC Press, Boca Raton (2014)
2. Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press, New York (2009)
3. Duda, R.O., Hart, P.A., Nilsson, N.L.: Subjective Bayesian methods for rule-based inference systems. In: Proceeding of the National Computer Conference, vol. 45, pp. 1075–1082 (1976)
4. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
5. Heckerman, D.: Probabilistic interpretations for MYCIN's certainty factors. In: Proceedings of the First Conference on Uncertainty in Artificial Intelligence (UAI), pp. 9–20 (1985)
6. Jensen, F.V.: Bayesian Networks and Decision Graphs. Springer, New York (2001)
7. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press, Cambridge (2009)
8. Lauritzen, S.L.: Graphical Models. Oxford University Press, Oxford (1996)
9. Li, S.Z.: Markov Random Field Modeling in Image Analysis. Springer, London (2009)
10. Neapolitan, R.E.: Probabilistic Reasoning in Expert Systems. Wiley, New York (1990)
11. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)
12. Pearl, J.: Causality: Models, Reasoning and Inference. Cambridge University Press, New York (2009)
13. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)
14. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton (1976)
15. Shortliffe, E.H., Buchanan, B.G.: A model of inexact reasoning in medicine. Math. Biosci. **23**, 351–379 (1975)
16. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. MIT Press, New York (2000)
17. Zadeh, L.A.: Knowledge Representation in Fuzzy Logic. IEEE Trans. Knowl. Data Eng. **1**(1), 89–100 (1989)

Chapter 2
Probability Theory



2.1 Introduction

Probability theory originated in games of chance and has a long and interesting history; it has developed into a mathematical language for quantifying uncertainty.

Consider a certain *experiment*, such as throwing a die; this experiment can have different results, we call each result an *outcome* or element. In the die example, the possible outcomes or elements are the following: {1, 2, 3, 4, 5, 6}. The set of all possible outcomes of an experiment is called the sample space, Ω. An *event* is a set of elements or subset of Ω. Continuing with the die example, one event could be that the die shows an even number, that is, {2, 4, 6}.

Before we mathematically define probability, it is worth discussing the meaning or interpretation of probability. Several definitions or interpretations of probability have been proposed, starting from the *classical* definition by Laplace, and including the *limiting frequency*, the *subjective*, the *logical*, and the *propensity* interpretations [1]:

Classical: probability has to do with equiprobable events; if a certain experiment has N possible outcomes, the probability of each outcome is $1/N$.

Logical: probability is a measure of rational belief; that is, according to the available evidence, a rational person will have a certain belief regarding an event, which will define its probability.

Subjective: probability is a measure of the personal degree of belief in a certain event; this could be measured in terms of a betting factor—the probability of a certain event for an individual is related to how much that person is willing to bet on that event.

Frequency: probability is a measure of the number of occurrences of an event given a certain experiment, when the number of repetitions of the experiment tends to infinity.

Propensity: probability is a measure of the number of occurrences of an event under repeatable conditions, even if the experiment only occurs once.

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_2

These interpretations can be grouped into what are the two main approaches in probability and statistics:

- Objective (classical, frequency, propensity): probabilities exist in the *real* world and can be measured.
- Epistemological (logical, subjective): probabilities have to do with human knowledge, they are measures of belief.

Both approaches follow the same mathematical axioms defined below; however, there are differences in the manner in which probability is applied, in particular in statistical inference. These differences gave way to the main two schools for statistics: the frequentist and the Bayesian schools. In the field of artificial intelligence, in particular in expert systems, the preferred approach tends to be the epistemological or subjective one; however, the objective approach is also used [4].

We will consider the logical or normative approach and define probabilities in terms of the degree of plausibility of a certain proposition given the available evidence [2]. Based on Cox's work, Jaynes establishes some basic desiderata that this degree of plausibility must follow [2]:

- Representation by real numbers.
- Qualitative correspondence with common sense.
- Consistency.

Based on these intuitive principles, we can derive the three axioms of probability:

1. $P(A)$ is a continuous monotonic function in $[0, 1]$.
2. $P(A, B \mid C) = P(A \mid C)P(B \mid A, C)$ (product rule).
3. $P(A \mid B) + P(\neg A \mid B) = 1$ (sum rule).

Where A, B, C are propositions (binary variables) and $P(A)$ is the probability of proposition A. $P(A \mid C)$ is the probability of A given that C is known, which is called *conditional probability*. $P(A, B \mid C)$ is the probability of A AND B given C (logical conjunction) and $P(\neg A \mid C)$ is the probability of NOT A (logical negation) given C. These rules are equivalent to the most commonly used Kolmogorov axioms. From these axioms, all conventional probability theory can be derived.

2.2 Basic Rules

The probability of the disjunction (logical sum) of two propositions is given by the *sum rule*: $P(A + B \mid C) = P(A \mid C) + P(B \mid C) - P(A, B \mid C)$; if propositions A and B are mutually exclusive given C, we can simplify it to: $P(A + B \mid C) = P(A \mid C) + P(B \mid C)$. This can be generalized for N mutually exclusive propositions to:

$$P(A_1 + A_2 + \cdots A_N \mid C) = P(A_1 \mid C) + P(A_2 \mid C) + \cdots + P(A_N \mid C) \quad (2.1)$$

In the case that there are N mutually exclusive and exhaustive hypotheses, $H_1, H_2, \ldots, H_N$, and if the evidence B does not favor any of them, then according to the principle of indifference: $P(H_i \mid B) = 1/N$.

According to the logical interpretation there are no *absolute* probabilities, all are conditional on some background information.[1] $P(H \mid B)$ conditioned only on the background B is called a *prior* probability; once we incorporate some additional information D we call it a *posterior* probability, $P(H \mid D, B)$. From the product rule we obtain:

$$P(D, H \mid B) = P(D \mid H, B)P(H \mid B) = P(H \mid D, B)P(D \mid B) \tag{2.2}$$

From which we obtain:

$$P(H \mid D, B) = \frac{P(H \mid B)P(D \mid H, B)}{P(D \mid B)} \tag{2.3}$$

This last equation is known as the *Bayes rule* and the term $P(D \mid H, B)$ as the *likelihood*, $L(H)$.

In some cases the probability of H is not influenced by the knowledge of D, so it is said that H and D are *independent* given some background B, therefore, $P(H, D \mid B) = P(H \mid B)$. In the case in which A and B are independent, the product rule can be simplified to: $P(A, B \mid C) = P(A \mid C)P(B \mid C)$, and this can be generalized to N mutually independent propositions:

$$P(A_1, A_2, \ldots, A_N \mid B) = P(A_1 \mid B)P(A_2 \mid B) \cdots P(A_N \mid B) \tag{2.4}$$

If two propositions are independent given only the background information they are *marginally* independent; however, if they are independent given some additional evidence, E, then they are *conditionally* independent: $P(H, D \mid B, E) = P(H \mid B, E)$. For example, consider that A represents the proposition *watering the garden*, B the *weather forecast* and C *raining*. Initially, watering the garden is not independent of the weather forecast; however, once we observe rain, they become independent. That is (omitting the background term), $P(A, B \mid C) = P(A \mid C)$.

Probabilistic graphical models are based on these conditions of marginal and conditional independence.

The probability of a conjunction of N propositions, that is, $P(A_1, A_2, \ldots, A_N \mid B)$, is usually called the *joint* probability. If we generalize the product rule to N propositions we obtain what is known as the *chain* rule:

$$P(A_1, A_2, \ldots, A_N \mid B) = P(A_1 \mid A_2, A_3, \ldots, A_N, B)P(A_2 \mid A_3, A_4, \ldots, A_N, B) \cdots P(A_N \mid B) \tag{2.5}$$

[1] It is commonly written $P(H)$ without explicit mention of the conditioning information. In this case we assume that there is still some context under which probabilities are considered even if it is not written explicitly.

Thus the joint probability of N propositions can be obtained by this rule. Conditional independence relations between the propositions can be used to simplify this product; that is, for instance if A_1 and A_2 are independent given $A_3, \ldots, A_N, B$, then the first term in Eq. 2.5 can be simplified to $P(A_1 \mid A_3, \ldots, A_N, B)$.

Another important relation is the rule of *total probability*. Consider a partition, $B = \{B_1, B_2, \ldots, B_n\}$, on the sample space Ω, such that $\Omega = B_1 \cup B_2 \cup \cdots \cup B_n$ and $B_i \cap B_j = \emptyset$. That is, B is a set of mutually exclusive events that cover the entire sample space. Consider another event A; A is equal to the union of its intersections with each event $A = (B_1 \cap A) \cup (B_2 \cap A) \cup \cdots \cup (B_n \cap A)$. Then, based on the axioms of probability and the definition of conditional probability we can derive the rule of total probability:

$$P(A) = \sum_i P(A \mid B_i) P(B_i) \tag{2.6}$$

Given the total probability rule, we can rewrite Bayes rule as (omitting the background term):

$$P(B \mid A) = \frac{P(B)P(A \mid B)}{\sum_i P(A \mid B_i)P(B_i)} \tag{2.7}$$

This last expression is commonly known as Bayes Theorem.

2.3 Random Variables

If we consider a finite set of exhaustive and mutually exclusive propositions,[2] then a discrete variable X can represent this set of propositions, such that each value x_i of X corresponds to one proposition. If we assign a numerical value to each proposition x_i, then X is a *discrete random variable*. For example, the outcome of the toss of a die is a discrete random variable with six possible values $1, 2, \ldots, 6$. The probabilities for all possible values of X, $P(X)$ is the probability distribution of X. Considering the die example, for a fair die the probability distribution will be:

x	1	2	3	4	5	6
$P(x)$	1/6	1/6	1/6	1/6	1/6	1/6

This is an example of a *uniform* probability distribution. There are several probability distributions that have been defined. Another common distribution is the *binomial* distribution. Assume we have an urn with N colored balls, red and black, of which M are red, so the fraction of red balls is $\pi = M/N$. We draw a ball at

[2] This means that one and only one of the propositions has a value of TRUE.

random, record its color, and return it to the urn, mixing the balls again (so that, in principle, each draw is independent of the previous one). The probability of getting r red balls in n draws is:

$$P(r \mid n, \pi) = \binom{n}{r} \pi^r (1-\pi)^{n-r}, \tag{2.8}$$

where $\binom{n}{r} = \frac{n!}{r!(n-r)!}$.

This is an example of a binomial distribution which is applied when there are n independent trials, each with two possible outcomes (success or failure), and the probability of success is constant over all trials. There are many other distributions; we refer the interested reader to the additional reading section at the end of the chapter.

There are two important quantities that in general help to characterize a probability distribution. The expected value or *expectation* of a discrete random variable is the average of the possible values, weighted according to their probabilities:

$$E(X \mid B) = \sum_{i=1}^{N} P(x_i \mid B) x_i \tag{2.9}$$

The *variance* is defined as the expected value of the square of the variable minus its expectation:

$$Var(X \mid B) = \sum_{i=1}^{N} P(x_i \mid B)(x_i - E(X))^2 \tag{2.10}$$

Intuitively, the variance gives a measure of how *wide* or *narrow* the probabilities are distributed for a certain random variable. The square root of the variance is known as the standard deviation, which is usually more intuitive as its units are the same as those of the variable.

So far we have considered discrete variables, however, the rules of probability can be extended to continuous variables. If we have a continuous variable X, we can divide it into a set of mutually exclusive and exhaustive intervals, such that $P = (a < X \leq b)$ is a proposition, thus the rules derived so far apply to it. A *continuous random variable* can be defined in terms of a *probability density function*, $f(X \mid B)$, such that:

$$P(a < X \leq b \mid B) = \int_a^b f(X \mid B) dx \tag{2.11}$$

The probability density function must satisfy $\int_{-\infty}^{\infty} f(X \mid B) dx = 1$.

An example of a continuous probability distribution is the *Normal* or Gaussian distribution. This distribution plays an important role in many applications of probability and statistics, as many phenomena in nature have an approximately normal distribution; it is also prevalent in probabilistic graphical models due to its mathematical properties.

A normal distribution is denoted as $N(\mu, \sigma^2)$, where μ is the *mean* (center) and σ is the *standard deviation* (spread); and it is defined as:

$$f(X \mid B) = \frac{1}{\sigma\sqrt{2\pi}} exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\} \tag{2.12}$$

The density function of a Gaussian distribution is depicted in Fig. 2.1.

Another important continuous distribution is the exponential distribution; for example, the time it takes for a certain piece of equipment to fail is usually modeled by an exponential distribution. The exponential distribution is denoted as $Exp(\beta)$; it has a single parameter $\beta > 0$, and it is defined as:

$$f(X \mid B) = \frac{1}{\beta} e^{-x/\beta}, x > 0 \tag{2.13}$$

An example of an exponential density function is shown in Fig. 2.2.

It is common to represent probability distributions, in particular for continuous variables, using the cumulative distribution function, F. The cumulative distribution function of a random variable, X, is the probability that $X \leq x$. For a continuous variable, it is defined in terms of the density function as:

$$F(x) = \int_{-\infty}^{x} f(X) \tag{2.14}$$

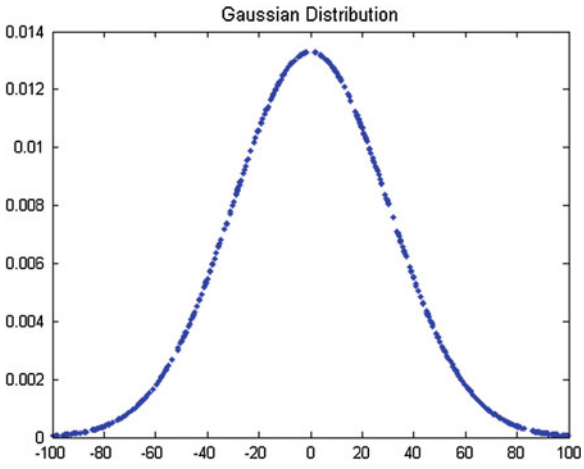


Fig. 2.1 Probability density function of the Gaussian distribution

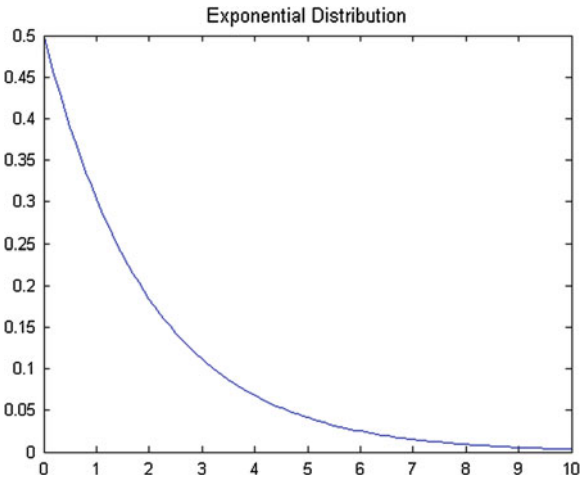


Fig. 2.2 Probability density function of the exponential distribution

The following are some properties of cumulative distribution functions:

- In the interval [0, 1]: $0 \leq F(X) \leq 1$
- Nondecreasing: $F(X_1) < F(X_2)$ if $X_1 < X_2$
- Limits: $lim_{x \to -\infty} = 0$ and $lim_{x \to \infty} = 1$

In the case of discrete variables, the cumulative probability, $P(X \leq x)$ is defined as:

$$\mathbf{P}(x) = \sum_{x=-\infty}^{X=x} P(X) \tag{2.15}$$

It has similar properties as the cumulative distribution function.

2.3.1 Two-Dimensional Random Variables

The concept of a random variable can be extended to two or more dimensions. Given two random variables, X and Y, their joint probability distribution is defined as $P(x, y) = P(X = x \wedge Y = y)$. For example, X might represent the number of products completed in one day in product line one, and Y the number of products completed in one day in product line two, thus $P(x, y)$ corresponds to the probability of producing x products in line one and y products in line two. $P(X, Y)$ must follow the axioms of probability, in particular: $0 \leq P(x, y) \leq 1$ and $\sum_x \sum_y P(X, Y) = 1$.

The distribution for two-dimensional discrete random variables (known as the *bivariate* distribution) can be represented in tabular form. For instance, consider the example of the two product lines, and assume that line one (X) may produce 1, 2, or 3 products per day, and line two (Y), 1 or 2 products. Then a possible joint distribution, $P(X, Y)$ is shown in Table 2.1.

Table 2.1 An example of a two-dimensional discrete probability distribution

	X = 1	X = 2	X = 3
Y = 1	0.1	0.3	0.3
Y = 2	0.2	0.1	0

Given the joint probability distribution, $P(X, Y)$, we can obtain the distribution for each individual random variable, what is known as the marginal probability:

$$P(x) = \sum_{y} P(X, Y);\ P(y) = \sum_{x} P(X, Y) \tag{2.16}$$

For instance, if we consider the joint distribution of Table 2.1, we can obtain the marginal probabilities for X and Y. For example, $P(X = 2) = 0.3 + 0.1 = 0.4$ and $P(Y = 1) = 0.1 + 0.3 + 0.3 = 0.7$.

We can also calculate the conditional probabilities of X given Y and vice versa:

$$P(x \mid y) = P(x, y)/P(y);\ P(y \mid x) = P(x, y)/P(x) \tag{2.17}$$

Following the example in Table 2.1:

$$P(X = 3 \mid Y = 1) = P(X = 3, Y = 1)/P(Y = 1) = 0.3/0.7 = 0.4286$$

The concept of independence can be applied to two-dimensional random variables. Two random variables, X, Y are independent if their joint probability distribution is equal to the product of their marginal distributions (for all values of X and Y):

$$P(X, Y) = P(X)P(X) \rightarrow \textit{Independent}\ (X, Y) \tag{2.18}$$

Another useful measure is called *correlation*—it is a measure of the degree of linear relation between two random variables, X, Y and is defined as:

$$\rho(X, Y) = E\{[X - E(X)][Y - E(Y)]\}/(\sigma_x \sigma_y) \tag{2.19}$$

where $E(X)$ is the expected value of X and σ_x its standard deviation. The correlation is in the interval $[-1, 1]$; a positive correlation indicates that as X increases, Y tends to increase; and a negative correlation that as X increases, Y tends to decrease.

Note that a correlation of zero does not necessarily imply independence, as the correlation only measures a linear relationship. So it could be that X and Y have a zero correlation but are related through a higher order function, and thus are not independent.

2.4 Information Theory

Information theory originated in the area of communications, although it is relevant for many different fields. In the case of probabilistic graphical models, it is mainly applied in learning. In this section we will cover the basic concepts of information theory.

Assume that we are communicating the occurrence of a certain event. Intuitively, we can think that the amount of *information* from communicating an event is inverse to the probability of the event. For example, consider that a message is sent informing about one of the following events:

1. It is raining in New York.
2. There was an earthquake in New York.
3. A meteorite fell over New York City.

The probability of the first event is higher than the second, and that of the second is higher than the third. Thus, the message for event 1 has the lowest amount of information and the message for event 3 gives the highest amount of information.

Let us now see how we can formalize the concept of information. Assume we have a source of information that can send q possible messages, $m_1, m_2, \ldots, m_q$; where each message corresponds to an event with probabilities $P_1, P_2, \ldots, P_q$. We want to find a function $I(m)$ based on the probability of m. The function must satisfy the following properties:

- The information ranges from zero to infinity: $I(m) \geq 0$.
- The information increases as the probability decreases: $I(m_i) > I(m_j)$ if $P(m_i) < P(m_j)$.
- The information tends to infinity as the probability tends to zero: $I(m) \to \infty$ if $P(m) \to 0$.
- The information of two messages is equal to the sum of that of the individual messages if these are independent: $I(m_i + m_j) = I(m_i) + I(m_j)$ if m_i independent of m_j.

A function that satisfies the previous properties is the logarithm of the inverse of the probability, that is,

$$I(m_k) = log(1/P(m_k)) \tag{2.20}$$

It is common to use base two logarithms, so the information is measured in "bits":

$$I(m_k) = log_2(1/P(m_k)) \tag{2.21}$$

For example, if we assume that the probability of the message m_r "raining in New York" is $P(m_r) = 0.25$, the corresponding information is $I(m_r) = log_2(1/0.25)$=2.

Once we have defined information for a particular message, another important concept is the average information for the q messages; that is, the expected value of the information also known as *entropy*. Given the definition of expected value, the average information of q message or entropy is defined as:

$$H(m) = E(I(m)) = \sum_{i=1}^{i=q} P(m_i) log_2(1/P(m_i)) \tag{2.22}$$

This can be interpreted as that on average H bits of information will be sent.

An interesting question is: When will H have its maximum and minimum values? Consider a binary source such that there are only two messages, m_1 and m_2; with $P(m_1) = p_1$ and $P(m_2) = p_2$. Given that there are only two possible messages, $p_2 = 1 - p_1$, so H only depends on one parameter, p_1 (or just p). Figure 2.3 shows a graph of H with respect to p. Observe that H is at its maximum when $p = 0.5$ and at its minimum (zero) when $p = 0$ and $p = 1$. In general, the entropy is at its maximum when there is a uniform probability distribution for the events; it is at its minimum when there is one element that has a probability of one and the rest have a probability of zero.

If we consider the conditional probabilities, we can extend the concept of entropy to *conditional entropy*:

$$H(X \mid y) = \sum_{i=1}^{i=q} P(X_i \mid y) log_2[1/P(X_i \mid y)] \tag{2.23}$$

Another extension of entropy is the *cross entropy*:

$$H(X, Y) = \sum_{X} \sum_{Y} P(X, Y) log_2[P(X, Y)/P(X)P(Y)] \tag{2.24}$$

The cross entropy provides a measure of the mutual information (dependency) between two random variables; it is zero when the two variables are independent.

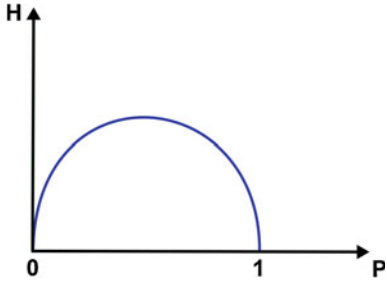


Fig. 2.3 Entropy versus probability for a binary source. The entropy is at its maximum when the probability is 0.5, and at its minimum when the probability is zero and one

2.5 Additional Reading

Donald Gillies [1] provides a comprehensive account of the different philosophical approaches to probability. An excellent book on probability theory from a logical perspective is [2]. Wasserman [5] gives a concise course on probability and statistics oriented for computer science and engineering students. There are several books on information theory; one that relates it to machine learning and inference is [3].

2.6 Exercises

1. If we throw two dice, what is the probability that the outcomes add to exactly seven? Seven or more?
2. If we assume that the height of a group of students follows a normal distribution with a mean of 1.7 m and a standard deviation of 0.1 m, how probable is it that there is a student of height above 1.9 m?
3. Demonstrate by mathematical induction the chain rule.
4. Assume that a person has one of two possible diseases, hepatitis (H) or typhoid (T). There are two symptoms associated to these diseases: headache (D) and fever (F), which could be TRUE or FALSE. Given the following probabilities: $P(T) = 0.5$, $P(D \mid T) = 0.7$, $P(D \mid \neg T) = 0.4$, $P(F \mid T) = 0.9$, $P(F \mid \neg T) = 0.5$. Describe the sampling space and complete the partial probability tables.
5. Given the data for the previous problem, and assuming that the symptoms are independent given the disease, obtain the probability that the person has hepatitis given that she does not have a headache and does have a fever.
6. Given the two-dimensional probability distribution in the table below, obtain: (a) $P(X_1)$, (b) $P(Y_2)$, and (c) $P(X_1 \mid Y_1)$.

	Y_1	Y_2	Y_3
X_1	0.1	0.2	0.1
X_2	0.3	0.1	0.2

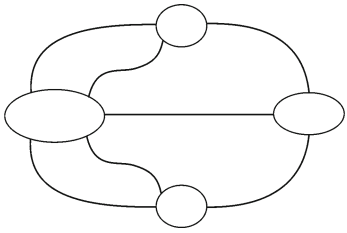
7. In the previous problem, are X and Y independent?
8. In a certain place, the statistics show that in a year the weather behaves in the following way. From 365 days, 200 are sunny, 60 cloudy, 40 rainy, 20 snowy, 20 with thundershowers, 10 with hail, 10 windy, and 5 with drizzle. If on each day a message is sent about the weather, what is the information of the message for each type of weather?

9. Considering the information for each type of weather in the previous problem, what is the average information (entropy) of the message?
10. ***Investigate the different philosophical interpretations of probability, and discuss the advantages and limitations of each one of them. Which one do you consider the most appropriate? Why?

References

1. Gillies, D.: Philosophical Theories of Probability. Routledge, London (2000)
2. Jaynes, E.T.: Probability Theory: The Logic of Science. Cambridge University Press, Cambridge (2003)
3. MacKay, D.J.: Information Theory, Inference and Learning Algorithms. Cambridge University Press, Cambridge (2004)
4. Sucar, L.E., Gillies, D.F., Gillies, D.A.: Objective Probabilities in Expert Systems. Artif. Intell. **61**, 187–208 (1993)
5. Wasserman, L.: All of Statistics: A Concise Course in Statistical Inference. Springer, New York (2004)

Chapter 3
Graph Theory



3.1 Definitions

A *graph* provides a compact way to represent binary relations between a set of objects. For example, consider a set of cities in a certain region, and the roads that connect these cities. Then a map of the region is essentially a graph, in which the object are the cities and the direct roads between pairs of cities are the relations. Graphs are usually represented graphically. Objects are represented as circles or ovals, and relations as lines or arrows; see Fig. 3.1. There are two basic types of graphs: *undirected graphs* and *directed graphs*. Next, we formalize the definitions of directed and undirected graphs.

Given V, a nonempty set, a binary relation $E \subseteq V \times V$ on V is a set of ordered pairs, (V_j, V_k), such that $V_j \in V$ and V_k in V. A *directed graph* or *digraph* is an ordered pair, $G = (V, E)$, where V is a set of vertices or nodes and E is a set of arcs that represent a binary relation on V; see Fig. 3.1b. Directed graphs represent antisymmetric relations between objects, for instance the "parent" relation.

An *undirected graph* is an ordered pair, $G = (V, E)$, where V is a set of vertices or nodes and E is a set of edges that represent symmetric binary relations: $(V_j, V_k) \in E \rightarrow (V_k, V_j) \in E$; see Fig. 3.1a. Undirected graphs represent symmetric relations between objects, for example, the "brother" relation.

If there is an edge $E_i(V_j, V_k)$ between nodes j and k, then V_j is adjacent to V_k. The *degree* of a node is the number of edges that are incident in that node. In Fig. 3.1a, the upper node has a degree of 2 and the two lower nodes have a degree of 1.

Two edges associated to the same pair of vertices are said to be *parallel edges*; an edge which is incident on a single vertex is a *cycle*; and a vertex that is not an endpoint to any edge is an *isolated vertex*—it has degree 0. These are illustrated in Fig. 3.2.

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_3

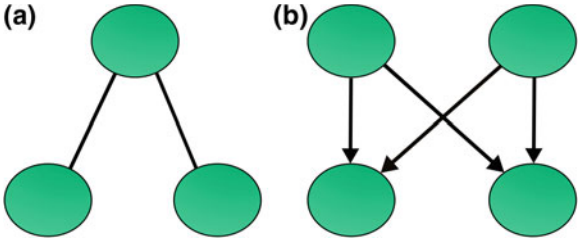


Fig. 3.1 Graphs: **a** undirected graph, **b** directed graph

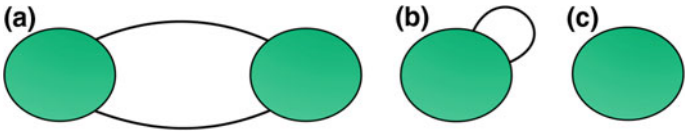


Fig. 3.2 Examples of: **a** parallel edges, **b** cycle, **c** isolated vertex

In a directed graph, the number of arcs pointing to a node is its *in-degree*, and the number of edges pointing away from a node is its *out-degree*. In Fig. 3.1b, the two upper nodes have an in-degree of zero and an out-degree of two; while the two lower nodes have an in-degree of two and an out-degree of zero.

Given a graph $G = (V, E)$, a subgraph $G' = (V', E')$ of G, is a graph such that $V' \subseteq V$ and $E' \subseteq E$, in which each edge in E' is incident on vertices in V'. For example, if we take out the direction of the edges in the graph of Fig. 3.1b (making it an undirected graph), then the graph in Fig. 3.1a is a subgraph of Fig. 3.1b.

3.2 Types of Graphs

In addition to the two basic graphs, directed and undirected, there other types of graphs, such as:

Chain graph: a hybrid graph that has directed and undirected edges.
Simple graph: a graph that does not include cycles and parallel arcs.
Multigraph: a graph with several components (subgraphs), such that each component has no edges to the other components, i.e., they are disconnected.
Complete graph: a graph that has an edge between each pair of vertices.
Bipartite graph: a graph in which the vertices are divided into two subsets, G_1, G_2, such that all edges connect a vertex in G_1 with a vertex in G_2; that is, there are no edges between nodes in each subset.
Weighted graph: a graph that has weights associated to its edges and/or vertices.

Examples of these types of graphs are depicted in Fig. 3.3.

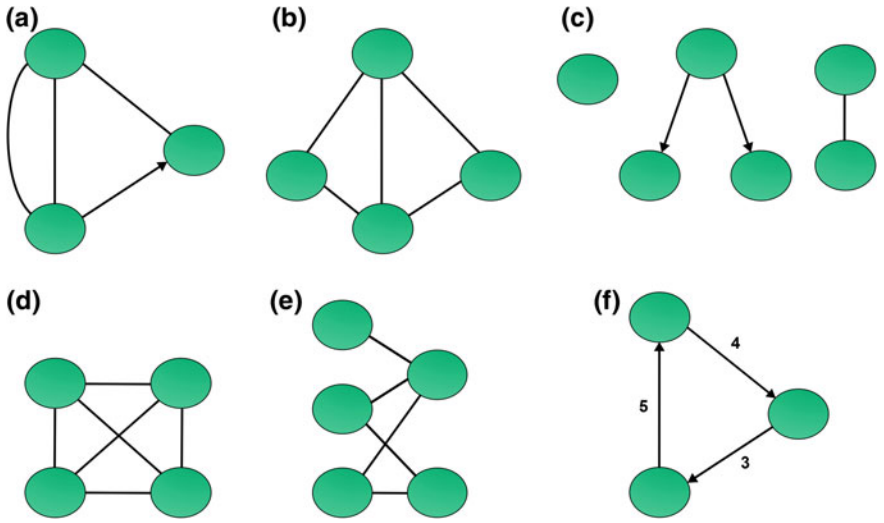


Fig. 3.3 Types of graphs: **a** chain graph, **b** simple graph, **c** multigraph, **d** complete graph, **e** bipartite graph, **f** weighted graph

3.3 Trajectories and Circuits

A *trajectory* is a sequence of edges, $E_1, E_2, \ldots, E_n$ such that the final vertex of each edge coincides with the initial vertex of the next edge in the sequence (except for the final vertex); that is, $E_i(V_j, V_k)$, $E_{i+1}(V_k, V_l)$, for $i = 1$ to $i = n - 1$. A *simple* trajectory does not include the same edge two o more times; an *elemental* trajectory is not incident on the same vertex more than once. Examples of different trajectories are illustrated in Fig. 3.4.

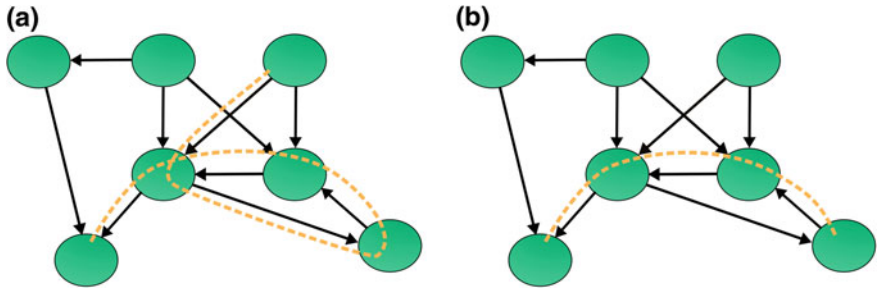


Fig. 3.4 Examples of trajectories: **a** a trajectory that is simple but not elemental, **b** a simple and elemental trajectory

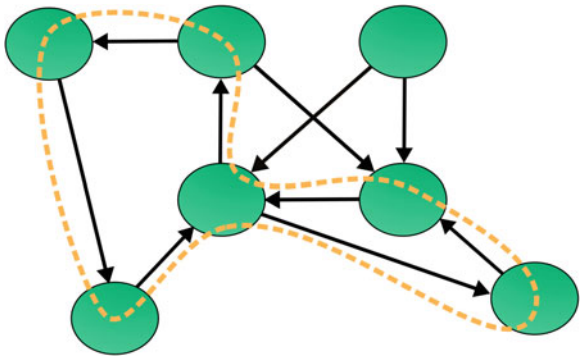


Fig. 3.5 Example of a circuit that is simple but not elemental

A graph G is *connected* if there is a trajectory between each pair of distinct vertices in G. If a graph G is not connected, each part that is connected is called a *component* of G.

A *circuit* is a trajectory such that the final vertex coincides with the initial one, i.e., it is a "closed trajectory". In an analogous way as with trajectories, we can define simple and elemental circuits. Figure 3.5 shows an example of a circuit.

An important type of graph for PGMs is a *Directed Acyclic Graph* (DAG). A DAG is a directed graph that has no directed circuits (a directed circuit is a circuit in which all edges in the sequence follow the directions of the arrows). For instance, Fig. 3.1b is a DAG and Fig. 3.3f is not a DAG.

Some classical problems in graph theory include trajectories and circuits, such as:

- Finding a trajectory that includes all edges in a graph only once (Euler trajectory).
- Finding a circuit that includes all edges in a graph only once (Euler circuit).
- Finding a trajectory that includes all vertices in a graph only once (Hamiltonian trajectory).
- Finding a circuit that includes all vertices in a graph only once (Hamiltonian circuit).
- Finding a Hamiltonian circuit in a weighted graph with minimum cost (Traveling salesman problem).[1]

The solution to these problems is beyond the scope of this book, the interested reader is referred to [2].

3.4 Graph Isomorphism

Two graphs are isomorphic if there is a one-to-one correspondence between their vertices and edges, so that the incidences are maintained. Given two graphs, G_1 and G_2, there are three basic types of isomorphisms:

[1] In this case the nodes represent cities and the edges roads with an associated distance or time, so the solution will provide a traveling salesman with the "best" (minimum distance or time) route to cover all the cities.

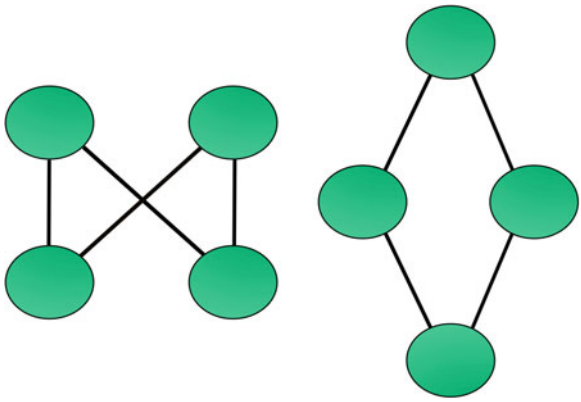


Fig. 3.6 These two graphs are isomorphic

1. *Graph isomorphism*. Graphs G_1 and G_2 are isomorphic.
2. *Subgraph isomorphism*. Graph G_1 is isomorphic to a subgraph of G_2 (or vice versa).
3. *Double subgraph isomorphism*. A subgraph of G_1 is isomorphic to a subgraph of G_2.

Figure 3.6 shows an example of two graphs that are isomorphic.

Determining if two graphs are isomorphic (type 1) is an NP problem; while the subgraph and double subgraph isomorphism problems (types 2 and 3) are NP-complete. See [1].

3.5 Trees

Trees are a type of graph which are very important in computer science in general, and for PGMs in particular. We will discuss two types of trees: undirected and directed.

An undirected tree is a connected graph that does not have simple circuits; Fig. 3.7 depicts an example of an undirected tree. There are two classes of vertices or nodes

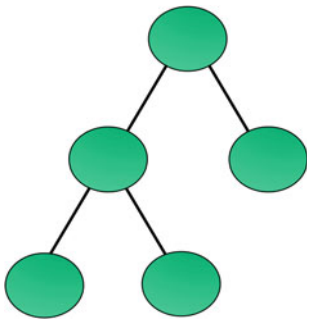


Fig. 3.7 An undirected tree. This tree has five nodes, three leaf nodes and two internal nodes

in an undirected tree: (i) leaf or terminal nodes, with degree one; (ii) internal nodes, with degree greater than one. Some basic properties of a tree are:

- There is a simple trajectory between each pair of vertices.
- The number of vertices, $|V|$, is equal to the number of edges, $|E|$ plus one: $|V| = |E| + 1$.
- A tree with two or more vertices has at least two leaf nodes.

A directed tree is a connected directed graph such that there is only a single directed trajectory between each pair of nodes (it is also known as a singly connected directed graph). There are two types of directed trees: (i) a rooted tree (or simply a tree), (ii) a polytree. A rooted tree has a single node with an in-degree of zero (the root node) and the rest have in-degree of one. A polytree might have more than one node with in-degree zero (roots), and certain nodes (zero or more) with in-degree greater than one (called multi-parent nodes). If we take out the direction of the edges in a polytree, it transforms into an undirected tree. We can think of a tree as a special case of a polytree. An example of a rooted tree and of a polytree is shown in Fig. 3.8.

Some relevant terminologies for directed trees are the following.

Root: a node with in-degree equal to zero.
Leaf: a node with out-degree equal to zero.
Internal node: a node with out-degree greater than zero.
Parent/Child: if there is a directed arc from A to B, A is parent of B and B is a child of A.
Brothers: two or more nodes are brothers if they have the same parent.
Ascendants /Descendants: if there is a directed trajectory from A to B, A is an ascendant of B and B is a descendant of A.
Subtree with root A: a subtree with A as its root.
Subtree of A: a subtree with a child of A as its root.
K-ary Tree: a tree in which each internal node has at most K children. It is a regular tree if each internal node has K children.
Binary Tree: a tree in which each internal node has at most two children.

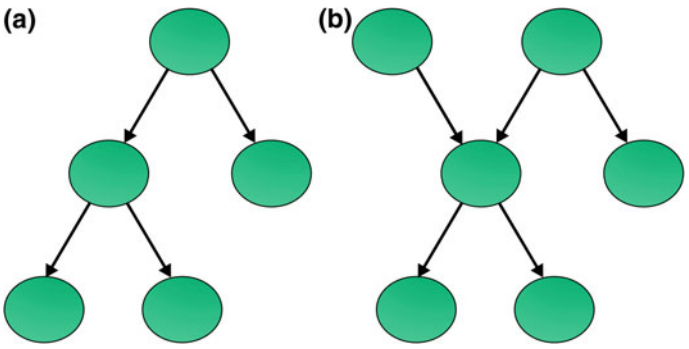


Fig. 3.8 **a** A rooted tree. **b** A polytree

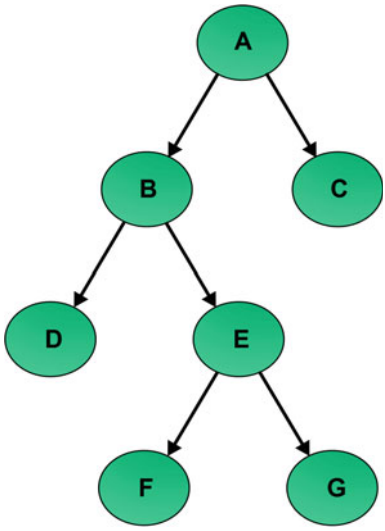


Fig. 3.9 An example of a directed tree to illustrate the terminology

For example, in the tree in Fig. 3.9; (i) A is a root node, (ii) C, D, F, G are leaf nodes, (iii) B, E are internal nodes, (iv) A is parent of B and B is child of A, (v) B and C are brothers, (vi) A is an ascendant of F and F is a descendant of A, (vii) the subtree B, D, E, F, G is a subtree with root B, (viii) the subtree E, F, G is a subtree of B. The tree in Fig. 3.9 is a nonregular binary tree.

3.6 Cliques

A *complete graph* is a graph, G_c, in which each pair of nodes is adjacent; that is, there is an edge between each pair of nodes. Figure 3.3d is an example of a complete graph. A *complete set*, W_c is a subset of G that induces a complete subgraph of G. It is a subset of vertices of G so that each pair of nodes in this subgraph is adjacent. For example, in Fig. 3.3d, each subset of three nodes in the graph is a complete set.

A *clique*, C, is a subset of graph G such that it is a complete set that is maximal; that is, there is no other complete set in G that contains C. The subsets of three nodes in Fig. 3.3d are not cliques, as these are not maximal; they are contained by the graph which is complete.

The graph in Fig. 3.10 has five cliques, one with four nodes, one with three nodes, and three with two nodes. Notice that every node in a graph is part of at least one clique; thus, the set of cliques of a graph always covers V.

The following sections cover some more advanced concepts of graph theory, as these are used by some of the inference algorithms for probabilistic graphical models.

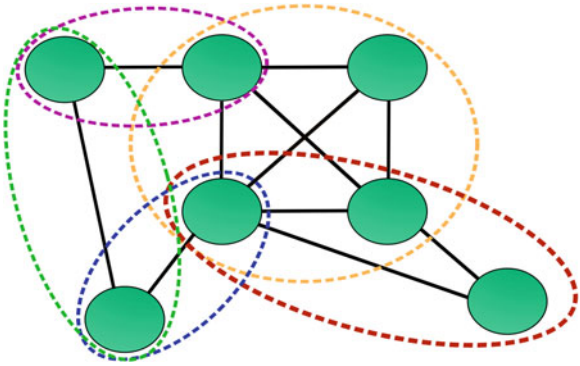


Fig. 3.10 Cliques: the five cliques in the graph are highlighted

3.7 Perfect Ordering

An ordering of the nodes in a graph consists in assigning an integer to each vertex. Given a graph $G = (V, E)$, with n vertices, then $\alpha = [V_1, V_2, \ldots, V_n]$ is an ordering of the graph; V_i is *before* V_j according to this ordering, if $i < j$.

An ordering α of a graph $G = (V, E)$ is a *perfect ordering* if all the adjacent vertices of each vertex V_i that are before V_i, according to this ordering, are completely connected. That is, for every i, $Adj(V_i) \cap \{V_1, V_2, \ldots, V_{i-1}\}$ is a complete subgraph of G. $Adj(V_i)$ is the subset of nodes in G that is adjacent to V_i. Figure 3.11 depicts an example of a perfect ordering.

Consider the set of cliques $C_1, C_2, \ldots C_p$ of an undirected connected graph G. In an analogous way as the ordering of nodes, we can define an ordering of the cliques, $\beta = [C_1, C_2, \ldots, C_p]$. An ordering β of the cliques has the *running intersection property*, if all the common nodes of each clique C_i with previous cliques according to this order are contained in a clique C_j; C_j is the *parent* of C_i. In other words, for

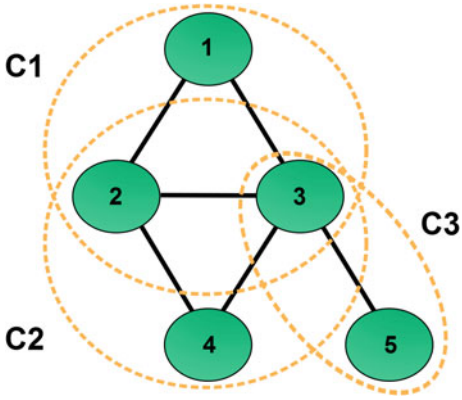


Fig. 3.11 An example of an ordering of nodes and cliques in a graph. In this case, the nodes have a perfect ordering, and the ordering of the cliques satisfies the running intersection property

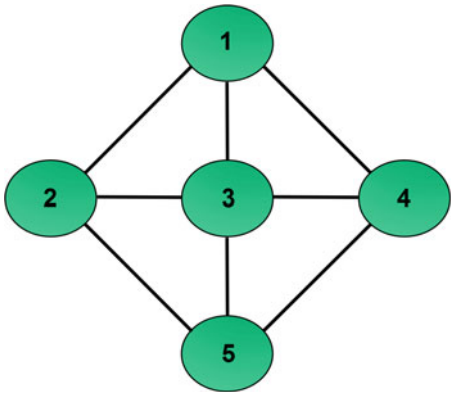


Fig. 3.12 An example of a graph that is not triangulated. The circuit 1, 2, 5, 4, 1 does not have a chord

every clique $i > 1$ there exists a clique $j < i$ such that $C_i \cap \{C_1, C_2, \ldots, C_{i-1}\} \subseteq C_j$. It is possible that a clique has more than one parent.

The cliques, C_1, C_2 and C_3 in Fig. 3.11 have a perfect ordering. In this example, C_1 is the parent of C_2, and C_1 and C_2 are parents of C_3.

A graph G is *triangulated* if every simple circuit of length greater than three in G has a chord. A chord is an edge that connects two of the vertices in the circuit and is not part of that circuit. For example, in Fig. 3.11 the circuit formed by the vertices 1, 2, 4, 3, 1 has a chord that connects nodes 2 and 3. The graph in Fig. 3.11 is triangulated. An example of a graph that is not triangulated is depicted in Fig. 3.12. Although visually this graph might seem triangulated, there is a circuit 1, 2, 5, 4, 1 that does not have any chord.

A condition for achieving a perfect ordering of the vertices, and having an ordering of the cliques that satisfies the running intersection property, is that the graph is triangulated. In the next section we will present algorithms for (i) ordering the nodes of a graph to achieve a prefect ordering, (ii) numerating the cliques to guarantee the running intersection property given a perfect ordering, and (iii) making a graph triangulated if it is not.

3.8 Ordering and Triangulation Algorithms

3.8.1 Maximum Cardinality Search

Given that a graph is triangulated, the following algorithm, known as *maximum cardinality search*, guarantees a perfect ordering. Given an undirected graph $G = (V, E)$ with n vertices:

Algorithm 3.1 Maximum Cardinality Search Algorithm

Select any vertex from V and assign it number 1.
while Not all vertices in G have been numbered **do**
From all the non-labeled vertices, select the one with higher number of adjacent labeled vertices and assign it the next number. Break ties arbitrarily.
end while

Given a perfect ordering of the vertices, it is easy to number the cliques so that the order satisfies the running intersection property. For this, the cliques are numbered in inverse order. Given p cliques, the clique that has the node with the highest number is assigned p; the clique that includes the next highest numbered node is assigned $p - 1$; and so on. This method can be illustrated with the example in Fig. 3.11. The node with the highest number is 5, so the clique that contains it is C_3. The next highest node is 4, so the clique that includes it is C_2. The remaining clique is C_1.

Now we will see how we can "fill-in" a graph to make it triangulated.

3.8.2 Graph Filling

The filling of a graph consists of adding arcs to an original graph G, to obtain a new graph, G_t, such that G_t is triangulated. Given an undirected graph $G = (V, E)$, with n nodes, the following algorithm makes the graph triangulated:

Algorithm 3.2 Graph Filling Algorithm

Order the vertices V with maximum cardinality search: $V_1, V_2, \ldots, V_n$.
for $i = n$ **to** $i = 1$ **do**
For node V_i, select all its adjacent nodes V_j such that $j > i$. Call this set of nodes A_i.
Add an arc from V_i to V_k if $k > i$ and $V_k \notin A_i$.
end for

For example, consider the graph in Fig. 3.12 which is not triangulated. If we apply the previous algorithm, we first order the nodes, generating the labeling illustrated in the figure. Next we process the nodes in inverse order and obtain the sets A for each node:

A_5: $\emptyset$
A_4: 5
A_3: 4, 5
A_2: 3, 5. An arc is added from 2 to 4.
A_1: 2, 3, 4. An arc is added from 1 to 5.

The resulting graph has two additional arcs $2 - 4$ and $1 - 5$ and we can verify that it is triangulated.

The filling algorithm guarantees that the resulting graph is triangulated, but generally it is not optimal in terms of adding the minimum number of additional arcs.

3.9 Additional Reading

There are several books in graph theory that cover most of the concepts introduced in this chapter in more detail, including [2–4]. Neapolitan [5], Chap. 3, covers the main graph theory background required for Bayesian networks, including the more advanced concepts. Some of the graph theory techniques from an algorithmic perspective are described in [1], including graph isomorphism.

3.10 Exercises

1. In the eighteenth century, the City of Könisberg (in Prussia, currently part of Russia) was divided into four parts connected by seven bridges. It is said that the residents tried to find a circuit along the entire city so that they crossed every bridge only once. Euler transformed the problem to a graph (illustrated at the beginning of the chapter) and established the condition for a circuit in a connected graph that passes through each edge exactly once: all the vertices in the graph must have an even degree. Determine if the residents of Könisberg were able to find a Euler circuit.
2. Prove the condition established by Euler: a graph G has a Euler circuit if and only if all the vertices in G have an even degree.
3. What is the condition for a graph to have a Euler trajectory?
4. Given the graph in Fig. 3.10, determine if it has (a) A Euler circuit (b) A Euler trajectory (c) A Hamiltonian circuit (d) A Hamiltonian trajectory.
5. Given the graph in Fig. 3.10, is it triangulated? If it is not triangulated, make it triangulated by applying the graph filling algorithm.
6. Prove that the number of vertices of odd degree in a graph is even.
7. Given the graph in Fig. 3.13, transform it to an undirected graph and order the nodes applying the maximum cardinality search algorithm.

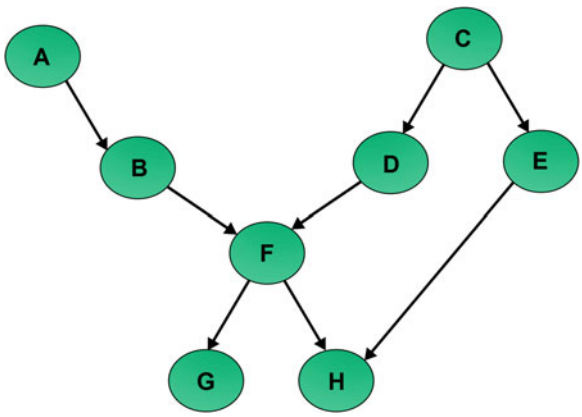


Fig. 3.13 A graph used in several exercises

8. Triangulate the graph of the previous problem.
9. Given the triangulated graph obtained in the previous problem: (a) find its cliques, (b) order the cliques according to the node ordering and verify that they satisfy the running intersection property, (c) show the resulting tree (or trees) of cliques.
10. *** Develop a program for generating a tree of cliques given an undirected graph. For this consider (a) ordering the nodes according to maximum cardinality search, (b) triangulating the graph using the graph filling algorithm, (c) finding the cliques of the resulting graph and numbering them. Think of an adequate data structure to represent the graph considering the implementation of the previous algorithms.
11. *** Incorporate to the program of the previous exercise some heuristics for selecting the node ordering such that the size of the biggest clique in the graph is minimized (this is important for the junction tree inference algorithm for Bayesian networks).

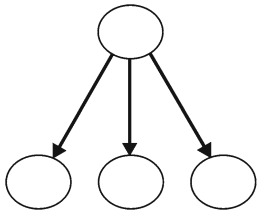
References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Boston (1974)
2. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Elsevier, The Netherlands (1994)
3. Gould, R.: Graph Theory. Benjamin/Cummings, Menlo Park (1988)
4. Gross, J.L., Yellen, J.: Graph Theory and its Applications. CRC Press, Boca Raton (2005)
5. Neapolitan, R.: Probabilistic Reasoning in Expert Systems: Theory and Algorithms. Wiley, New York (1990)

Part II
Probabilistic Models

The main types of probabilistic graphical models are presented, including Bayesian classifiers, hidden Markov models, Markov random fields, Bayesian networks, and dynamic and temporal Bayesian networks. A chapter is dedicated to each type of model, including representation, inference and learning; and practical application examples. The models covered in Part II do not include decisions, these are covered in Part III.

Chapter 4
Bayesian Classifiers



4.1 Introduction

Classification consists in assigning classes or labels to objects. There are two basic types of classification problems:

Unsupervised: in this case the classes are unknown, so the problem consists in dividing a set of objects into n groups or clusters, so that a class is assigned to each different group. It is also known as *clustering*.

Supervised: the possible classes or labels are known *a priori*, and the problem consists in finding a function or rule that assigns each object to one of the classes.

In both cases the objects are usually described by a set of properties or attributes. In this chapter we will focus on supervised classification.

From a probabilistic perspective, the problem of supervised classification consists in assigning to a particular object described by its attributes, $A_1, A_2, \ldots, A_n$, one of m classes, $C = \{c_1, c_2, \ldots, c_m\}$, such that the probability of the class given the attributes is maximized; that is:

$$Arg_C[Max\, P(C \mid A_1, A_2, \ldots, A_n)] \tag{4.1}$$

If we denote the set of attributes as $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$, Eq. (4.1) can be written as: $Arg_C[Max\, P(C \mid \mathbf{A})]$.

There are many ways to build a classifier, including decision trees, rules, neural networks, and support vector machines, among others.[1] In this book we will cover classification models based on probabilistic graphical models, in particular Bayesian classifiers. Before we describe the Bayesian classifier, we will present a brief introduction to how classifiers are evaluated.

[1] For an introduction and comparison of different types of classifiers we refer the interested reader to [10].

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_4

4.1.1 Classifier Evaluation

To compare different classification techniques, we can evaluate a classifier in terms of several aspects. Which aspects to evaluate, and the importance given to each aspect, depends on the final application of the classifiers. The main aspects to consider are:

Accuracy: it refers to how well a classifier predicts the correct class for unseen examples (that is, those not considered for learning the classifier).
Classification time: how long it takes the classification process to predict the class, once the classifier has been trained.
Training time: how much time is required to learn the classifier from data.
Memory requirements: how much space in terms of memory is required to store the classifier parameters.
Clarity: if the classifier is easily understood by a person.

Usually the most important aspect is accuracy. This can be measured by predicting N unseen data samples, and determining the percentage of correct predictions. Thus, the accuracy in percentage is:

$$Acc = (N_C/N) \times 100 \tag{4.2}$$

where N_C is the number of correct predictions.

When comparing classifiers, in general we want to maximize the classification accuracy; however, this is only *optimal* if the cost of a wrong classification is the same for all the classes. Consider a classifier used for predicting breast cancer. If the classifier predicts that a person has cancer but this is not true (false positive), this might produce an unnecessary treatment. On the other hand, if the classifier predicts no cancer and the person does have breast cancer (false negative), this might cause a delay in the treatment and even death. Clearly the consequences of the different types of errors are different.

When there is imbalance in the costs of misclassification, we must then minimize the expected cost (EC). For two classes, this is given by:

$$EC = FN \times P(-)C(- \mid +) + FP \times P(+)C(+ \mid -) \tag{4.3}$$

where: FN is the false negative rate, FP is the false positive rate, $P(+)$ is the probability of positive, $P(-)$ is the probability of negative, $C(- \mid +)$ is the cost of classifying a positive as negative, and $C(+ \mid -)$ is the cost of classifying a negative as positive. Determining these costs may be difficult for some applications.

4.2 Bayesian Classifier

The formulation of the *Bayesian Classifier* is based on the application of the Bayes rule to estimate the probability of each class given the attributes:

$$P(C \mid A_1, A_2, \ldots, A_n) = P(C)P(A_1, A_2, \ldots, A_n \mid C)/P(A_1, A_2, \ldots, A_n) \tag{4.4}$$

which can be written more compactly as:

$$P(C \mid \mathbf{A}) = P(C)P(\mathbf{A} \mid C)/P(\mathbf{A}) \tag{4.5}$$

The classification problem, based on Eq. (4.5), can be formulated as:

$$Arg_C[Max[P(C \mid \mathbf{A}) = P(C)P(\mathbf{A} \mid C)/P(\mathbf{A})]] \tag{4.6}$$

Equivalently, we can express Eq. (4.6) in terms of any function that varies monotonically with respect to $P(C \mid \mathbf{A})$, for instance:

- $Arg_C[Max[P(C)P(\mathbf{A} \mid C)]]$
- $Arg_C[Max[log(P(C)P(\mathbf{A} \mid C))]]$
- $Arg_C[Max[(log P(C) + log P(\mathbf{A} \mid C)]]$

Note that the probability of the attributes, $P(\mathbf{A})$, does not vary with respect to the class, so it can be considered as a constant for the maximization.

Based on the previous equivalent formulations for solving a classification problem we will require an estimate of $P(C)$, known as the *prior* probability of the classes, and $P(\mathbf{A} \mid C)$, known as the *likelihood*; $P(C \mid \mathbf{A})$ is the *posterior* probability. Therefore, to obtain the posterior probability of each class, we just need to multiply its prior probability by the likelihood which depends on the values of the attributes.[2]

The direct application of the Bayes rule results in a computationally expensive problem, as it was mentioned in Chap. 1. This is because the number of parameters in the likelihood term, $P(A_1, A_2, \ldots, A_n \mid C)$, increases exponentially with the number of attributes. This will not only imply a huge amount of memory to store all the parameters, but it will also be very difficult to estimate all the probabilities from the data or with the aid of a domain expert. Thus, the Bayesian classifier can only be of practical use for relatively *small* problems in terms of the number of attributes. An alternative is to consider some independence properties as in graphical models, in particular that all attributes are independent given the class, resulting in the *Naive Bayesian Classifier*.

4.2.1 Naive Bayes Classifier

The Naive or simple Bayesian classifier (NBC) is based on the assumption that all the attributes are independent given the class variable; that is, each attribute A_i is conditionally independent of all the other attributes given the class: $P(A_i \mid A_j, C) = P(A_i \mid C)$, $\forall j \neq i$. Under this assumption, Eq. (4.4) can be written as:

[2]The posterior probabilities of the classes will be affected by a constant as we are not considering the denominator in Eq. (4.6), that is, they will not add to one; however, they can be easily *normalized* by dividing each one by the sum for all classes.

$$P(C \mid A_1, A_2, \ldots, A_n) = P(C)P(A_1 \mid C)P(A_2 \mid C)\ldots P(A_n \mid C)/P(\mathbf{A}) \quad (4.7)$$

where $P(\mathbf{A})$ can be considered, as mentioned before, a normalization constant.

The Naive Bayes formulation drastically reduces the complexity of the Bayesian classifier, as in this case we only require the prior probability (one dimensional vector) of the class, and the n conditional probabilities of each attribute given the class (two-dimensional matrices) as the parameters for the model. That is, the space requirement is reduced from exponential to linear in the number of attributes. Also, the calculation of the posterior is greatly simplified, as to estimate it (unnormalized) only n multiplications are required.

A graphical representation of the NBC is shown in Fig. 4.1. This star-like structure depicts the property of conditional independence between all the attributes given the class—as there are no arcs between the attribute nodes.

Learning an NBC consists in estimating the prior probability of the class, $P(C)$, and the conditional probability of each attribute given the class, $P(A_i \mid C)$. These can be obtained via subjective estimates from an expert or from the data by maximum likelihood.[3]

The probabilities can be estimated from data using, for instance, maximum likelihood estimation. The prior probabilities of the class variable, C, are given by:

$$P(c_i) \sim N_i/N \quad (4.8)$$

where N_i is the number of times c_i occurs in the N samples.

The conditional probabilities of each attribute, A_j can be estimated as:

$$P(A_{jk} \mid c_i) \sim N_{jki}/N_i \quad (4.9)$$

where N_{jki} is the number of times that the attribute A_j takes the value k and it is from class i, and N_i is the number of samples of class c_i in the dataset.

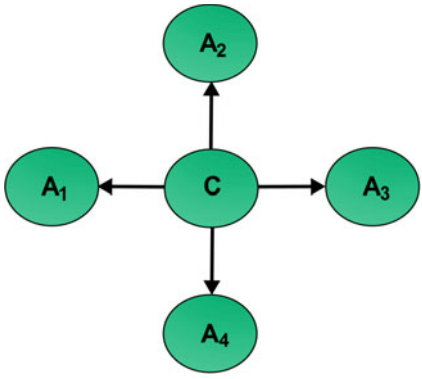


Fig. 4.1 An example of a Naive Bayesian classifier with the class variable, C, and four attributes, $A_1, \ldots, A_4$

[3] We will cover parameter estimation in detail in the chapter on Bayesian Networks.

Table 4.1 Dataset for the golf example

Outlook	Temperature	Humidity	Windy	Play
Sunny	High	High	False	No
Sunny	High	High	True	No
Overcast	High	High	False	Yes
Rain	Medium	High	False	Yes
Rain	Low	Normal	False	Yes
Rain	Low	Normal	True	No
Overcast	Low	Normal	True	Yes
Sunny	Medium	High	False	No
Sunny	Low	Normal	False	Yes
Rain	Medium	Normal	False	Yes
Sunny	Medium	Normal	True	Yes
Overcast	Medium	High	True	Yes
Overcast	High	Normal	False	Yes
Rain	Medium	High	True	No

Once the parameters have been estimated, the posterior probability can be obtained just by multiplying the prior by the likelihood for each attribute. Thus, given the values for m attributes, $a_1, \ldots, a_m$, for each class c_i, the posterior is proportional to:

$$P(c_i \mid a_1, \ldots, a_m) \sim P(c_i) P(a_1 \mid c_i) \ldots P(a_m \mid c_i) \tag{4.10}$$

The class c_k that maximizes the previous equation will be selected.[4]

Returning to the *golf* example, Table 4.1 depicts 14 records with 5 variables: four attributes (Outlook, Temperature, Humidity, Windy) and one class (Play). A NBC for the golf example is depicted in Fig. 4.2, including some of the required probability tables.

In summary, the main advantages of the Naive Bayesian classifier are:

- The low number of required parameters, which reduces the memory requirements and facilitates learning them from data.
- The low computational cost for inference (estimating the posteriors) and learning.
- The relatively good performance (classification precision) in many domains.
- A simple and intuitive model.

While its main limitations are the following:

- In some domains, the performance is reduced given that the conditional independence assumption is not valid.

[4]This assumes that the misclassification cost is the same for all classes; if these costs are not the same, the class the minimizes the misclassification cost should be selected.

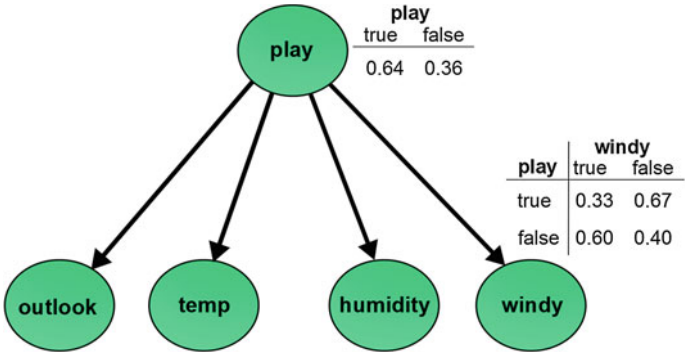


Fig. 4.2 A Naive Bayesian classifier for the golf example. Some of the parameters of the model are shown: P(play), P(windy | play)

- If there are continuous attributes, these need to be discretized (or consider alternative models such as the linear discriminator).

In the following sections we will focus on alternatives to solve the first limitation. First, we will cover other models that consider certain dependencies between attributes. Then we will describe techniques that can eliminate or join attributes, as another way to overcome the conditional independence assumption.

4.3 Alternative Models: TAN, BAN

The general Bayesian classifier and the Naive Bayesian classifier are the two extremes of possible dependency structures for Bayesian classifiers; the former represents the most complex structure with no independence assumptions, while the latter is the simplest structure that assumes that all the attributes are independent given the class. Between these two extremes there is a wide variety of possible models of varying complexities. Two interesting alternatives are the TAN and BAN classifiers.

The *Tree augmented Bayesian Classifier*, or TAN, incorporates some dependencies between the attributes by building a directed tree among the attribute variables. That is, the n attributes form a graph which is restricted to a directed tree that represents the dependency relations between the attributes. Additionally there is an arc between the class variables and each attribute. The structure of a TAN classifier is depicted in Fig. 4.3.

If we take out the limitation of a tree structure between attributes, we obtain the *Bayesian Network augmented Bayesian Classifier*, or BAN, which considers that the dependency structure among the attributes constitutes a directed acyclic graph (DAG). As with the TAN classifier, there is a directed arc between the class node and each attribute. The structure of a BAN classifier is depicted in Fig. 4.4.

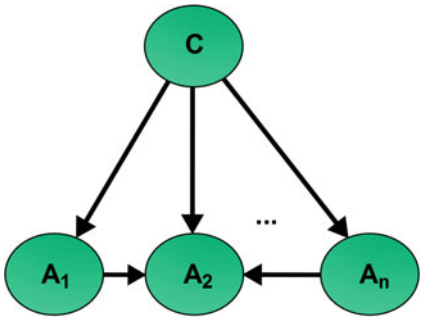


Fig. 4.3 An example of a TAN classifier

The posterior probability for the class variable given the attributes can be obtained in a similar way as with the NBC; however, now each attribute not only depends on the class but also on other attributes according to the structure of the graph. Thus, we need to consider the conditional probability of each attribute given the class and its *parent* attributes:

$$P(C \mid A_1, A_2, \ldots, A_n) = P(C)P(A_1 \mid C, Pa(A_1))P(A_2 \mid C, Pa(A_2)) \ldots P(A_n \mid C, Pa(A_n))/P(\mathbf{A}) \tag{4.11}$$

where $Pa(A_i)$ is the set of parent attributes of A_i according to the attribute dependency structure of the TAN or BAN classifier.

The TAN and BAN classifiers can be considered as particular cases of a more general model, that is, Bayesian networks, which we will cover in Chap. 7. In Chaps. 7 and 8, we will cover different techniques for inference and for learning Bayesian networks, which can be applied to obtain the posterior probabilities (inference) and the model (learning) for the TAN and BAN classifiers.

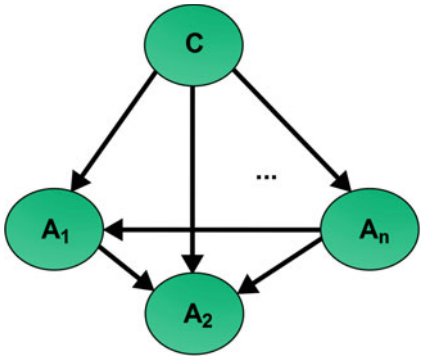


Fig. 4.4 An example of a BAN classifier

4.4 Semi-Naive Bayesian Classifiers

Another alternative to deal with dependent attributes is to transform the basic structure of a Naive Bayesian classifier, while maintaining a star or tree-structured network. This has the advantage that the efficiency and simplicity of the NBC is maintained, and at the same time the performance is improved for cases where the attributes are not independent. These types of Bayesian classifiers are known as *Semi-Naive Bayesian Classifiers* (SNBC), and several authors have proposed different variants of SNBCs [11, 16].

The basic idea of the SNBC is to eliminate or *join* attributes which are not independent given the class, such that the performance of the classifier improves. This is analogous to *feature selection* in machine learning, and there are two types of approaches:

Filter: the attributes are selected according to a local measure, for instance, the mutual information between the attribute and the class.

Wrapper: the attributes are selected based on a global measure, usually by comparing the performance of the classifier with and without the attribute.

Additionally, the learning algorithm can start from an empty structure and add (or combine) attributes; or from a full structure with all the attributes, and eliminate (or combine) attributes.

Figure 4.5 illustrates the two alternative operations to modify the structure of an NBC: (i) node elimination, and (ii) node combination, considering that we start from a full structure.

Node elimination consists in simply eliminating an attribute, A_i, from the model; this could be because it is not relevant for the class (A_i and C are independent), or because the attribute A_i and another attribute, A_j, are not independent given the class (which is a basic assumption of the NBC). The rationale for eliminating one of the dependent attributes is that if the attribute is not independent given the class, one of them is redundant and could be eliminated.

Node combination consists in merging two attributes, A_i and A_j, into a new attribute A_k, such that A_k has as possible values the cross product of the values of A_i and A_j (assuming discrete attributes). For example, if $A_i = a, b, c$ and $A_j = 1, 2$, then $A_k = a1, a2, b1, b2, c1, c2$. This is an alternative when two attributes are not independent given the class. By merging them into a single combined attribute, the independence condition is no longer relevant.

Thus, when two attributes are not independent given the class there are two alternatives: eliminate one or combine them into a single variable; in principle we should select the alternative that implies a higher improvement in the performance of the classifier, although in practice, this could be difficult to evaluate.

As mentioned before, there are several alternatives for learning an SNBC. A simple greedy scheme is outlined in Algorithm 4.1, where we start from a full NBC with all the attributes [9].

This process is repeated until there are no more superfluous or dependent attributes.

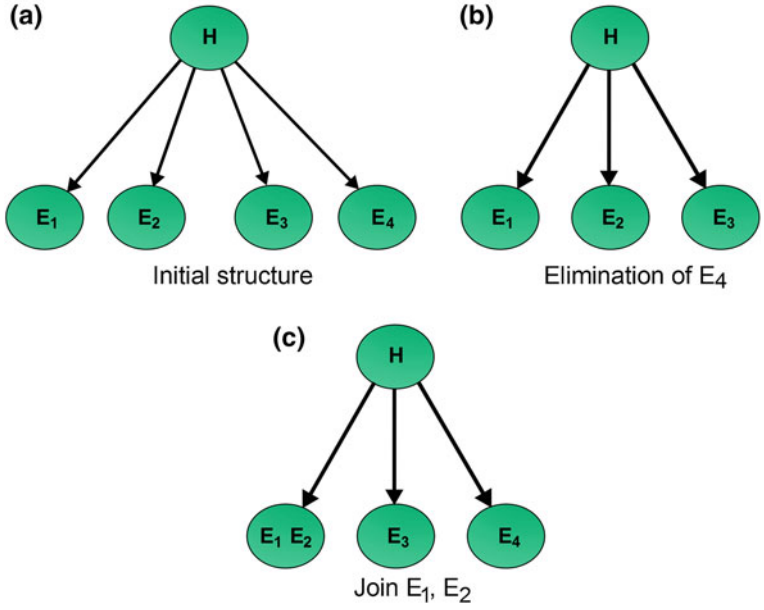


Fig. 4.5 Structural improvement: **a** original structure, **b** one attribute is eliminated, **c** two attributes are joined into one variable

References [8, 16] introduce an alternative operation for modifying the structure of an NBC, which consists in adding a new attribute that makes two dependent attributes independent; see Fig. 4.6. This new attribute is a kind of virtual or hidden node in the model, for which we do not have any data for learning its parameters. An alternative for estimating the parameters of hidden variables in Bayesian networks, such as in this case, is based on the Expectation-Maximization (EM) procedure, and it is described in Chap. 8.

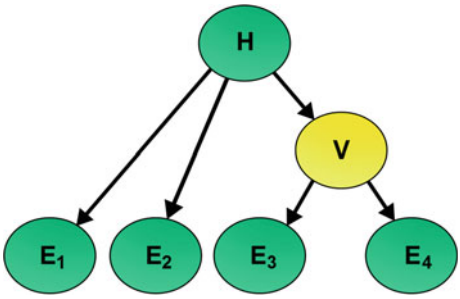


Fig. 4.6 An example of node creation for making two dependent attributes independent. Node V is inserted in the model given that E_3 and E_4 are not conditionally independent given H

All previous classifiers consider that there is a single class variable; that is, each instance belongs to one and only one class. Next we consider that an instance can belong to several classes, which is known as *multidimensional* classification.

Algorithm 4.1 Structural Improvement Algorithm

Require: A, the attributes
Require: C, the class
/* The dependency between each attribute and the class is estimated (for instance using mutual information).*/
for all $a \in \mathbf{A}$ **do**
/* Those attributes below a threshold are eliminated.*/
if $MI(a, C) < \varepsilon$ **then**
Eliminate a
end if
end for
/* The remaining attributes are tested to see if they are independent given the class, for example, using conditional mutual information (CMI).*/
for all $a \in \mathbf{A}$ **do**
for all $b \in \mathbf{A} - a$ **do**
/* Those attributes above a threshold are eliminated or combined based on which option gives the best classification performance.*/
if $CMI(a, b|C) > \omega$ **then**
Eliminate or Combine a and b
end if
end for
end for

4.5 Multidimensional Bayesian Classifiers

Several important problems need to predict several classes simultaneously. For example: text classification, where a document can be assigned to several topics; gene classification, as a gene may have different functions; image annotation, as an image may include several objects, among others. These are examples of *multidimensional classification*, in which more than one class can be assigned to an object. Formally, the *multidimensional classification* problem corresponds to searching for a function h that assigns to each instance represented by a vector of m features $\mathbf{X} = (X_1, \ldots, X_m)$ a vector of d class values $\mathbf{C} = (C_1, \ldots, C_d)$. The h function should assign to each instance $\mathbf{X}$ the most likely combination of classes, that is,

$$ArgMax_{c_1,\ldots,c_d} P(C_1 = c_1, \ldots, C_d = c_d|\mathbf{X}) \tag{4.12}$$

Multi-label classification is a particular case of multidimensional classification, where all class variables are binary. In the case of multi-label classification, there are two basic approaches: *binary relevance* and *label power-set* [18]. Binary relevance approaches transform the multi-label classification problem into d independent

binary classification problems, one for each class variable, $C_1, \ldots, C_d$. A classifier is independently learned for each class and the results are combined to determine the predicted class set; the dependencies between classes are not considered. The label power-set approach transforms the multi-label classification problem into a single-class scenario by defining a new compound class variable whose possible values are all the possible combinations of values of the original classes. In this case the interactions between classes are implicitly considered. It can be an effective approach for domains with only a few class variables; however, for many classes this approach is impractical. Essentially, binary relevance can be effective when the classes are relatively independent, and label power-set when there are few class variables.

Under the framework of Bayesian classifiers, we can consider two alternatives to the basic approaches. One is based on Bayesian networks, in which the dependencies between class variables (and also between attributes) are explicitly considered. The other implicitly incorporates the dependencies between classes by adding additional attributes to each independent classifier. Both are described in the following sections.

4.5.1 Multidimensional Bayesian Network Classifiers

A multidimensional Bayesian network classifier (MBC) over a set $\mathbf{V} = \{\mathbf{Z_1}, \ldots, \mathbf{Z_n}\}$, $n \geq 1$, of discrete random variables is a Bayesian network with a particular structure.[5] The set $\mathbf{V}$ of variables is partitioned into two sets $\mathbf{V_C} = \{\mathbf{C_1}, \ldots, \mathbf{C_d}\}, d \geq 1$, of class variables and $\mathbf{V_X} = \{\mathbf{X_1}, \ldots, \mathbf{X_m}\}, m \geq 1$, of feature variables ($d + m = n$). The set $\mathbf{A}$ of arcs is also partitioned into three sets, $\mathbf{A_C}, \mathbf{A_X}, \mathbf{A_{CX}}$, such that $\mathbf{A_C} \subseteq \mathbf{V_C} \times \mathbf{V_C}$ is composed of the arcs between the class variables, $\mathbf{A_X} \subseteq \mathbf{V_X} \times \mathbf{V_X}$ is composed of the arcs between the feature variables, and finally, $\mathbf{A_{CX}} \subseteq \mathbf{V_C} \times \mathbf{V_X}$ is composed of the arcs from the class variables to the feature variables. The corresponding induced sub-graphs are $\mathbf{G_C} = (\mathbf{V_C}, \mathbf{A_C})$, $\mathbf{G_X} = (\mathbf{V_X}, \mathbf{A_X})$ and $\mathbf{G_{CX}} = (\mathbf{V}, \mathbf{A_{CX}})$, called respectively *class*, *feature* and *bridge* subgraphs (see Fig. 4.7).

Different graphical structures for the class and feature subgraphs may lead to different families of MBCs [19]. For instance, we could restrict the class subgraph to a tree, and assume that the attributes are independent given the class variables. Or, we could have the same structure for both subgraphs, such as tree–tree, polytree–polytree or DAG –DAG. As we consider more complex structures for each subgraph, the complexity of learning these structures increases.

The problem of obtaining the classification of an instance with an MBC, that is, the most likely combination of classes, corresponds to the MPE (Most Probable Explanation) or *abduction* problem for Bayesian networks. In other words, determining the most probable values for the class variables $\mathbf{V} = \{\mathbf{C_1}, \ldots, \mathbf{C_n}\}$, given the features. This is a complex problem with high computational cost. There are a few ways to try to reduce the time complexity [2], however, this approach is still limited to problems with only a limited number of classes.

[5] Bayesian networks are introduced in Chap. 7.

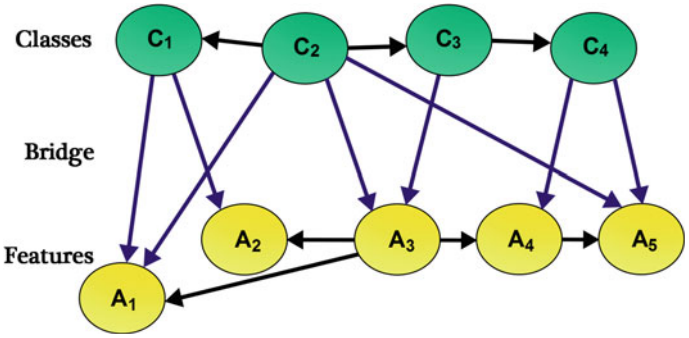


Fig. 4.7 A multidimensional Bayesian network classifier, showing the three subgraphs: classes, features and bridge

We will describe how to learn Bayesian networks (MBCs are a particular type of Bayesian networks) and the calculation of the MPE in Chaps. 7 and 8.

4.5.2 Bayesian Chain Classifiers

Chain classifiers are an alternative method for multi-label classification that incorporate class dependencies, while keeping the computational efficiency of the binary relevance approach [12]. A chain classifier consists of d base binary classifiers which are linked in a chain, such that each classifier incorporates the classes predicted by the previous classifiers as additional attributes. Thus, the feature vector for each binary classifier, L_i, is extended with the labels (0/1) for all previous classifiers in the chain. Each classifier in the chain is trained to learn the association of label l_i given the features augmented with all previous class labels in the chain, $L_1, L_2, \ldots, L_{i-1}$. For classification, it starts at L_1, and propagates the predicted classes along the chain, such that for $L_i \in \mathscr{L}$ (where $\mathscr{L} = \{L_1, L_2, \ldots, L_d\}$) it predicts $P(L_i \mid \mathbf{X}, L_1, L_2, \ldots, L_{i-1})$. As in the binary relevance approach, the class vector is determined by combining the outputs of all the binary classifiers in the chain.

Bayesian chain classifiers are a type of chain classifier under a probabilistic framework. If we apply the chain rule of probability theory, we can rewrite Eq. (4.12):

$$ArgMax_{C_1,\ldots,c_d} P(C_1|C_2,\ldots,C_d,\mathbf{X})P(C_2|C_3,\ldots,C_d,\mathbf{X})\ldots P(C_d|\mathbf{X}) \quad (4.13)$$

If we consider the dependency relations between the class variables, and represent these relations as a directed acyclic graph (DAG), then we can simplify Eq. (4.13) by considering the independencies implied in the graph so that only the *parents* of each class variable are included in the chain, and all other *previous* classes according to the chain order are eliminated. We can rewrite Eq. (4.13) as:

$$ArgMax_{C_1,\ldots,C_d} \prod_{i=1}^{d} P(C_i|\mathbf{Pa}(C_i), \mathbf{X}) \tag{4.14}$$

where $\mathbf{Pa}(C_i)$ are the parents of class i in the DAG that represents the dependencies between the class variables.

Next we make a further simplification by assuming that the most probable joint combination of classes can be approximated by simply concatenating the individual most probable classes. That is, we solve the following set of equations as an approximation of Eq. (4.14):

$$\begin{aligned} &ArgMax_{C_1} P(C_1|\mathbf{Pa}(C_1), \mathbf{X}) \\ &ArgMax_{C_2} P(C_2|\mathbf{Pa}(C_2), \mathbf{X}) \\ &\qquad \ldots\ldots\ldots\ldots\ldots\ldots \\ &ArgMax_{C_d} P(C_d|\mathbf{Pa}(C_d), \mathbf{X}) \end{aligned}$$

This last approximation corresponds to a Bayesian chain classifier. Thus, a BCC makes two basic assumptions:

1. The class dependency structure given the features can be represented by a DAG.
2. The most probable joint combination of class assignments (total abduction) is approximated by the concatenation of the most probable individual classes.

The first assumption is reasonable if we have enough data to obtain a good approximation of the class dependency structure, and assuming that this is obtained conditioned on the features. With respect to the second assumption, it is well known that the total abduction or most probable explanation is not always equivalent to the maximization of the individual classes. However, the assumption is less strong than that assumed by the binary relevance approach. Bayesian chain classifiers provide an attractive alternative to multidimensional classification, as they incorporate in certain way the dependencies between class variables, and they keep the efficiency of the Binary relevance approach.

For the *base* classifier belonging to each class we can use any of the Bayesian classifiers presented in the previous sections, for instance a NBC. Assuming that we have a class dependency structure represented as a DAG (this structure can be learned from data, see Chap. 8), each classifier can be learned in a similar way as a NBC, by simply including as additional attributes the class variables according to the class dependency structure. The simplest option is to only include the parent nodes of each class according to this dependency graph. The general idea for building a BCC is illustrated in Fig. 4.8.

For classifying an instance, all the classifiers are applied simultaneously, and all the classes that have a posterior probability above a threshold are returned as the output.

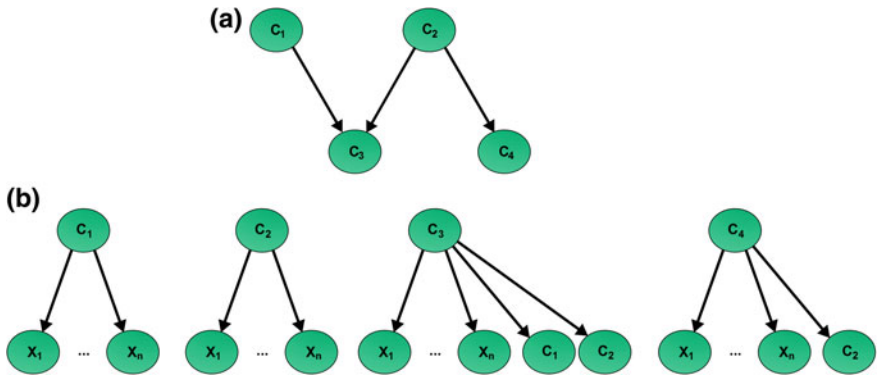


Fig. 4.8 An example of a BCC. **a** A BN that represents the class dependency structure. **b** Naive Bayes classifiers, one for each class. Each base classifier defined for C_i includes the set of attributes, $X_1, \ldots, X_n$, plus its parents in the dependency structure as an additional attribute

4.6 Hierarchical Classification

Hierarchical classification is a type of multidimensional classification in which the classes are ordered in a predefined structure, typically a tree, or in general a directed acyclic graph (DAG). By taking into account the hierarchical organization of the classes, the classification performance can be improved. In hierarchical classification, an example that belongs to certain class automatically belongs to all its superclasses; this is known as the *hierarchy constraint*. Hierarchical classification has application in several areas, such as text categorization, protein function prediction, and object recognition.

As in the case of multidimensional classification, there are two basic approaches for hierarchical classification: global classifiers and local classifiers. The global approach builds a classifier to predicts all the classes at once; this becomes too complex computationally for large hierarchies. The local classifiers schemes train several classifiers and combine their outputs. There are three basic approaches. A Local Classifier per hierarchy Level, that trains one multi-class classifier for each level of the class hierarchy. Local binary Classifier per Node, in which a binary classifier is built for each node (class) in the hierarchy, except the root node. Local Classifier per Parent Node (LCPN), where a multi-class classifier is trained to predict its child nodes.

Local methods commonly use a top-down approach for classification [14]; the classifier at the top level selects certain class, so the other classes are discarded, and then the successors of the selected class are analyzed, and so on. This has the problem that if there is an error at the upper levels of the hierarchy, this can not be recovered and it propagates down to the other levels. An alternative approach is to analyze the paths in the hierarchy, and select the *best* path according to the results of the local classifiers. A method based on this idea is described next.

4.6.1 Chained Path Evaluation

Chained Path Evaluation (CPE) [13] analyzes each possible path from the root to a leaf node in the hierarchy, taking into account the level of the predicted labels to give a score to each path and finally return the one with the best score. Additionally, it considers the relations of each node with its ancestors in the hierarchy, based on chain classifiers. CPE consists of two parts, training and classification.

4.6.1.1 Training

A local classifier is trained for each node, C_i, in the hierarchy, except the leaf nodes, to classify its child nodes; that is, using the LCPN scheme, see Fig. 4.9. The classifier for each node, C_i, for instance a Naive Bayes classifier, is trained considering examples from all it child nodes, as well as some examples of it sibling nodes in the hierarchy. For instance, the classifier C_2 in Fig. 4.9 will be trained to classify C_5, C_6, C_7; additional examples will be considered from the classes C_3 and C_4, which represent an *unknown* class for C_1.

To consider the relation with other nodes in the hierarchy, the class predicted by the parent (tree structure) or parents (DAG), are included as additional attributes in each local classifier, inspired by Bayesian chain classifiers. For example, the classifier C_2 in Fig. 4.9 will have as an additional attribute the class predicted by its parent, C_1.

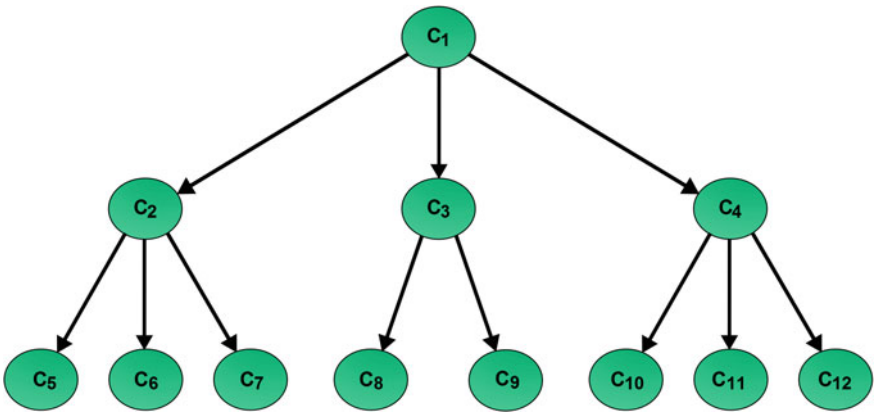


Fig. 4.9 Example of a hierarchical structure (a tree). For each non-leaf node, a local classifier is trained to predict its child nodes: C_1 classifies C_2, C_3, C_4; C_2 classifies C_5, C_6, C_7; and similarly for C_3 and C_4

4.6.1.2 Classification

In the classification phase, the probabilities for each class for all local classifiers are obtained based on the input data (features of each instance). After computing the probabilities for each non-leaf node in the hierarchy, these are combined to obtain a score for each path.

The *score* for each path in the hierarchy is calculated by a weighted sum of the log of the probabilities of all local classifiers in the path:

$$score = \sum_{i=0}^{n} w_{C_i} \times \log(P(C_i|X_i, pa(C_i))) \tag{4.15}$$

where C_i are the classes for each LCPN, X_i is the vector of attributes, $pa(C_i)$ is the parent predicted class, and w_{C_i} is a weight. The purpose of these weights is to give more importance to the upper levels of the hierarchy, as errors at the upper hierarchy levels (which correspond to more generic concepts) are more *expensive* than those at the lower levels (which correspond to more specific concepts) [13]. Taking the sum of logarithms is used to ensure numerical stability when computing the probability for long paths.

Once the scores for all the paths are obtained, the path with the highest score will be selected as the set of classes corresponding to certain instance. For the example in Fig. 4.9, the score will be calculated for each path from the root to each leaf node: Path 1: $C_1 - C_2 - C_5$, Path 2: $C_1 - C_2 - C_6$, etc. In this case there are eight paths. Suppose the path with the highest score is Path 4: $C_1 - C_3 - C_8$; then these three classes will be returned as the output of the classifier.

4.7 Applications

In this section we show the application of two types of Bayesian classifiers in two practical problems. First we illustrate the use of the semi-Naive classifier for labeling pixels in an image as skin or not skin. Then we demonstrate the use of multidimensional chain classifiers for HIV drug selection.

4.7.1 Visual Skin Detection

Skin detection is a useful preprocessing stage for many applications in computer vision, such as person detection and gesture recognition, among others. Thus it is critical to have a very accurate and efficient classifier. A simple and very fast way to obtain an approximate classification of pixels in an image as *skin* or *not-skin* is based on the color attributes of each pixel. Usually, pixels in a digital image are represented as the combination of three basic (primary) colors: Red (R), Green (G), and Blue

(B), in what is known as the RGB model. Each color component can take numeric values in a certain interval, i.e., 0...255. There are alternative color models, such as *HSV*, *YIQ*, etc.

An NBC can be built to classify pixels as skin or not-skin using the three color values—RGB—as attributes. However, it is possible for a different color model to produce a better classification. Alternatively, we can combine several color models into a single classifier, having as attributes all the attributes from the different models. This last option has the potential of taking advantage of the information provided by different models; however, if we use an NBC the independence assumption is violated—the different models are not independent, one model could be derived from another.

An alternative is to consider a semi-Naive Bayesian classifier and *select* the best attributes from the different color models for skin classification by eliminating or joining attributes. For this we used three different color models: RGB, HSV, and YIQ, so there are nine attributes in total. The attributes (color components) were previously discretized into a reduced number of intervals. Then an initial NBC was learned based on data—examples of skin and not-skin pixels taken from several images. This initial classifier obtained a 94 % accuracy when applied to other (test) images.

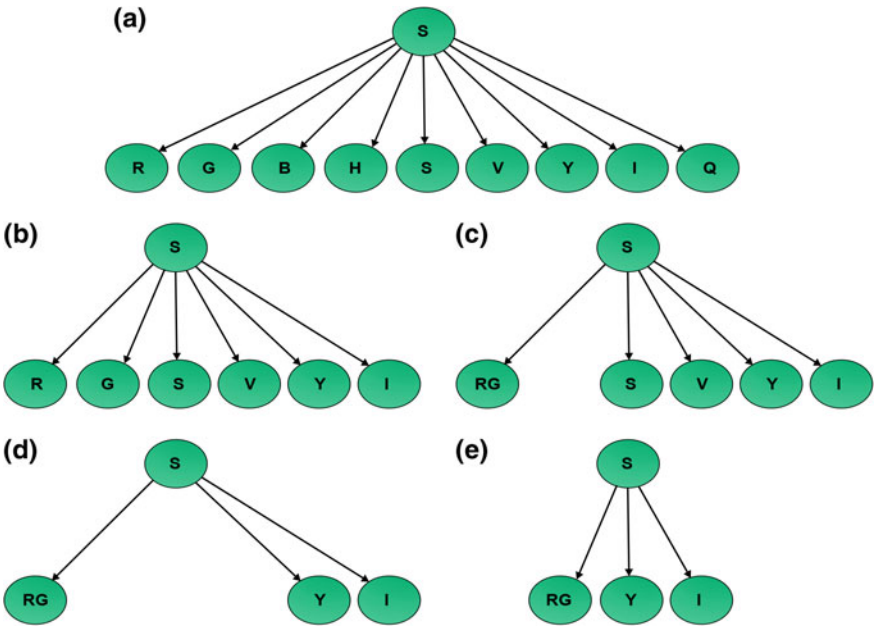


Fig. 4.10 The figure illustrates the process of optimizing the semi-Naive Bayesian classifier for skin detection, from the initial model with nine attributes (**a**) to the final one with three attributes (**e**). **a** Initial structure wih color model RGB, HSV and YIG. **b** Eliminate B, Q and H. **c** Join R and G. **d** Eliminate V and S. **e** Final model

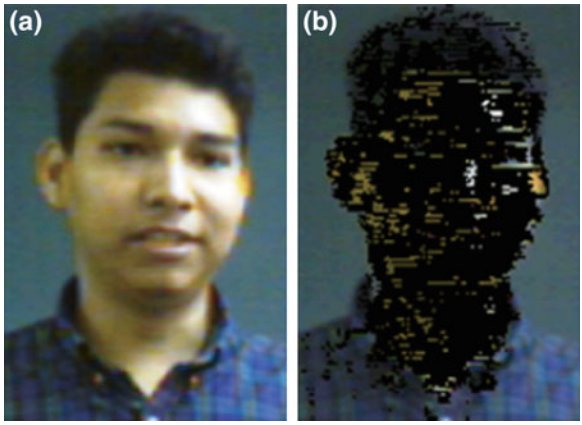


Fig. 4.11 An example of an image in which the pixels have been classified as skin or not skin. **a** Original image. **b** Image with skin pixels detected (in black)

The classifier was then *optimized* using the method described in Sect. 4.4. Starting from the *full* NBC with nine attributes, the method applies the variable elimination and combination stages until the *simplest* classifier with maximum accuracy is obtained. The sequence of operations and the final structure are depicted in Fig. 4.10. We can observe that initially the algorithm eliminates a number of irrelevant or redundant attributes, it then combines two dependent attributes and subsequently eliminates two more attributes, until it arrives to the final structure with three attributes: RG, Y, I (one is a combination of two original attributes). This final model was evaluated with the same test images and the accuracy improved to 98 %. An example of an image with the skin pixels detected by this classifier is shown in Fig. 4.11.

In this example, the SNBC obtains a significant advantage compared to the original NBC, while at the same time producing a simpler model (in terms of the number of variables and parameters required) [9].

4.7.2 HIV Drug Selection

The Human Immunodeficiency Virus (HIV) is the causing agent of AIDS, a condition in which progressive failure of the immune system allows opportunistic life-threatening infections to occur. To combat HIV infection several antiretroviral (ARV) drugs belonging to different drug classes that affect specific steps in the viral replication cycle have been developed. Antiretroviral therapy (ART) generally consists of combinations of three or four ARV drugs. Selecting a drug combination depends on the patient's condition, which can be characterized according to the mutations of the virus present in the patient. Thus, it is important to select the best drug combination according to the virus' mutations in a patient.

Selecting the best group of antiretroviral drugs for a patient can be seen as an instance of a multi-label classification problem, in which the classes are the different

types of antiretroviral drugs and the attributes are the virus mutations. Since multi-label classification is a subcase of multidimensional classification, this particular problem can be accurately modeled with an MBC. By applying a learning algorithm we can discover the relations that exist between antiretrovirals and mutations, while also retrieving a model with a high predictive power.

An alternative for learning an MBC is the MB-MBC algorithm [3]. This particular algorithm uses the *Markov blanket* of each class variable to lighten the computational burden of learning the MBC by filtering out those variables that do not improve classification. A Markov blanket of a variable C, denoted as $MB(C)$ is the minimal set of variables such that $I(C, \mathbf{S}|MB(C))$ is true for every variable subset $\mathbf{S}$, where $\mathbf{S}$ does not have as members any variables that belong to $MB(C) \cup C$. In other words, the Markov blanket of C is the minimal set of variables under which C is conditionally independent of all remaining variables.

To predict the most likely group of antiretrovirals drugs for a patient given the viral mutations present, the Markov blanket for each antiretroviral is learned. For example, if we consider a group of reverse transcriptase inhibitors (an antiretroviral drug group that attacks the reverse transcriptase phase of the viral HIV lifecycle) as the class variables, and a group of mutations as the attributes, the Markov blanket for the entire set of reverse transcriptase inhibitors is learned to determine the existing relations of type antiretroviral–antiretroviral and antiretroviral-mutation. Learning the Markov blanket of each class variable corresponds to learning an undirected structure for the MBC, i.e., the three subgraphs. Finally, directionality for all three subgraphs is determined in the last step of the MB-MBC algorithm. Figure 4.12 shows the resulting MBC.

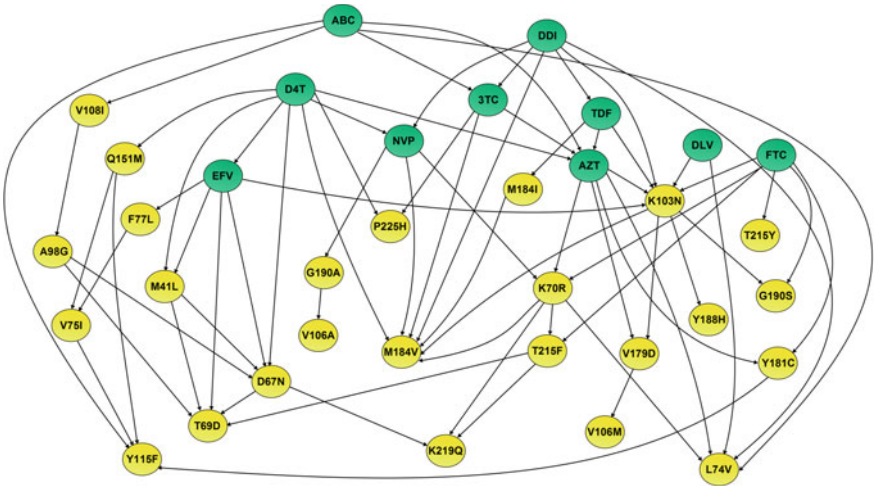


Fig. 4.12 A graphical structure for the multidimensional Bayesian network classifier learned with the MB-MBC algorithm for a set of reverse transcriptase inhibitors (*green*) and a set of mutations (*yellow*) (Figure based on [3])

4.8 Additional Reading

For a general introduction and comparison of different classification approaches see [10]. The consideration of classification costs is described in [5]. The TAN and BAN classifiers are described in [6]; a comparison of different BN classifiers is presented in [4]. The semi-Naive approach was initially introduced in [16], and later extended by [11]. Reference [18] presents an overview of multidimensional classifiers. Different alternatives for MBC are presented in [2]. Chain classifiers were introduced in [12], and Bayesian chain classifiers in [17]. A review of different approaches for hierarchical classification and their applications is presented in [15].

4.9 Exercises

1. Given the data for the *golf* example in Table 4.1, complete the CPTs for the NBC using maximum likelihood estimation.
2. Obtain the class with the maximum probability for the *golf* NBC of the previous problem, considering all the combinations of values of the attributes.
3. Based on the results of the previous problem, design a set of classification rules that are equivalent to the NBC for determining play/no-play based on the attribute values.
4. Consider that we transform the NBC for *golf* to a TAN with the following dependency structure for the attributes: *outlook* $\rightarrow$ *temperature*, *outlook* $\rightarrow$ *humidity*, *temperature* $\rightarrow$ *wind*. Using the same dataset, estimate the CPTs for this TAN.
5. Given the data set for the *golf* example, estimate the mutual information between the class and each attribute. Build a semi-Naive Bayesian classifier by eliminating those attributes that have a low mutual information with the class (define a threshold).
6. Extend the previous problem by now estimating the mutual information between each pair of attributes given the class variable. Eliminate or join those attributes that are not conditionally independent given the class according to a predefined threshold. Show the structure and parameters of the resulting classifiers.
7. Consider that we transform the *golf* example to a multidimensional problem such that there are two classes, play and outlook, and three attributes, temperature, humidity and windy. Consider that we build a multidimensional classifier based on binary relevance—an independent classifier for each class variable. Given that each classifier is a NBC, what will the structure of the resulting classifier be? Obtain the parameters for this classifier based on the same data in Table 4.1.
8. For the previous problem, consider that we now build a NBC based on the power set approach. What will the structure and parameters of the resulting model be? Use the same dataset.
9. *** Compare the structure, complexity (in terms of the number of parameters) and classification accuracy of different Bayesian classifiers—NBC, TAN, BAN—

using several datasets (use, for example, the WEKA [7] implementation of the Bayesian classifiers; and some of the data sets from the UCI repository [1]). Do TAN or BAN always outperform the Naive Bayes classifier? Why?

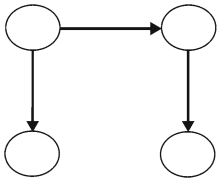
10. *** A hierarchical classifier is a particular type of multidimensional classifier in which the classes are arranged in a hierarchy; for instance, an animal hierarchy. A restriction of a hierarchy is that an instance that belongs to a certain class, must belong to all its superclasses in the hierarchy (hierarchical constraint). How can a multidimensional classifier be designed to guarantee the hierarchical constraint? Extend the Bayesian chain classifier for hierarchical classification.

References

1. Bache, K., Lichman, M.: UCI machine learning repository. University of California, School of Information and Computer Science. Irvine. http://archive.ics.uci.edu/ml. Accessed 22 Sept 2014 (2013)
2. Bielza, C., Li, G., Larrañaga, P.: Multi-dimensional classification with bayesian networks. Int. J. Approx. Reason. **52**, 705–727 (2011)
3. Borchani, H., Bielza, C., Toro, C., Larrañaga, P.: Predicting human immunodeficiency virus inhibitors using multi-dimensional Bayesian network classifiers. Artif. Intell. Med. **57**, 219–229 (2013)
4. Cheng, J., Greiner, R.: Comparing Bayesian network classifiers. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 101–108 (1999)
5. Drummond, C., Holte, R.C.: Explicitly representing expected cost: an alternative to the ROC representation. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 198–207 (2000)
6. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Mach. Learn. **29**, 131–163 (1997)
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B. and Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. In: ACM SIGKDD Explorations Newsletter. ACM, pp. 10–18 (2009)
8. Kwoh, C.K., Gillies, D.F.: Using hidden nodes in Bayesian networks. Artificial Intelligence, vol. 88, pp. 1–38. Elsevier, Essex (1996)
9. Martinez, M., Sucar, L.E.: Learning an optimal naive Bayes classifier. In: International Conference on Pattern Recognition (ICPR), vol. 3, pp. 1236–1239 (2006)
10. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine Learning, Neural and Statistical Classification. Ellis Howard, England (2004)
11. Pazzani, M.J.: Searching for Dependencies in Bayesian Classifiers. Artificial Intelligence and Statistics IV. Lecture Notes in Statistics, Springer-Verlag, New York (1997)
12. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. In: Proceedings ECML/PKDD, pp. 254–269 (2009)
13. Ramírez, M., Sucar, L.E., Morales, E.: Path evaluation for hierarchical multi-label classification. In: Proceedings of the Twenty-Seventh International Florida Artificial Intelligence Research Society Conference (FLAIRS), pp. 502–507 (2014)
14. Silla Jr., C.N., Freitas, A.A.: Novel top-down approaches for hierarchical classification and their application to automatic music genre classification. In: IEEE International Conference on Systems, Man, and Cybernetics, pp. 3499–3504. October 2009
15. Silla Jr, C.N., Freitas, A.A.: A survey of hierarchical classification across different application domains. Data Min. Knowl. Discov. **22**(1–2), 31–72 (2011)
16. Sucar, L.E., Gillies, D.F., Gillies, D.A.: Objective probabilities in expert systems. Artif. Intell. **61**, 187–208 (1993)

17. Sucar, L.E., Bielza, C., Morales, E., Hernandez, P., Zaragoza, J., Larrañaga, P.: Multi-label classification with Bayesian network-based chain classifiers. Pattern Recognit. Lett. **41**, 14–22 (2014)
18. Tsoumakas, G., Katakis, I.: Multi-label classification: an overview. Int. J. Data Wareh. Min. **3**, 1–13 (2007)
19. van der Gaag L.C., de Waal, P.R.: Multi-dimensional Bayesian network classifiers. In: Third European Conference on Probabilistic Graphic Models, pp. 107–114. Prague, Czech Republic (2006)

Chapter 5
Hidden Markov Models



5.1 Introduction

Markov Chains are another class of PGMs that represent dynamic processes, in particular how the state of a process changes with time. For instance, consider that we are modeling how the weather in a particular place changes over time. In a very simplified model, we assume that the weather is constant throughout the day, and can have three possible states: *sunny, cloudy, raining*. Additionally, we assume that the weather on a certain day only depends on the previous day. Thus, we can think of this simple weather model as a Markov chain in which there is a state variable per day, with three possible values; these variables are linked in a *chain*, with a directed arc from one day to the next, (see Fig. 5.1). This implies what is known as the *Markov property*, the state of the weather for the next day, S_{t+1}, is independent of all previous days given the present weather, S_t, i.e., $P(S_{t+1} \mid S_t, S_{t-1}, \ldots) = P(S_{t+1} \mid S_t)$. Thus, in a Markov chain the main parameter required is the probability of a state given the previous one.

The previous model assumes that we can measure the weather with precision each day, that is, the state is *observable*. However, this is not necessarily true. In many applications we cannot observe the state of the process directly, so we have what is called a *Hidden Markov Model*, where the state is hidden. In this case, in addition to the probability of the next state given the current state, there is another parameter which models the uncertainty about the state, represented as the probability of the *observation* given the state, $P(O_t \mid S_t)$. This type of model is more powerful than the simple Markov chain and has many applications, for example, in speech and gesture recognition.

After a brief introduction to Markov chains, in the following sections we will discuss hidden Markov models in detail, including how the computations of interest for this type of model are solved. Then we discuss several extensions to standard HMMs, and we conclude the chapter with two application examples.

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_5

Fig. 5.1 The figure illustrates a Markov chain where each node represents the state at certain point in time

5.2 Markov Chains

A Markov chain (MC) is a *state machine* that has a discrete number of states, $q_1, q_2, \ldots, q_n$, and the transitions between states are nondeterministic, i.e., there is a probability of transiting from a state q_i to another state q_j: $P(S_t = q_j \mid S_{t-1} = q_i)$. Time is also discrete, such that the chain can be at a certain state q_i for each time step t. It satisfies the Markov property, that is, the probability of the next state only depends on the current state.

Formally, a Markov chain is defined as

Set of states: $Q = \{q_1, q_2, \ldots, q_n\}$
Vector of prior probabilities: $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, where $\pi_i = P(S_0 = q_i)$
Matrix of transition probabilities: $A = \{a_{ij}\}$, $i = [1..n]$, $j = [1..n]$, where $a_{ij} = P(S_t = q_j \mid S_{t-1} = q_i)$

where n is the number of states, and S_0 is the initial state. In a compact way, an MC is represented as $\lambda = \{A, \Pi\}$.

A (first order) Markov chain satisfies the following properties:

1. Probability axioms: $\sum_i \pi_i = 1$ and $\sum_j a_{ij} = 1$
2. Markov property: $P(S_t = q_j \mid S_{t-1} = q_i, S_{t-2} = q_k, \ldots) = P(S_t = q_j \mid S_{t-1} = q_i)$

For example, consider the previous simple weather model with three states: $q_1 =$ *sunny*, $q_2 =$ *cloudy*, $q_3 =$ *raining*. In this case to specify an MC we will require a vector with three prior probabilities (see Table 5.1) and a 3×3 matrix of transition probabilities (see Table 5.2).

The transition matrix can be represented graphically with what is called a *state transition diagram* or simply a *state diagram*. This diagram is a directed graph,

Table 5.1 Prior probabilities for the weather example

Sunny	Cloudy	Raining
0.2	0.5	0.3

Table 5.2 Transition probabilities for the weather example

	Sunny	Cloudy	Raining
Sunny	0.8	0.1	0.1
Cloudy	0.2	0.6	0.2
Raining	0.3	0.3	0.4

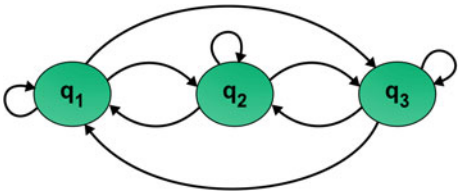


Fig. 5.2 The figure illustrates the state transition diagram for the weather example

where each node is a state and the arcs represent possible transitions between states. If an arc between state q_i and q_j does not appear in the diagram, it means that the corresponding transition probability is zero. An example of a state diagram for the weather example is depicted in Fig. 5.2.[1]

Given a Markov chain model, there are three basic questions that we can ask:

- What is the probability of a certain state sequence?
- What is the probability that the chain remains in a certain state for a period of time?
- What is the expected time that the chain will remain in a certain state?

Next, we will see how we can answer these questions and we will illustrate them using the weather example.

The probability of a sequence of states given the model is basically the product of the transition probabilities of the sequence of states:

$$P(q_i, q_j, q_k, \ldots) = a_{0i} a_{ij} a_{jk} \ldots \tag{5.1}$$

where a_{0i} is the transition to the initial state in the sequence, which could be its prior probability (π_i) or the transition from the previous state (if this is known).

For example, in the weather model, we might want to know the probability of the following sequence of states: $Q =$ *sunny, sunny, rainy, rainy, sunny, cloudy, sunny*. Assuming that *sunny* is the initial state in the MC, then:

$$\begin{aligned} P(Q) &= \pi_1 a_{11} a_{13} a_{33} a_{31} a_{12} a_{21} = (0.2)(0.8)(0.1)(0.4)(0.3)(0.1)(0.2) \\ &= 3.84 \times 10^{-5} \end{aligned}$$

The probability of staying d time steps in a certain state, q_i, is equivalent to the probability of a sequence in this state for $d-1$ time steps and then transiting to a different state. That is,

$$P(d_i) = a_{ii}^{d-1}(1 - a_{ii}) \tag{5.2}$$

[1] Do not confuse a state diagram, where a node represents each state—a specific value of a random variable—and the arcs transitions between states, with a graphical model diagram, where a node represents a random variable and the arcs represent probabilistic dependencies.

Considering the weather model, what is the probability of three cloudy days? This can be computed as follows:

$$P(d_2 = 3) = a_{22}^2(1 - a_{22}) = 0.6^2(1 - 0.6) = 0.144$$

The average duration of a state sequence in a certain state is the expected value of the number of stages in that state, that is, $E(D) = \sum_i d_i P(d_i)$. Substituting Eq. (5.2) we obtain:

$$E(d_i) = \sum_i d_i a_{ii}^{d-1}(1 - a_{ii}) \tag{5.3}$$

which can be written in a compact form as:

$$E(d_i) = 1/(1 - a_{ii}) \tag{5.4}$$

For instance, what is the expected number of days that the weather will remain cloudy? Using the previous equation:

$$E(d_2) = 1/(1 - a_{22}) = 1/(1 - 0.6) = 2.5$$

5.2.1 Parameter Estimation

Another important question is how to determine the parameters of the model, which is known as *parameter estimation*. For an MC, the parameters can be estimated simply by counting the number of times that the sequence is in a certain state, i, and the number of times there is a transition from a state i to a state j. Assume there are N sequences of observations. γ_{0i} is the number of times that the state i is the initial state in a sequence, γ_i is the number of times that we observe state i, and γ_{ij} is the number of times that we observe a transition from state i to state j. The parameters can be estimated with the following equations.

Initial probabilities:

$$\pi_i = \gamma_{0i}/N \tag{5.5}$$

Transition probabilities:

$$a_{ij} = \gamma_{ij}/\gamma_i \tag{5.6}$$

Note that for the last observed state in a sequence we do not observe the next state, so the last state for all the sequences is not considered in the counts.

Table 5.3 Calculated prior probabilities for the weather example

Sunny	Cloudy	Raining
0.25	0.5	0.25

Table 5.4 Calculated transition probabilities for the weather example

	Sunny	Cloudy	Raining
Sunny	0	0.33	0.67
Cloudy	0.285	0.43	0.285
Raining	0.18	0.18	0.64

For instance, consider that for the weather example we have the following four observation sequences ($q_1 =$ Sunny, $q_2 =$ Cloudy, $q_3 =$ Raining):

$$q_2, q_2, q_3, q_3, q_3, q_3, q_1$$
$$q_1, q_3, q_2, q_3, q_3, q_3, q_3$$
$$q_3, q_3, q_2, q_2$$
$$q_2, q_1, q_2, q_2, q_1, q_3, q_1$$

Given these four sequences, the corresponding parameters can be estimated as depicted in Tables 5.3 and 5.4.

5.2.2 Convergence

An additional interesting question is: if a sequence transits from one state to another a large number of times, M, what is the probability in the limit (as $M \rightarrow \infty$) of each state, q_i?

Given an initial probability vector, Π, and transition matrix, A, the probability of each state, $P = \{p_1, p_2, \ldots, p_n\}$ after M *iterations* is:

$$P = \pi A^M \tag{5.7}$$

What happens when $M \rightarrow \infty$? The solution is given by the Perron-Frobenius theorem, which says that when the following two conditions are satisfied:

1. Irreducibility: from every state i there is a probability $a_{ij} > 0$ of transiting to any state j.
2. Aperiodicity: the chain does not form *cycles* (a subset of states in which the chain remains once it transits to one of these state).

Then as $M \rightarrow \infty$, the chain converges to an invariant distribution P, such that $P \times A = P$, where A is the transition probability matrix. The rate of convergence is determined by the second *eigenvalue* of matrix A.

For example, consider an MC with three states and the following transition probability matrix:

$$A = \begin{matrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{matrix}$$

It can be shown that in this case the steady-state probabilities converge to $P = \{0.625, 0.3125, 0.0625\}$.

An interesting application of this convergence property of Markov chains for ranking web pages is presented in the applications section. Next we discuss hidden Markov models.

5.3 Hidden Markov Models

A Hidden Markov model (HMM) is a Markov chain where the states are not directly observable. For example, if we consider the weather example, the *weather* cannot be directly measured; in reality, the weather is estimated based on a series of sensors—temperature, pressure, wind velocity, etc. So in this, as in many other phenomena, states are not directly observable, and HMMs provide a more appropriate and powerful modeling tool. Another way of thinking about an HMM is that it is a double stochastic process: (i) a hidden stochastic process that we cannot directly observe, (ii) and a second stochastic process that produces the sequence of observations given the first process.

For instance, consider that we have two unfair or "biased" coins, M_1 and M_2. M_1 has a higher probability of *heads*, while M_2 has a higher probability of *tails*. Someone sequentially flips these two coins, however, we do not know which one. We can only observe the outcome, *heads* or *tails*:

$$H, T, T, H, T, H, H, H, T, H, T, H, T, T, T, H, T, H, H, \ldots$$

Assume the person flipping the coins selects the first coin in the sequence (prior probabilities) and the next coin to flip given the previous one (transition probabilities) with equal probability. Aside from the prior and transition probabilities for the states (as with a MC), in an HMM we need to specify the *observation* probabilities, in this case the probabilities of H or T given each coin (the state). Let us assume that M_1 has an 80 % probability for *heads* and M_2 has an 80 % probability for *tails*. Then we have specified all the required parameters for this simple example, which are summarized in Table 5.5.

The state diagram for the coins example is depicted in Fig. 5.3, with two state variables and two possible observations, which depend on the state.

Table 5.5 The prior probabilities (Π), transition probabilities (A) and observation probabilities (B) for the unfair coins example

$$\Pi = \begin{array}{|cc|}\hline M_1 & M_2 \\ \hline 0.5 & 0.5 \\ \hline \end{array} \qquad A = \begin{array}{c|cc} & M_1 & M_2 \\ \hline M_1 & 0.5 & 0.5 \\ M_2 & 0.5 & 0.5 \end{array} \qquad B = \begin{array}{c|cc} & M_1 & M_2 \\ \hline H & 0.8 & 0.2 \\ T & 0.2 & 0.8 \end{array}$$

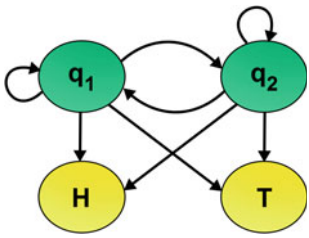


Fig. 5.3 State diagram for the HMM coins example. The two states, q_1 and q_2, and two observations, H and T are shown, with *arcs* representing the transitions and observation probabilities

Formally, a hidden Markov model is defined as

Set of states: $Q = \{q_1, q_2, \ldots, q_n\}$
Set of observations: $O = \{o_1, o_2, \ldots, o_m\}$
Vector of prior probabilities: $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, where $\pi_i = P(S_0 = q_i)$
Matrix of transition probabilities: $A = \{a_{ij}\}$, $i = [1..n]$, $j = [1..n]$, where $a_{ij} = P(S_t = q_j \mid S_{t-1} = q_i)$
Matrix of observation probabilities: $B = \{b_{ij}\}$, $i = [1..n]$, $j = [1..m]$, where $b_{ik} = P(O_t = o_k \mid S_t = q_i)$

where n is the number of states and m the number of observations; S_0 is the initial state. Compactly, an HMM is represented as $\lambda = \{A, B, \Pi\}$.

A (first order) HMM satisfies the following properties:

Markov property: $P(S_t = q_j \mid S_{t-1} = q_i, S_{t-2} = q_k, \ldots) = P(S_t = q_j \mid S_{t-1} = q_i)$
Stationary process: $P(S_{t-1} = q_j \mid S_{t-2} = q_i) = P(S_t = q_j \mid S_{t-1} = q_i)$ and $P(O_{t-1} = o_k \mid S_{t-1} = q_j) = P(O_t = o_k \mid S_t = q_i), \forall(t)$
Independence of observations: $P(O_t = o_k \mid S_t = q_i, S_{t-1} = q_j, \ldots) = P(O_t = o_k \mid S_t = q_i)$

As in the case of an MC, the Markov property implies that the probability of the current state only depends on the previous state, and it is independent of the rest of the history. The second property says that the transition and observation probabilities do not change over time, i.e., the process is stationary. The third property specifies that the observations only depend on the current state. There are extensions to the *basic* HMM that relax some of these assumptions; some of these will be discussed in the next section and in further chapters.

According to the previous properties, the graphical model of an HMM is shown in Fig. 5.4, which includes two series of random variables, the state at time t, S_t, and the observation at time t, O_t.

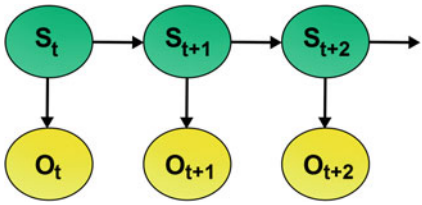


Fig. 5.4 Graphical model representing a hidden Markov model

Given an HMM representation of a certain domain, there are three basic questions that are of interest in most applications [7]:

1. *Evaluation*: given a model, estimate the probability of a sequence of observations.
2. *Optimal Sequence*: given a model and a particular observation sequence, estimate the most probable state sequence that produced the observations.
3. *Parameter learning*: given a number of sequence of observations, adjust the parameters of the model.

Algorithms for solving these questions, assuming a *standard HMM* with finite number of states and observations, are described next.

5.3.1 Evaluation

Evaluation consists in determining the probability of an observation sequence, $O = \{o_1, o_2, o_3, \ldots\}$, given a model, λ, that is, estimating $P(O \mid \lambda)$. We present two methods. First we present the direct method, a *naive* algorithm that motivates the need for a more efficient one, which is then described.

5.3.1.1 Direct Method

A sequence of observations, $O = \{o_1, o_2, o_3, \ldots o_T\}$, can be generated by different state sequences, Q_i, as the states are unknown for HMMs. Thus, to calculate the probability of an observation sequence, we can estimate it for a certain state sequence, and then add the probabilities for all the possible state sequences:

$$P(O \mid \lambda) = \sum_i P(O, Q_i \mid \lambda) \tag{5.8}$$

To obtain $P(O, Q_i \mid \lambda)$ we simply multiply the probability of the initial state, q_1, by the transition probabilities for a state sequence, $q_1, q_2, \ldots$ and the observation probabilities for an observation sequence, $o_1, o_2, \ldots$:

$$P(O, Q_i \mid \lambda) = \pi_1 b_1(o_1) a_{12} b_2(o_2) \ldots a_{(T-1)T} b_T(o_T) \tag{5.9}$$

Thus, the probability of O is given by a summation over all the possible state sequences, Q:

$$P(O \mid \lambda) = \sum_{Q} \pi_1 b_1(o_1) a_{12} b_2(o_2) \ldots a_{(T-1)T} b_T(o_T) \tag{5.10}$$

For a model with N states and an observation length of T, there are N^T possible state sequences. Each term in the summation requires $2T$ operations. As a result, the evaluation requires a number of operations of the order of $2T \times N^T$.

For example, if we consider a model with five states, $N = 5$, and an observation sequence of length $T = 100$, which are common parameters for HMM applications, the number of required operations is of the order of 10^{72}. A more efficient method is required!

5.3.1.2 Iterative Method

The basic idea of the iterative method, also known as *Forward*, is to estimate the probabilities of the states/observations per time step. That is, calculate the probability of a partial sequence of observations until time t (starting form $t = 1$), and based on this partial result, calculate it for time $t + 1$, and so on.

First, we define an auxiliary variable called *forward*:

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, S_t = q_i \mid \lambda) \tag{5.11}$$

That is, the probability of the partial sequence of observations until time t, being in state q_i at time t.

The iterative algorithm consists of three main parts: initialization, induction, and termination. In the initialization phase, the α variables for all states at the initial time are obtained. The induction phase consists in calculating $\alpha_{t+1}(i)$ in terms of $\alpha_t(i)$; this is repeated from $t = 2$ to $t = T$. Finally, $P(O \mid \lambda)$ is obtained by adding all the α_T in the termination phase. The procedure is shown in Algorithm 5.1.

Algorithm 5.1 The Forward Algorithm

Require: HMM, λ; Observation sequence, O; Number of states, N; Number of observations, T
  for $i = 1$ **to** N **do**
    $\alpha_1(i) = P(O_1, S_1 = q_i) = \pi_i b_i(O_1)$ (Initialization)
  end for
  for $t = 2$ **to** T **do**
    for $j = 1$ **to** N **do**
      $\alpha_t(j) = [\sum_i \alpha_{t-1}(i) a_{ij}] b_j(O_t)$ (Induction)
    end for
  end for
  $P(O) = \sum_i \alpha_T(i)$ (Termination)
  return $P(O)$

Let us now analyze the time complexity of the iterative method. Each iteration requires N multiplications and N additions (approx.), so for the T iterations, the number of operations is of the order of $N^2 \times T$. Thus, the time complexity is reduced from exponential in T for the direct method to linear in T and quadratic in N for the iterative method, a significant reduction in complexity; note that in most applications $T \gg N$.

Returning to the example where $N = 5$ and $T = 100$, now the number of operations is approximately 2500, which can be carried out in a few milliseconds with a standard personal computer.

The iterative procedure just described is the basis for solving the other two questions for HMMs. These are described next.

5.3.2 State Estimation

Finding the most probable sequence of states for an observation sequence, $O = \{o_1, o_2, o_3, \ldots\}$, can be interpreted in two ways: (i) obtaining the most probable state, S_t at each time step t, (ii) obtaining the most probable sequence of states, $s_0, s_1, \ldots s_T$. Notice that the concatenation of the most probable states for each time step, for $t = 1 \ldots T$, is not necessarily the same as the most probable sequence of states.[2] First, we solve the problem of finding the most probable or *optimum* state for a certain time t, and then the problem of finding the *optimum* sequence.

We first need to define some additional auxiliary variables. The *backward* variable is analogous to the forward one, but in this case we start from the end of the sequence, that is,

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T, S_t = q_i \mid \lambda) \tag{5.12}$$

That is, the probability of the partial sequence of observations from $t + 1$ to T, being in state q_i at time t. In a similar way to α, it can be calculated iteratively but now backwards:

$$\beta_t(i) = \sum_j \beta_{t+1}(j) a_{ij} b_j(o_t) \tag{5.13}$$

The β variables for T are defined as $\beta_T(j) = 1$.

Thus, we can also solve the evaluation problem of the previous section using β instead of α, starting from the end of the observation sequence and iterating backwards through time. Or we can combine both variables, and iterate forward and backward, *meeting* at some intermediate time t; that is,

$$P(O, s_t = q_i \mid \lambda) = \alpha_t(i)\beta_t(i) \tag{5.14}$$

[2] This is a particular case of the *Most Probable Explanation* or MPE problem, which will be discussed in Chap. 7.

Then:

$$P(O \mid \lambda) = \sum_i \alpha_t(i)\beta_t(i) \tag{5.15}$$

Now we define an additional variable, γ, that is, the conditional probability of being in a certain state q_i given the observation sequence:

$$\gamma_t(i) = P(s_t = q_i \mid O, \lambda) = P(s_t = q_i, O \mid \lambda)/P(O) \tag{5.16}$$

which can be written in terms of α and β as:

$$\gamma_t(i) = \alpha_t(i)\beta_t(i)/\sum_i \alpha_t(i)\beta_t(i) \tag{5.17}$$

This variable, γ, provides the answer to the first subproblem, the most probable state (MPS) at a time t; we just need to find for which state it has the maximum value. That is,

$$MPS(t) = ArgMax_i\, \gamma_t(i) \tag{5.18}$$

Let us now solve the second subproblem—the most probable state sequence Q given the observation sequence O, such that we want to maximize $P(Q \mid O, \lambda)$. By Bayes rule: $P(Q \mid O, \lambda) = P(Q, O \mid \lambda)/P(O)$. Given that $P(O)$ does not depend on Q, this is equivalent to maximizing $P(Q, O \mid \lambda)$.

The method for obtaining the optimum state sequence is known as the *Viterbi* algorithm, which in an analogous way as the forward algorithm, solves the problem iteratively. Before we see the algorithm, we need to define an additional variable, δ. This variable gives the maximum value of the probability of a subsequence of states and observations until time t, being at state q_i at time t; that is:

$$\delta_t(i) = MAX[P(s_1, s_2, \ldots s_t = q_i, o_1, o_2, \ldots, o_t \mid \lambda)] \tag{5.19}$$

which can also be obtained in an iterative way:

$$\delta_{t+1}(i) = [MAX\, \delta_t(i)a_{ij}]b_j(o_{t+1}) \tag{5.20}$$

The Viterbi algorithm requires four phases: initialization, recursion, termination, and backtracking. It requires an additional variable, $\psi_t(i)$, that stores for each state i at each time step t the previous state that gave the maximum probability. This is used to reconstruct the sequence by backtracking after the termination phase. The complete procedure is depicted in Algorithm 5.2.

With the Viterbi algorithm we can obtain the most probable sequence of states, even if these are hidden for HMMs.

Algorithm 5.2 The Viterbi Algorithm

Require: HMM, λ; Observation sequence, O; Number of states, N; Number of observations, T
  for $i = 1$ **to** N **do**
    (Initialization)
    $\delta_1(i) = \pi_i b_i(O_1)$
    $\psi_1(i) = 0$
  end for
  for $t = 2$ **to** T **do**
    for $j = 1$ **to** N **do**
      (Recursion)
      $\delta_t(j) = MAX_i[\delta_{t-1}(i)a_{ij}]b_j(O_t)$
      $\psi_t(j) = ARGMAX_i[\delta_{t-1}(i)a_{ij}]$
    end for
  end for
  (Termination)
  $P^* = MAX_i[\delta_T(i)]$
  $q_T^* = ARGMAX_i[\delta_T(i)]$
  for $t = T$ **to** 2 **do**
    (Backtracking)
    $q_{t-1}^* = \psi_t(q_t^*)$
  end for

5.3.3 Learning

Finally, we will see how we can learn an HMM from data via the *Baum-Welch* algorithm. First, we should note that this method assumes that the *structure* of the model is known: the number of states and observations is previously defined; therefore, it only estimates the parameters. Usually the observations are given by the application domain, but the number of states, which are hidden, are not so easy to define. Sometimes the number of hidden states can be defined based on domain knowledge; at other times it is done experimentally through trial and error: test the performance of the model with different numbers of states (2, 3, ...) and select the number that gives the best results. It should be noted that there is a tradeoff in this selection, as a larger number of states tend to produce better results but also imply additional computational complexity.

The Baum-Welch algorithm determines the parameters of an HMM, $\lambda = A, B, \Pi$, given a number of observation sequences, $\mathbf{O} = O_1, O_2, \ldots O_K$. For this it maximizes the probability of the model given the observations: $P(\mathbf{O} \mid \lambda)$. For an HMM with N states and M observations, we need to estimate $N + N^2 + N \times M$ parameters, for Π, A and B, respectively.

We need to define one more auxiliary variable, ξ, the probability of a transition from a state i at time t to a state j at time $t + 1$ given an observation sequence O:

$$\xi_t(i, j) = P(s_t = q_i, s_{t+1} = q_j \mid O, \lambda) = P(s_t = q_i, s_{t+1} = q_j, O \mid \lambda)/P(O) \tag{5.21}$$

In terms of α and β:

$$\xi_t(i, j) = \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)/P(O) \tag{5.22}$$

Writing $P(O)$ also in terms of α and β:

$$\xi_t(i, j) = \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)/\sum_i \sum_j \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j) \tag{5.23}$$

γ can also be written in terms of ξ: $\gamma_t(i) = \sum_j \xi_t(i, j)$

By adding $\gamma_t(i)$ for all time steps, $\sum_t \gamma_t(i)$, we obtain an estimate of the number of times that the chain is in state i, and by accumulating $\xi_t(i, j)$ over t, $\sum_t \xi_t(i, j)$, we estimate the number of transitions from state i to state j.

Based on the previous definitions, the Baum-Welch procedure for parameter estimation for HMMs is summarized in Algorithm 5.3.

Algorithm 5.3 The Baum-Welch Algorithm

1. Estimate the prior probabilities—the number of times being in state i at time t.

$$\pi_i = \gamma_1(i)$$

2. Estimate the transition probabilities—the number of transitions from state i to j between the number of times in state i.

$$a_{ij} = \sum_t \xi_t(i, j)/\sum_t \gamma_t(i)$$

3. Estimate the observation probabilities—the number of times being in state j and observing k between the number of times in state j.

$$b_{jk} = \sum_{t, O=k} \gamma_t(j)/\sum_t \gamma_t(j)$$

Notice that the calculation of γ and ξ variables is done in terms of α and β, which require the parameters of the HMM, Π, A, B. So we have encountered a "chicken and egg" problem—we need the model parameters for the Baum-Welch algorithm, which estimates the model parameters! The solution to this problem is based on the EM (for expectation-maximization) principle.

The idea is to start with some initial parameters for the model (E-step), $\lambda = \{A, B, \Pi\}$, which can be initialized randomly or based on some domain knowledge. Then, via the Baum-Welch algorithm, these parameters are re-estimated (M-step). This cycle is repeated until convergence; that is, until the difference between the parameters for the model from one step to the next is below a certain threshold.

The EM algorithm provides what is known as a *maximum-likelihood* estimator, which does not guarantee an optimal solution—it depends on the initial conditions. However, this estimator tends to work well in practice.[3]

5.3.4 Extensions

Several extensions to standard HMMs have been proposed to deal with particular issues in several applications [2]. Next we briefly describe some of these extensions, whose graphical models are depicted in Fig. 5.5.

Parametric HMMs (PHMMs) represent domains that involve variations in the models. In PHMMs, observation variables are conditioned to the state variable and one or more parameters that account for such variations, see Fig. 5.5b. Parameter values are known and constant on training. On testing, values that maximize

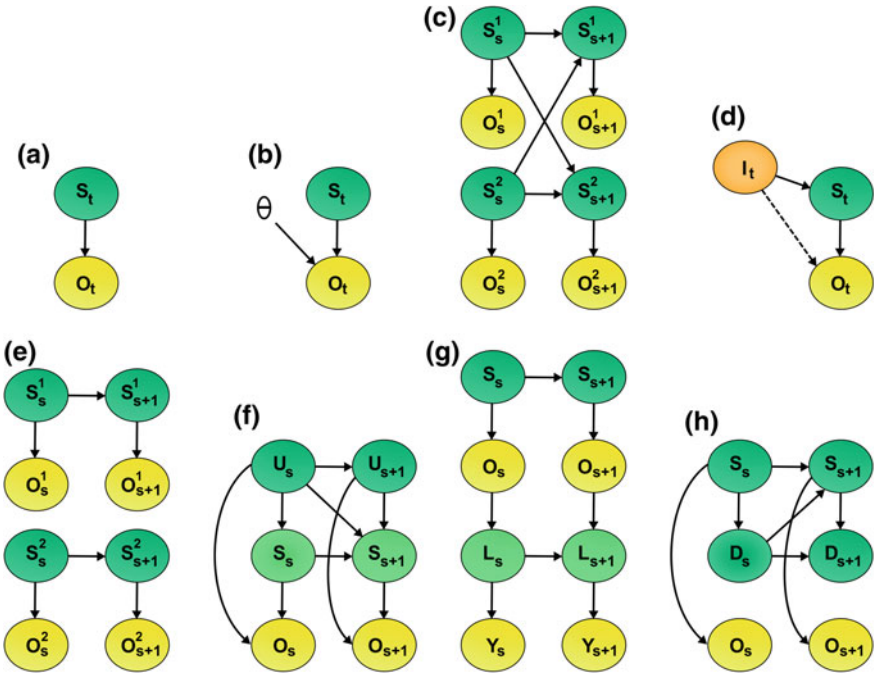


Fig. 5.5 Graphical model representation for the basic HMM and several extensions. **a** Basic model. **b** Parametric HMMs. **c** Coupled HMMs. **d** Input-Output HMMs. **e** Parallel HMMs. **f** Hierarchical HMMs. **g** Mixed-state dynamic Bayesian networks. **h** Hidden semi-Markov models

[3] If we have some domain knowledge this could provide a *good* initialization for the parameters; otherwise, we can set them to uniform probabilities.

the likelihood of the PHMM are recovered via a tailored expectation-maximization algorithm.

Coupled HMMs (CHMMs) join HMMs by introducing conditional dependencies between state variables—see Fig. 5.5c. These models are suitable to represent influences between subprocesses that occur in parallel.

Input–Output HMMs (IOHMMs) consider an extra *input* parameter that affects the states of the Markov chain, and optionally, the observation variables. These types of models are illustrated in Fig. 5.5d. The input variable corresponds to the observations. The output signal of IOHMMs is the class of model (e.g., a phoneme in speech recognition or a particular movement in gesture recognition) that is being executed. A single IOHMM can describe a complete set of classes.

Parallel HMMs (PaHMMs) require fewer HMMs than CHMMs for composite processes by assuming mutual independence between HMMs, see Fig. 5.5e. The idea is to construct independent HMMs for two (or more) independent parallel processes (for example, the possible motions of each hand in gesture recognition), and combine them by multiplying their individual likelihoods. PaHMMs with the most probable joint likelihood define the desired class.

Hierarchical hidden Markov models (HHMMs) arrange HMMs into layers at different levels of abstraction; Fig. 5.5f. In a two-layer HHMM, the lower layer is a set of HMMs that represents submodel sequences. The upper layer is a Markov chain that governs the dynamics of these submodels. Layering allows us to reuse the basic HMMs simply by changing upper layers.

Mixed-state dynamic Bayesian networks (MSDBNs)[4] combine discrete and continuous state spaces into a two-layer structure. MSDBNs are composed of an HMM in the upper layer and a linear dynamic system (LDS) in the lower layer. The LDS is used to model transitions between real-valued states. The output values of the HMM drive the linear system. The graphical representation of MSDBNs is depicted in Fig. 5.5g. In MSDBNs, HMMs can describe discrete high-level concepts, such as a grammar, while the LDS describes the input signals in a continuous-state space.

Hidden semi-Markov models (HSMMs) exploit temporal knowledge belonging to the process by defining an explicit duration on each state, see Fig. 5.5h. HSMMs are suitable to avoid an exponential decay of the state probabilities when modeling large observation sequences.

5.4 Applications

In this section we illustrate the application of Markov chains and HMMs in two domains. First we describe the use of Markov chains for ordering web pages with the PageRank algorithm. Then we present an application of HMMs in gesture recognition.

[4]Hidden Markov models, including these extensions, are particular types of dynamic Bayesian networks, a more general model that is described in Chap. 9.

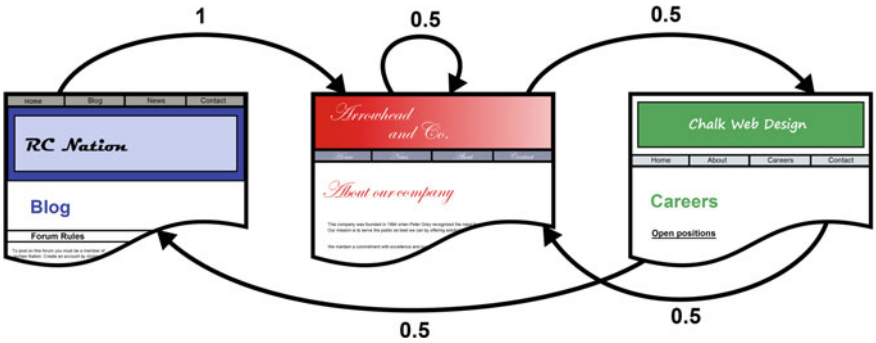


Fig. 5.6 A small example of a WWW with 3 pages

5.4.1 PageRank

We can think of the World Wide Web (WWW) as a very large Markov chain, such that each web page is a state and the hyperlinks between web pages correspond to state transitions. Assume that there are N web pages. A particular web page, w_i, has m outgoing hyperlinks. If someone is at web page w_i, she can select any of the hyperlinks in this page to go to another page. A reasonable assumption is that each outgoing link can be selected with equal probability; thus, the transition probability from w_i to any of the web pages with which it has hyperlinks, w_j, is $A_{ij} = 1/m$. For the other web pages for which it has no outgoing links, the transition probability is zero. In this way, according to the structure of the WWW, we can obtain the transition probability matrix, A, for the corresponding Markov chain. The state diagram of a small example with three web pages is shown in Fig. 5.6.

Given the transition probability matrix of the WWW, we can obtain the convergence probabilities for each state (web page) according to the Perron-Frobenius theorem (see Sect. 5.2). The convergence probability of a certain web page can be thought to be equivalent to the probability of a person, who is navigating the WWW, visiting this web page. Intuitively, web pages that have more ingoing links, from web pages with more ingoing links, will have a higher probability of being accessed.

Based on the previous ideas, L. Page et al. developed the *PageRank* algorithm, which is the basis of how web pages are ordered when we make a search in *Google* [6]. The web pages retrieved by the search algorithm are presented to the user according to their convergence probabilities. The idea is that more relevant (important) web pages will tend to have higher convergence probability.

5.4.2 Gesture Recognition

Gestures are essential for human–human communication, so they are also important for human–computer interaction. For example, we can use gestures to command a

Fig. 5.7 A video sequence depicting a person performing a *stop* gesture with his right hand. A few key frames are shown

service robot. We will focus on dynamic hand gestures, those that consist in movements by the hand/arm of a person. For example, Fig. 5.7, depicts some frames of a person executing a *stop* gesture.

For recognizing gestures, a powerful option is a hidden Markov model [1, 9]. HMMs are appropriate for modeling sequential processes, and are robust to the temporal variations common in the execution of dynamic gestures. Before we can apply HMMs to model and recognize gestures, the images in the video sequence need to be processed and a set of features extracted from them; these will constitute the observations for the HMM.

Image processing consists in detecting the person in the image, detecting their hand, and then tracking the hand in the sequence of images. From the image sequence, the position of the hand (*XYZ*) is extracted from each image. Additionally, some other regions of the body could be detected, such as the head and torso, which are used to obtain posture features as described below.

Alternatives to describe gestures can be divided in: (a) motion features, (b) posture features, and (c) posture-motion features. Motion features describe the motion of the person's hand in the Cartesian space *XYZ*. Posture features represent the position of the hand with respect to other parts of the person's body, such as the head or torso. These motion and posture features are usually codified in a finite number of code words that provide the observations for the HMMs. For example, if we consider three values to describe each motion coordinate and two binary posture features (e.g., hand above the head, etc.), there will be $m = 3 \times 3 \times 3 \times 2 \times 2 = 108$ possible observations. These are obtained for each frame (or each n frames) in the video sequence of the gesture.

To recognize N different gestures, we need to train N HMMs, one for each gesture. The first parameter that needs to be defined is the number of hidden states for each model. As mentioned before, this can be set based on domain knowledge, or obtained via cross-validation by experimentally evaluating different amounts of states. In the case of dynamic gestures, we can think of the states as representing the different stages in the movement of the hand. For instance, the *stop* gesture can be thought of as having three phases: moving the arm up and forward, extending the hand, and moving the arm down; this implies three hidden states. Experimentally, it has been found that using three–five states usually produces good results.

Once the number of states for each gesture model is defined (these could be different for each model), the parameters are obtained using the Baum-Welch procedure.

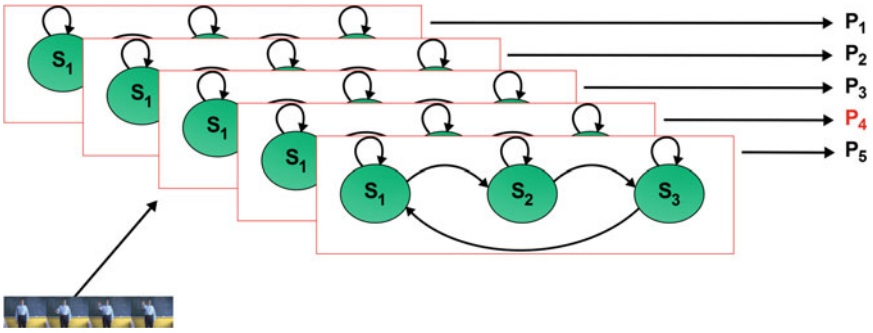


Fig. 5.8 Gesture recognition using HMMs. The features extracted from the video sequence are fed as observations to each HMM, one per gesture class, and the probability of each model is obtained. The model with highest probability defines the recognized gesture

For this, a set of M training sequences for each gesture class are required, and the more samples the better. Thus, N HMMs are obtained, one for each gesture type.

For recognition, the features are extracted from the video sequence. These are the observations, O, for the N HMMs models λ_i, one for each gesture type. The probability of each model given the observation sequence, $P(O \mid \lambda_i)$, are obtained using the Forward algorithm. The model with the highest probability, λ_k^*, is selected as the recognized gesture. Figure 5.8 illustrates the recognition process, considering five classes of gestures.

5.5 Additional Reading

A general introduction to Markov chains is provided in [4]. Rabiner [7] presents an excellent introduction to HMMs and their application to speech recognition; [8] provides a more general overview of speech recognition. A review of several extensions of HMMs in the context of gesture recognition is given in [2]. Reference [5] analyzes search engines and the PageRank algorithm. The application of HMMs to gesture recognition is described by [1, 9]. Open software for HMMs is available in [3].

5.6 Exercises

1. For the Markov chain weather model: (i) Determine the probability of the state sequence: *cloudy*, *raining*, *sunny*, *sunny*, *sunny*, *cloudy*, *raining*, *raining*. (ii) What is the probability of four continuous rainy days? (iii) What is the expected number of days that it will continue raining?
2. For the small web page example of Fig. 5.6, determine: (i) if the convergence conditions are satisfied, and if so, (ii) the order in which the three web pages will be presented to the user.

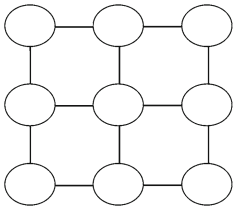
3. Consider the unfair coin example. Given the parameters in Table 5.5, obtain the probability of the following observation sequence: *HHTT* using (i) the direct method, (ii) the forward algorithm.
4. For the previous problem, what is the number of operations for each of the two methods?
5. For problem 3, obtain the most probable state sequence using Viterbi's algorithm.
6. What are the three basic assumptions in standard HMMs? Express them mathematically.
7. Assume there are two HMMs that represent two phonemes: $ph1$ and $ph2$. Each model has two states and two observations with the following parameters:
   Model 1: $\Pi = [0.5, 0.5]$, $A = [0.5, 0.5 \mid 0.5, 0.5]$, $B = [0.8, 0.2 \mid 0.2, 0.8]$
   Model 2: $\Pi = [0.5, 0.5]$, $A = [0.5, 0.5 \mid 0.5, 0.5]$, $B = [0.2, 0.8 \mid 0.8, 0.2]$
   Given the following observation sequence: "o1,o1,o2,o2", which is the most probable phoneme?
8. We want to develop a head gesture recognition system and we have a vision system that can detect the following movements of the head: (1) up, (2) down, (3) left, (4) right, (5) stable. The vision system provides a number for each type of movement (1–5) each second, which is the input (observation) to the gesture recognition system. The gesture recognition system should recognize four classes of head gestures: (a) affirmative action, (b) negative action, (c) turn right, (d) turn left. (i) Specify a model that is adequate for this recognition problem, including the structure and required parameters. (ii) Indicate which algorithms are appropriate for learning the parameters of the model, and for recognition.
9. *** Develop a program to solve the previous problem.
10. *** An open problem for HMMs is establishing the *optimum* number of states for each model. Develop a search strategy for determining the optimum number of states for each model for the head gesture recognition system that maximizes the recognition rate. Consider that the data set (examples for each class of gesture) is divided into three sets: (a) training—to estimate the parameters of the model, (b) validation—to compare the different models, (c) test—for testing the final models.

References

1. Aviles, H., Sucar, L.E., Mendoza C.E.: Visual recognition of similar gestures. In: 18th International Conference on Pattern Recognition, pp. 1100–1103 (2006)
2. Aviles, H., Sucar, L.E., Mendoza, C.E., Pineda, L.A.: A Comparison of dynamic naive Bayesian classifiers and hidden Markov models for gesture recognition. J. Appl. Res. Technol. **9**(1), 81–102 (2011)
3. Kanungo, T.: UMDHMM: Hidden Markov Model Toolkit. In: Kornai, A. (ed.) Extended Finite State Models of Language. Cambridge University Press (1999). http://www.kanungo.com/software/software.html
4. Kemeny, J.K., Snell, L.: Finite Markov Chains. Van Nostrand, Princeton (1965)
5. Langville, N., Carl, D., Meyer, C.D.: Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press, Princeton (2012)

6. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Libraries Working Paper (1998)
7. Rabiner, L.E.: A tutorial on hidden Markov models and selected applications in speech recognition. In: Waibel, A., Lee, K. (eds.) Readings in Speech Recognition, pp. 267–296. Morgan Kaufmann, San Francisco (1990)
8. Rabiner, L., Juang, B.H.: Fundamentals on Speech Recognition. Prentice-Hall Signal Processing Series, New Jersey (1993)
9. Wilson, A., Bobick, A.: Using hidden Markov models to model and recognize gesture under variation. Int. J. Pattern Recognit. Artif. Intell., Spec. Issue Hidden Markov Models Comput. Vis. **15**(1), 123–160 (2000)

Chapter 6
Markov Random Fields



6.1 Introduction

Certain processes, such as a ferromagnetic material under a magnetic field, or an image, can be modeled as a series of *states* in a chain or a regular grid. Each state can take different values and is influenced probabilistically by the states of its neighbors. These models are known as *Markov random fields* (MRFs).

MRFs originated from modelling ferromagnetic materials in what is known as the *Ising* model [2]. In an Ising model, there are a series of random variables in a line; each random variable represents a dipole that could be in two possible states, *up* $(+)$ or *down* $(-)$. The state of each dipole depends on an external field and the state of its neighbor dipoles in the line. A simple example with four variables is illustrated in Fig. 6.1. A *configuration* of an MRF is a particular assignment of values to each variable in the model; in the case of the model in Fig. 6.1, there are 16 possible configurations: $++++, +++-, ++-+, \ldots, ----$.

An MRF is represented as an undirected graphical model, such as in the previous example. An important property of an MRF is that the state of a variable is independent of all other variables in the model given its neighbors in the graph. For instance, for the example in Fig. 6.1, q_1 is independent of q_3 and q_4 given q_2. That is $P(q_1 \mid q_2, q_3, q_4) = P(q_1 \mid q_2)$.

The central problem in an MRF is to find the configuration of maximum probability. Usually, the probability of a configuration depends on the combination of an *external* influence (e.g., a magnetic field in the Ising model) and the *internal* influence of its neighbors. More generally, the posterior probability of a configuration depends on the prior knowledge or context, and the data or likelihood.

Using a physical analogy, an MRF can be thought of as a series of rings in poles, where each ring represents a random variable, and the height of a ring in a pole corresponds to its state. The rings are arranged in a line, see Fig. 6.2. Each ring is attached to its neighbors with a spring, this corresponds to the internal influences, and it is also attached to the base of its pole with another spring, representing the external influence.The relation between the springs' constants defines the relative weight between

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_6

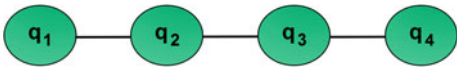


Fig. 6.1 An example of an Ising (MRF) model with four variables

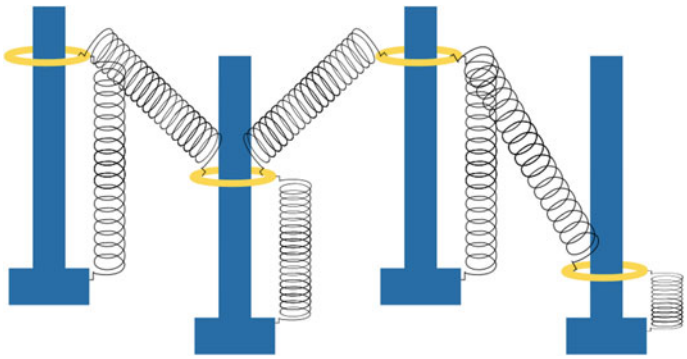


Fig. 6.2 Physical analogy of an MRF. The rings will tend to the configuration of minimum energy according to the springs that attach them to the base (external influence) and its neighbors (internal influence)

the internal and external influences. If the rings are left loose, they will stabilize to a configuration of minimum energy, which corresponds to the configuration with maximum probability in an MRF.

Markov random fields, also known as *Markov networks*, are formally defined in the next section.

6.2 Markov Networks

A *random field* (RF) is a collection of S random variables, $\mathbf{F} = F_1, \ldots F_s$, indexed by sites. Random variables can be discrete or continuous. In a discrete RF, a random variable can take a value f_i from a set of m possible values or labels $L = \{l_1, l_2, \ldots l_m\}$. In a continuous RF, a random variable can take values from the real numbers, R, or from an interval of them.

A *Markov random field* or Markov network (MN) is a random field that satisfies the locality property: a variable F_i is independent of all other variables in the field given its neighbors, $Nei(F_i)$. That is,

$$P(F_i \mid \mathbf{F_c}) = P(F_i \mid Nei(F_i)) \tag{6.1}$$

where $\mathbf{F_c}$ is the set of all random variables in the field except F_i.

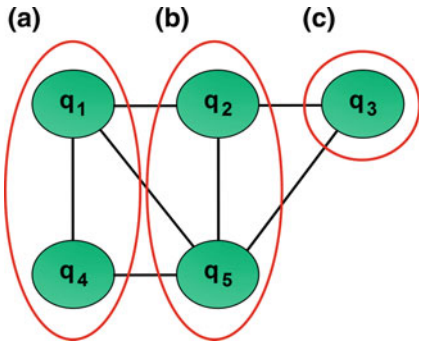


Fig. 6.3 Example of a Markov network, in which q_1, q_5 (**a**) are independent of q_3 (**c**) given q_2, q_5 (**b**)

Graphically, a Markov network (MN) is an undirected graphical model which consists of a set of random variables, **V**, and a set of undirected edges, **E**. These form an undirected graph that represents the independency relations between the random variables according to the following criteria. A subset of variables **A** is independent of the subset of variables **C** given **B**, if the variables in **B** *separate* **A** and **C** in the graph. That is, if the nodes in **B** are removed from the graph, then there are no trajectories between **A** and **C**.

Figure 6.3 depicts an example of a Markov network with five variables, $q_1, \ldots, q_5$. For instance, in this example, q_1, q_4 (**A**) are independent of q_3 (**C**) given q_2, q_5 (**B**).

The joint probability of an MN can be expressed as the product of local functions on subsets of variables in the model. These subsets should include, at least, all the *cliques* in the network. For the MN of Fig. 6.3, the joint probability distribution can be expressed as:

$$P(q_1, q_2, q_3, q_4, q_5) = (1/k)P(q_1, q_4, q_5)P(q_1, q_2, q_5)P(q_2, q_3, q_5) \tag{6.2}$$

where k is a normalizing constant. For practical convenience, other subsets of variables can also be considered for the joint probability calculation. If we also include subsets of size two, then the joint distribution for the previous example can be written as:

$$P(q_1, q_2, q_3, q_4, q_5) = (1/k)P(q_1, q_4, q_5)P(q_1, q_2, q_5)P(q_2, q_3, q_5) \tag{6.3}$$
$$P(q_1, q_2)P(q_1, q_4),\ P(q_1, q_5)P(q_2, q_3)P(q_2, q_5)P(q_3, q_5)P(q_4, q_5)$$

Formally, a Markov network is a set of random variables, $\mathbf{X} = X_1, X_2, \ldots, X_n$ that are indexed by V, such that $G = (V, E)$ is an undirected graph, which satisfies the Markov property: a variable X_i is independent of all other variables given its neighbors, $Nei(X_i)$:

$$P(X_i \mid X_1, \ldots X_{i-1}, X_{i+1}, \ldots, X_n) = P(X_i \mid Nei(X_i)) \tag{6.4}$$

The neighbors of a variable are all the variables that are directly connected to it in the graph. Under certain conditions (if the probability distribution is strictly positive), the joint probability distribution of an MRF can be factorized over the cliques of the graph:

$$P(\mathbf{X}) = (1/k) \prod_{C \in Cliques(G)} \phi_C(X_C) \tag{6.5}$$

where k is a normalizing constant and ϕ_C is a local function over the variables in the corresponding clique C.

An MRF can be categorized as *regular* or *irregular*. When the random variables are in a lattice it is considered regular; for instance, they could represent the pixels in an image, if not, they are irregular. Next we will focus on regular Markov random fields.

6.2.1 Regular Markov Random Fields

A neighboring system for a regular MRF $\mathbf{F}$ is defined as:

$$\mathbf{V} = \{Nei(F_i) \mid \forall_i \in \mathbf{F_i}\} \tag{6.6}$$

$\mathbf{V}$ satisfies the following properties:

1. A site in the field is not a neighbor to itself.
2. The neighborhood relations are symmetric, that is, if $F_j \in Nei(F_i)$ then $F_i \in Nei(F_j)$.

Typically, an MRF is arranged as a regular grid. An example of a 2D grid is depicted in Fig. 6.4. For a regular grid, a neighborhood of order i is defined as:

$$Nei_i(F_i) = \{F_j \in \mathbf{F} \mid dist(F_i, F_j) \leq r\} \tag{6.7}$$

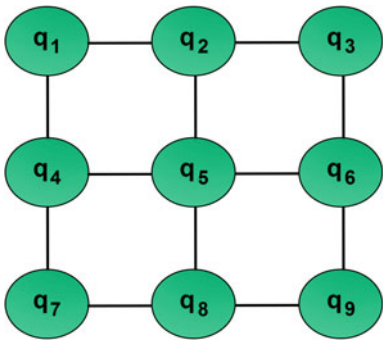


Fig. 6.4 A regular 2D MRF with a first-order neighborhood

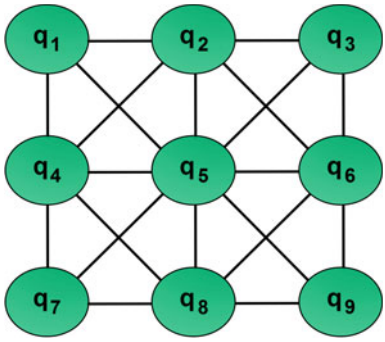


Fig. 6.5 A regular 2D MRF with a second-order neighborhood

where $dist(x, y)$ is the Euclidean distance between x and y, considering a unit distance as the vertical and horizontal distance between sites in the grid. The radius, r, is defined for each order. For example, $r = 1$ for order one, each interior site has 4 neighbors; $r = \sqrt{2}$ for order two, each interior site has 8 neighbors; $r = 2$ for order three, each interior site has 12 neighbors; and so on. Figure 6.4 shows and example of a neighborhood of order one, and Fig. 6.5 of a neighborhood of order two.

Once the structure of the MRF is specified based on the neighborhood order, its parameters must be defined. The parameters of a regular MRF are specified by a set of local functions. These functions correspond to joint probability distributions of subsets of completely connected variables in the graph. It is sufficient to include all the cliques in the graph, but other completely connected subsets can also be included. For instance, in the case of a first-order MRF, there are subsets of 2 variables; in the case of a second-order MRF, there are subsets of 2, 3, and 4 variables. In general, the joint probability distribution for the whole field can be expressed as the product of the local functions for different subsets of variables:

$$P(\mathbf{F}) = (1/k) \prod_i f(X_i) \tag{6.8}$$

where $f(X_i)$ are the local functions for subsets of variables X_i and k is a normalizing constant. We can think of these local functions as *constraints* that will favor certain configurations. For example, in the case of the Ising model, if we consider two neighboring variables, X, Y, the local function will favor (higher probabilities) configurations in which $X = Y$ and will disfavor (lower probabilities) configurations in which $X \neq Y$. These local functions can be defined subjectively depending on the application domain, or they can be learned from data.

6.3 Gibbs Random Fields

The joint probability of an MRF can be expressed in a more convenient way given its equivalence with a Gibbs Random Field (GRM), according to the Hammersley–Clifford theorem [4]. Given this equivalence, we can rewrite Eq. (6.8) as:

$$P(\mathbf{F}) = (1/z)\exp(-\mathbf{U}) \tag{6.9}$$

where $\mathbf{U}$ is known as the *energy*, given its analogy with physical energy. So maximizing $P(\mathbf{F})$ is equivalent to minimizing $\mathbf{U}$. The energy function can also be written in terms of local functions, but as this is an exponent, it is the sum of these functions (instead of a product):

$$\mathbf{U_F} = \sum_i U_i(X_i) \tag{6.10}$$

Considering a regular MRF of order n, the energy function can be expressed in terms of functions of subsets of completely connected variables of different sizes, $1, 2, 3, \ldots$:

$$\mathbf{U_F} = \sum_i U_1(F_i) + \sum_{i,j} U_2(F_i, F_j) + \sum_{i,j,k} U_3(F_i, F_j, f_k) + \cdots \tag{6.11}$$

where U_i are the local energy functions, known as *potentials*, for subsets of size i. Note that potentials are the inverse of probabilities, so low potentials are equivalent to high probabilities.

Given the Gibbs equivalence, the problem of finding the configuration of maximum probability for an MRF is transformed to finding the configuration of minimum energy.

In summary, to specify an MRF we must define:

- A set of random variables, $\mathbf{F}$, and their possible values, L.
- The dependency structure, or in the case of a regular MRF a neighborhood scheme.
- The potentials for each subset of completely connected nodes (at least the cliques).

6.4 Inference

As mentioned before, the more common application of MRFs consists in finding the most probable configuration; that is, the value for each variable that maximizes the joint probability. Given the Gibbs equivalence, this is the same as minimizing the energy function, expressed as a sum of local functions.

The set of all possible configurations of an MRF is usually very large, as it increases exponentially with the number of variables in $\mathbf{F}$. For the discrete case with m possible labels, the number of possible configurations is m^N, where N is the

number of variables in the field. If we consider an MRF representing a binary image of 100×100 pixels (a small image), then the number of configurations is $2^{10,000}$! Thus, it is impossible to calculate the energy (potential) for every configuration, except in the case of very small fields.

Finding the most probable configuration is usually posed as a stochastic search problem. Starting from an initial, random assignment of each variable in the MRF, this configuration is *improved* via local operations, until a configuration of minimum energy is obtained. In general, the minimum is a local minimum in the energy function; it is difficult to guarantee a global optimum.

A general stochastic search procedure for finding a configuration of minimum energy is outlined in Algorithm 6.1. After initializing all the variables with a random value, each variable is changed to an alternative value and its new energy is estimated. If the new energy is lower than the previous one, the value is changed; otherwise, the value may also change with a certain probability—this is done to avoid local minima. This process is repeated for a number of iterations (or until convergence).

Algorithm 6.1 Stochastic Search Algorithm

Require: MRF, **F**; Energy function, U_F; Number of iterations, N; Number of variables, S; Probability threshold, T; Convergence threshold, ε
for $i = 1$ **to** S **do**
  $F(i) = l_k$ (Initialization)
end for
for $i = 1$ **to** N **do**
  for $j = 1$ **to** S **do**
    $t = l_{k+1}$ (An alternative value for variable F(i))
    if $U(t) < U(F(i))$ **then**
      $F(i) = t$ (Change value of F(i) if the energy is lower)
    else
      if $random(U(t) - U(F(i))) < T$ **then**
        $F(i) = t$ (With certain probability change F(i) if the energy is higher)
      end if
    end if
  end for
end for
return **F*** (Return final configuration)

There are several variants of this general algorithm according to variations on different aspects. One is the way in which the *optimal* configuration is defined, for which there are two main alternatives: MAP and MPM. In the *Maximum A posteriori Probability* or MAP, the optimum configuration is taken as the configuration at the end of the iterative process. In the case of *Maximum Posterior Marginals* or MPM, the most frequent value for each variable in all the iterations is taken as the optimum configuration.

Regarding the optimization process, there are three main variations:

Iterative Conditional Modes (ICM): it always selects the configuration of minimum energy.

Metropolis: with a fixed probability, P, it selects a configuration with a higher energy.

Simulated annealing (SA): with a variable probability, $P(T)$, it selects a configuration with higher energy; where T is a parameter known as *temperature*. The probability of selecting a value with higher energy is determined based on the following expression: $P(T) = e^{-\delta U/T}$; where δU is the energy difference. The algorithm starts with a *high* value for T and this is reduced with each iteration. This makes the probability of going to higher energy states high initially, and it subsequently decreases tending to zero at the end of the process.

6.5 Parameter Estimation

The definition of a Markov random field includes several aspects:

- The structure of the model—in the case of a regular MRF the neighborhood system.
- The form of the local probability distribution functions—for each complete set in the graph.
- The parameters of the local functions.

In some applications, the previous aspects can be defined subjectively, however, this is not always easy. Different choices for the structure, distribution, and parameters can have a significant impact on the results of applying the model to specific problems. Thus, it is desirable to learn the model from data, which can have several levels of complexity. The simplest case, which is nontrivial, is when we know the structure and functional form, and we only need to estimate the parameters given a *clean* realization (without noise) of the MRF, f. It becomes more complex if the data is noisy, and even more difficult if we want to learn the functional form of the probability distribution and the order of the neighborhood system. Next we will cover the basic case, learning the parameters of an MRF from data.

6.5.1 Parameter Estimation with Labeled Data

The set of parameters, θ, of an MRF, F, is estimated from data, f, assuming no noise. Given f, the maximum likelihood (ML) estimator maximizes the probability of the data given the parameters, $P(f \mid \theta)$; thus the optimum parameters are:

$$\theta^* = ArgMax_\theta P(f \mid \theta) \tag{6.12}$$

When the prior distribution of the parameters, $P(\theta)$, is known, we can apply a Bayesian approach and maximize the posterior density obtaining the MAP estimator:

$$\theta^* = ArgMax_\theta P(\theta \mid f) \tag{6.13}$$

where:

$$P(\theta \mid f) \sim P(\theta) P(f \mid \theta) \tag{6.14}$$

The main difficulty in the ML estimation for an MRF is that it requires the evaluation of the normalizing partition function Z, in the Gibbs distribution, since it involves summing over all possible configurations. Remember that the likelihood function is given by:

$$P(f \mid \theta) = (1/Z) \exp(-U(f \mid \theta)) \tag{6.15}$$

where the partition function is:

$$Z = \sum_{f \in F} \exp(-U(f \mid \theta)) \tag{6.16}$$

Thus, the computation of Z is intractable even for MRFs of moderate size. So approximations are used for solving this problem efficiently.

One possible approximation is based on the conditional probabilities of each variable in the field, f_i, given its neighbors, N_i: $P(f_i \mid f_{N_i})$, and assuming that these are independent, we obtain what is known as the *pseudo-likelihood* (PL) [1]. Then the energy function can be written as:

$$U(f) = \sum_i U_i(f_i, f_{N_i}) \tag{6.17}$$

Assuming a first-order regular MRF, only single and pairs of nodes are considered, so:

$$U_i(f_i, N_i) = V_1(f_i) + \sum_j V_2(f_i, f_j) \tag{6.18}$$

where V_1 and V_2 are the single and pair potentials, respectively, and f_j are the neighbors of f_i.

The pseudo-likelihood (PL) is defined as the simple product of the conditional likelihoods:

$$PL(f) = \prod_i P(f_i \mid f_{N_i}) = \prod_i \frac{\exp -U_i(f_i, f_{N_i})}{\sum_{f_i} \exp -U_i(f_i, f_{N_i})} \tag{6.19}$$

Given that f_i and f_{N_i} are not independent, the PL is not the *true* likelihood; however, it has been proven that in the large lattice limit it converges to the truth with probability one [3].

Using the PL approximation, and given a particular structure and form of the local functions, we can estimate the parameters of an MRF model based on data.

Assuming a discrete MRF and given several realizations (examples), the parameters can be estimated using histogram techniques. Assume there are N distinct sets of instances of size k in the dataset, and that a particular configuration (f_i, f_{N_i}) occurs H times, then an estimate of the probability of this configuration is $P(f_i, f_{N_i}) = H/N$.

6.6 Conditional Random Fields

A limitation of MRFs (and HMMs) is that it is usually assumed that the observations are independent given each state variable. For example, in a hidden Markov model, an observation O_t is conditionally independent of all other observations and states given S_t. In traditional MRFs, it is also assumed that each observation only depends on a single variable, and it is conditionally independent of the other variables in the field (see Sect. 6.7). There are applications in which these independence assumptions are not appropriate, for example labeling the words in a sentence in natural language, in which there could be long-range dependencies between observations (words).

HMMs and *traditional* MRFs are *generative* models, which represent the joint probability distribution as the product of local functions based on the independence assumptions. If these conditional independence assumptions are removed, the models become intractable. One alternative that does not require these assumptions are *conditional random fields* (CRF) [10, 11].

A conditional random field is an undirected graphical model globally conditioned on $\mathbf{X}$, the random variable representing observations [8]. Conditional models are used to label an observation sequence $\mathbf{X}$ by selecting the label sequence $\mathbf{Y}$ that maximizes the conditional probability $P(\mathbf{Y}|\mathbf{X})$. The conditional nature of such models means that no effort is wasted on modeling the observations, and one is free from having to make unnecessary independence assumptions. The simplest structure of a CRF is that in which the nodes corresponding to elements of Y form a simple first-order chain, as illustrated in Fig. 6.6.

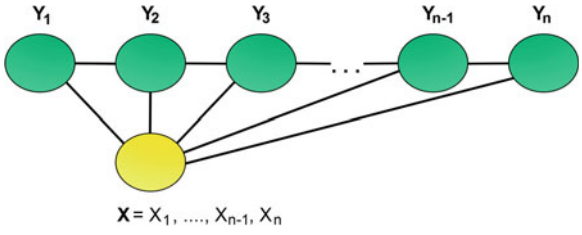


Fig. 6.6 Graphical representation of a chain-structured conditional random field

In an analogous way to MRFs, the joint distribution is factorized on a set of potential functions, where each potential function is defined over a set of random variables whose corresponding vertices form a maximal clique of G, the graphical structure of the CRF. In the case of a chain-structured CRF each potential function will be specified on pairs of adjacent label variables, Y_i and Y_{i+1}.

The potential functions can be defined in terms of *feature functions* [8], which are based on real-valued features that express some characteristic of the empirical distribution of the training data. For instance, a feature may represent the presence (1) or absence (0) of a word in a text sequence; or the presence of a certain element (edge, texture) in an image. For a first-order model (like a chain), the potential functions are defined in terms of the feature functions of the entire observation sequence, $\mathbf{X}$, and a pair of consecutive labels, Y_{i-1}, Y_i: $U(Y_{i-1}, Y_i, \mathbf{X}, f)$.

Considering a first-order chain, the energy function can be defined in terms of the potential function of single variables and pairs of variables, similar to MRFs:

$$E = \sum_j \lambda_j t_j(Y_{i-1}, Y_i, \mathbf{X}, f) + \sum_k \mu_k(Y_i, \mathbf{X}, f), \tag{6.20}$$

where λ_j and μ_k are parameters that weigh the contribution of the variable pairs (internal influence) and the single variables (external influence) respectively; these could be estimated from training data. The main difference from MRFs is that these potentials are conditioned on the entire observation sequence. Parameter estimation and inference are performed in a similar way as for MRFs.

6.7 Applications

Markov random fields have been applied to several tasks in image processing and computer vision. For example, MRFs are used for image smoothing, image restoration, segmentation, image registration, texture synthesis, superresolution, stereo matching, image annotation, and information retrieval. We describe two applications: image smoothing and improving image annotation.

6.7.1 Image Smoothing

Digital images are usually corrupted by high frequency noise. For reducing the noise a *smoothing* process can be applied to the image. For this, there are several alternatives; one is to use an MRF.

We can define an MRF associated to a digital image, in which each pixel corresponds to a random variable. Considering a first-order MRF, each interior variable is connected to its four neighbors. Additionally, each variable is also connected

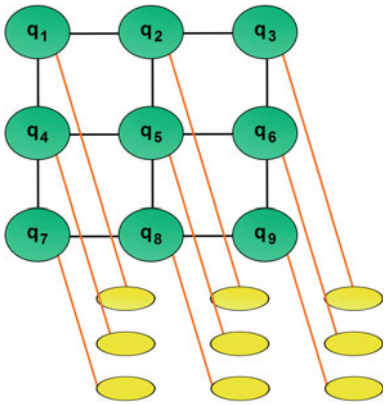


Fig. 6.7 An example of an MRF associated to an image. The *upper part* depicts a first-order 2D MRF for a 3×3 image. The *lower part* represents the image pixels, each one connected to a corresponding variable in the field

to an *observation* variable that has the value of the corresponding pixel in the image, see Fig. 6.7.

Once the structure of the MRF is defined, we need to specify the local potential functions. A property of natural images is that, in general, they have certain continuity, that is, neighboring pixels will tend to have similar values. Thus, we can propose a restriction that forces neighboring pixels to have similar values, by punishing (higher energy) configurations in which neighbors have different values. At the same time, it is desirable for each variable in the MRF to have a value similar to the one in the original image; so we also punish configurations in which the variables have different values to their corresponding observations. So the solution will be a compromise between these two types of restrictions, similarity to neighbors and similarity to observations.

The energy function, in this case, can be expressed as the sum of two types of potentials: one associated to pairs of neighbors, $U_c(f_i, f_j)$; and the other for each variable and its corresponding observation, $U_o(f_i, g_i)$. Thus, the energy will be the summation of these two types of potentials:

$$\mathbf{U_F} = \sum_{i,j} U_c(F_i, F_j) + \lambda \sum_i U_o(F_i, G_i) \tag{6.21}$$

where λ is a parameter which controls which aspect is given more importance, the observations ($\lambda > 1$) or the neighbors ($\lambda < 1$); and G_i is the observation variable associated to F_i.

Depending on the desired behavior for each type of potential, these can be defined to penalize the difference with the neighbors or the observations. Thus, a reasonable function is the quadratic difference. Then, the neighbors potential is:

$$U_c(f_i, f_j) = (f_i - f_j)^2 \tag{6.22}$$

Fig. 6.8 An illustration of image smoothing with an MRF. *Left* original image; *center* processed image with $\lambda = 1$; *right* processed image with $\lambda = 0.5$. It can be observed that a smaller value of λ produces a smoother image

And the observation potential is:

$$U_o(f_i, g_i) = (f_i - g_i)^2 \tag{6.23}$$

Using these potentials and applying the stochastic optimization algorithm, a smoothed image is obtained as the final configuration of $\mathbf{F}$. Figure 6.8 illustrates the application of a smoothing MRF to a digital image varying the value of λ.

6.7.2 Improving Image Annotation

Automatic image annotation is the task of automatically assigning annotations or labels to images or segments of images, based on their local features. Image annotation is frequently performed by automatic systems; it is a complex task due to the difficulty of extracting adequate features which allow to generalize and distinguish an object of interest from others with similar visual properties. Erroneous labeling of regions is a common consequence of the lack of a good characterization for the classes by low-level features.

When labeling a segmented image, we can incorporate additional information to improve the annotation of each region of the image. The labels of each region of an image are usually not independent; for instance in an image of animals in the jungle, we will expect to find a sky region *above* the animal, and trees or plants *below* or *near* the animal. Thus, the *spatial relations* between the different regions in the image can help to improve the annotation [5].

We can use Markov random fields to represent the information about the spatial relations among the regions in an image, such that the probability of occurrence of a certain spatial relation between each pair of labels could be used to obtain the most probable label for each region, i.e., the most probable configuration of labels for the entire image. Thus, using an MRF we can combine the information provided by

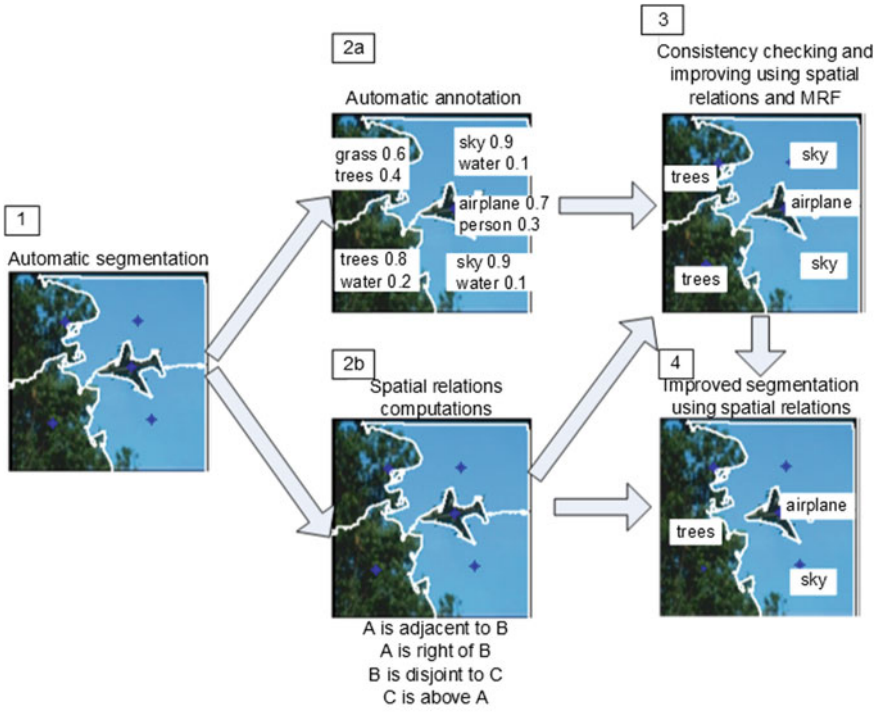


Fig. 6.9 Improving image labeling by incorporating spatial relations with an MRF. *1* Automatic image segmentation. *2a* Initial region labeling. *2b* Spatial relations among regions are obtained. *3* Improved labeling with an MRF. *4* Segmentation is improved [5]

the visual features for each region (external potential) and the information from the spatial relations with other regions in the image (internal potential). By combining both aspects in the potential function, and applying the optimization process, we can obtain a configuration of labels that *best* describe the image.

The procedure is basically the following (see Fig. 6.9):

1. An image is automatically segmented (using Normalized cuts).
2. The obtained segments are assigned a list of labels and their corresponding probabilities based on their visual features using a classifier.
3. Concurrently, the spatial relations among the same regions are computed.
4. The MRF is applied, combining the original labels and the spatial relations, resulting in a new labeling for the regions by applying simulated annealing.
5. Adjacent regions with the same label are joined.

The energy function to be minimized combines the information provided by the classifiers (labels' probabilities) with the spatial relations (relations' probabilities). In this study, spatial relations are divided into three groups: topological relations,

horizontal relations, and vertical relations. Thus, the energy function contains four terms, one for each type of spatial relation and one for the initial labels. So the energy function is:

$$U_p(f) = \alpha_1 V_T(f) + \alpha_2 V_H(f) + \alpha_3 V_V(f) + \lambda \sum_o V_o(f) \tag{6.24}$$

where V_T is the potential for topological relations, V_H for horizontal relations, and V_V for vertical relations; $\alpha_1, \alpha_2, \alpha_3$ are the corresponding constants for giving more or less weight to each type of relation. V_o is the classification (label) potential weighted by the λ constant. These potentials can be estimated from a set of labeled training images. The potential for a certain type of spatial relation between two regions of classes A and B is inversely proportional to the probability (frequency) of that relation occurring in the training set.

By applying this approach, a significant improvement can be obtained over the initial labeling of an image [6]. In some cases, by using the information provided by this new set of labels, we can also improve the initial image segmentation as illustrated in Fig. 6.10.

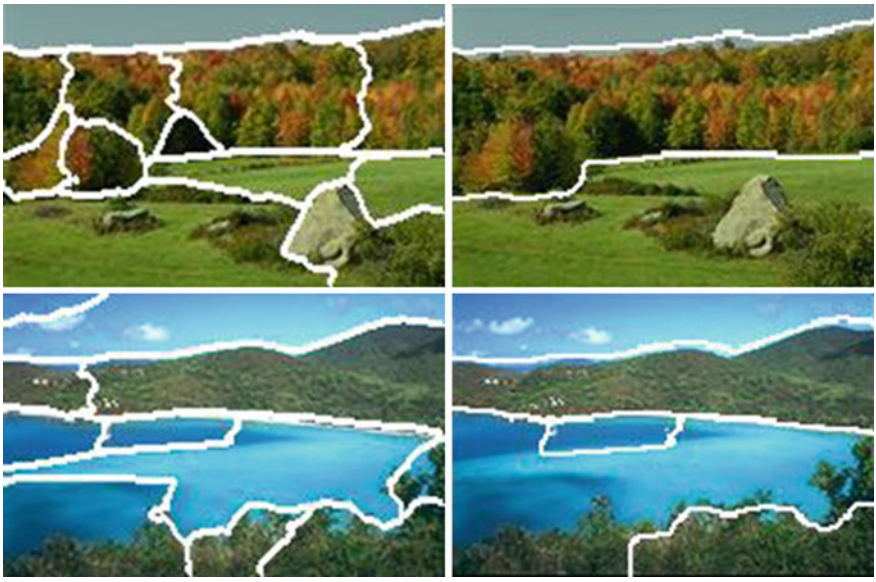


Fig. 6.10 An example of improving image segmentation by joining adjacent regions with the same label. *Left* original segmented images. *Right* improved segmentation [5]

6.8 Additional Reading

An introduction to Markov random fields and their applications is given in [7]. A comprehensive coverage of MRFs for image processing is presented in [9]. General introductions to conditional Markov random fields are given in [10, 11].

6.9 Exercises

1. For the Markov network in Fig. 6.3: (a) determine the cliques in the graph, (b) express the joint probability as a product of clique potentials, (c) assuming all the variables are binary, define the required parameters for this model.
2. Given the first-order MRF of Fig. 6.4, specify the minimum *Markov blanket* for each variable. The Markov blanket of a variable, q_i, is a set of variables that make it independent from the rest of the variables in the graph.
3. Repeat the previous problem for the second-order MRF of Fig. 6.5.
4. Given a regular MRF of 4×4 sites with a first-order neighborhood, consider that each site can take one of two values, 0 and 1. Consider that we use the smoothing potentials as in the image smoothing application, with $\lambda = 4$, giving more weight to the observations. Given the initial configuration F and the observation G, obtain the MAP configuration using the ICM variant of the stochastic simulation algorithm.

$$\mathrm{F}: \begin{matrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \end{matrix}$$

$$\mathrm{G}: \begin{matrix} 0\,0\,0\,0 \\ 0\,1\,1\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \end{matrix}$$

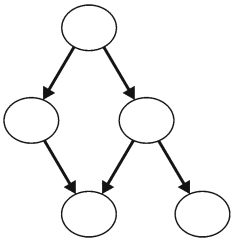
5. Repeat the previous problem using the Metropolis version of stochastic simulation, with $P = 0.5$. Hint: you can use a coin flip to decide if a higher energy configuration is kept or not.
6. Solve the previous two problems using MPM instead of MAP.
7. An edge in an image is where there is an abrupt change in the values of the neighboring pixels (a high value of the first derivative if we consider the image as a two dimensional function). Specify the potentials for a first-order MRF that emphasizes the edges in an image, considering that each site is binary, where 1 indicates an edge and 0 no edge. The observation is a gray level image in which each pixel varies from 0 to 255.

8. What is the time complexity of the stochastic simulation algorithm for its different variants?
9. *** Implement the image smoothing algorithm using a first-order regular MRF. Vary the parameter λ and observe the effects on the processed image. Repeat considering a second-order MRF.
10. *** Implement a program to generate a superresolution image using MRFs. For example, generate an image that doubles the dimensions $2n \times 2m$ of the original $n \times m$ image.

References

1. Besag, J.: Statistical analysis of non-lattice data. Statistician **24**(3), 179–195 (1975)
2. Binder, K.: Ising Model. Hazewinkel, Michiel, Encyclopedia of Mathematics. Springer, New York (2001)
3. Geman, D., Geman, S., Graffigne, C.: Locating Object and Texture Boundaries. Pattern recognition theory and applications. Springer, Heidelberg (1987)
4. Hammersley, J.M., Clifford, P.: Markov fields on finite graphs and lattices. Unpublished Paper. http://www.statslab.cam.ac.uk/grg/books/hammfest/hamm-cliff.pdf (1971). Accessed 14 Dec 2014
5. Hernández-Gracidas, C., Sucar, L.E.: Markov random fields and spatial information to improve automatic image annotation. Advances in Image and Video Technology. Lecture Notes in Computer Science, vol. 4872, pp. 879–892. Springer (2007)
6. Hernández-Gracidas, C., Sucar, L.E., Montes, M.: Improving image retrieval by using spatial relations. J. Multimed. Tools Appl. **62**, 479–505 (2013)
7. Kindermann, R., Snell, J.L.: Markov random fields and their applications. Am. Math. Soc. **34**, 143–167 (1980)
8. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: International Conference on Machine Learning (2001)
9. Li, S.Z.: Markov Random Field Modeling in Image Analysis. Springer, London (2009)
10. Sutton, C., McCallum, A.: An Introduction to Conditional Random Fields for Relational Learning. In: Geetor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning, MIT Press, Cambridge (2006)
11. Wallach, H.M.: Conditional random fields: an introduction. Technical Report MS-CIS-04-21, University of Pennsylvania (2004)

Chapter 7
Bayesian Networks: Representation and Inference



7.1 Introduction

In contrast to Markov networks, which are undirected graphical models, Bayesian networks are directed graphical models that represent the joint distribution of a set of random variables. Some of the techniques that we have revised in previous chapters, such as Bayesian classifiers and HMMs, are particular cases of Bayesian networks. Given their importance and the amount of research done for this topic in recent years, we have devoted two chapters to Bayesian networks. In this chapter, we will cover the representational aspects and inference techniques. The next chapter will discuss learning; specifically, structure and parameter learning.

An example of a hypothetical medical Bayesian network is shown in Fig. 7.1. In this graph, the nodes represent random variables and the arcs direct dependencies between variables. The structure of the graph encodes a set of conditional independence relations between the variables. For instance, the following conditional independencies can be inferred from the example:

- *Fever* is independent of *body ache* given *flu* (common cause).
- *Fever* is independent of *unhealthy food* given *typhoid* (indirect cause).
- *Typhoid* is independent of *flu* when *Fever* is NOT known (common effect). Knowing fever makes typhoid and flu dependent—for example, if we now that someone has typhoid and fever, this *diminishes* the probability of having flu.

In addition to the structure, a Bayesian network considers a set of local parameters, which are the conditional probabilities for each variable given its parents in the graph. For example, the conditional probability of fever given flu and typhoid, $P(fever \mid typhoid, flu)$. Thus, the joint probability of all the variables in the network can be represented based on these local parameters; this usually implies an important saving in the number of required parameters.

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_7

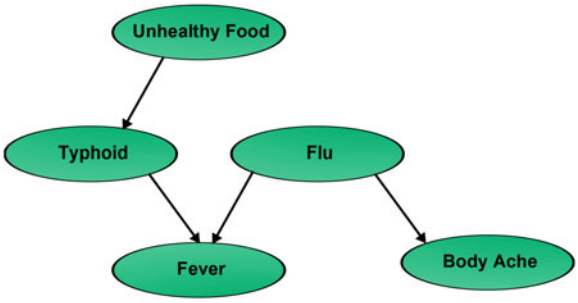


Fig. 7.1 A simple, hypothetical example of a medical Bayesian network

Given a Bayesian network (structure and parameters), we can answer several probabilistic queries. For instance, for the previous example: What is the probability of Fever given Flu? Which is more probable, Typhoid or Flu, given Fever and Unhealthy food?

In the next section, we formalize the representation of a Bayesian network, and then we present several algorithms to answer different types of probabilistic queries.

7.2 Representation

A Bayesian network (BN) represents the joint distribution of a set of n (discrete) variables, $X_1, X_2, \ldots, X_n$, as a directed acyclic graph (DAG) and a set of conditional probability tables (CPTs). Each node, that corresponds to a variable, has an associated CPT that contains the probability of each state of the variable given its parents in the graph. The structure of the network implies a set of conditional independence assertions, which give power to this representation.

Figure 7.2 depicts an example of a simple BN. The structure of the graph implies a set of conditional independence assertions for this set of variables. For example,

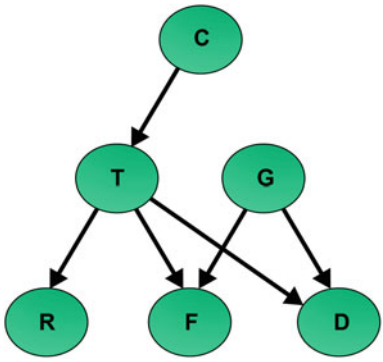


Fig. 7.2 A Bayesian network

R is conditionally independent of C, G, F, D given T, that is:

$$P(R \mid C, T, G, F, D) = P(R \mid T) \tag{7.1}$$

7.2.1 Structure

The conditional independence assertions implied by the structure of a BN should correspond to the conditional independence relations of the joint probability distribution, and vice versa. These are usually represented using the following notation. If X is conditionally independent of Z given Y:

- In the probability distribution: $P(X|Y, Z) = P(X|Y)$.
- In the graph: $I < X \mid Y \mid Z >$.

Conditional independence assertions can be verified directly from the structure of a BN using a criteria called *D-separation*. Before we define it, we consider the 3 basic BN structures for 3 variables and 2 arcs:

- Sequential: $X \rightarrow Y \rightarrow Z$.
- Divergent: $X \leftarrow Y \rightarrow Z$.
- Convergent: $X \rightarrow Y \leftarrow Z$.

In the first two cases, X and Z are conditionally independent given Y; however, in the third case this is not true. This last case, called *explaining away*, corresponds intuitively to having two *causes* with a common *effect*; knowing the effect and one of the causes, alters our belief in the other cause. These cases can be associated to the separating node, Y, in the graph. Thus, depending on the case, Y is sequential, divergent, or convergent.

D-Separation

Given a graph G, a set of variables A is conditionally independent of a set B given a set C, if there is no trajectory in G between A and B such that:

1. All convergent nodes are or have descendants in C.
2. All other nodes are outside C.

For instance, for the BN in Fig. 7.2, R is independent of C given T, but T and G are not independent given F.

Another way to verify D-Separation is by using an algorithm known as the *Bayes ball*. Consider that we have a path from node X to Z with Y in the middle (see Fig. 7.3); Y is shaded if it is known (instantiated), otherwise it is not shaded. We *throw a ball* from X to Z, if the ball arrives to Z then X and Z are NOT independent given Y according to the following rules:

1. If Y is sequential or divergent and is not shaded, the ball goes through.
2. If Y is sequential or divergent and it is shaded, the ball is blocked.

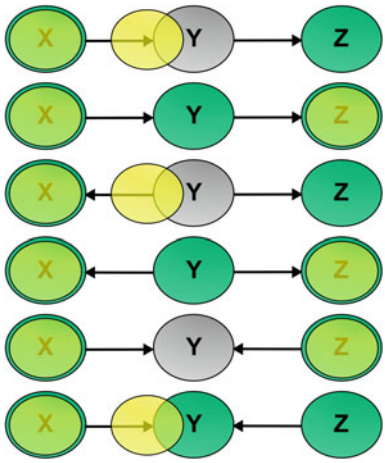


Fig. 7.3 An illustration of the different cases for D-Separation using the Bayes ball procedure. In the cases where the ball is blocked by Y the conditional independence condition is satisfied; when the ball passes (over Y) the conditional independence condition is not satisfied

3. If Y is convergent and not shaded, the ball is blocked.
4. If Y is convergent and shaded, the ball goes through.

According to the previous definition of D-separation, any node X is conditionally independent of all nodes in G that are not descendants of X given its parents in the graph, $Pa(X)$. This is known as the *Markov assumption*. The structure of a BN can be specified by the parents of each variable; thus the set of parents of a variable X is known as the *contour* of X. For the example in Fig. 7.2, its structure can be specified as:

1. Pa(C) = ∅
2. Pa(T) = C
3. Pa(G) = ∅
4. Pa(R) = T
5. Pa(F) = T, G
6. Pa(D) = T, G

Given this condition and using the chain rule, we can specify the joint probability distribution of the set of variables in a BN as the product of the conditional probability of each variable given its parents:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | Pa(X_i)) \tag{7.2}$$

For the example in Fig. 7.2:

$$P(C, T, G, R, F, D) = P(C)P(G)P(T \mid C)P(R \mid T)P(F \mid T, G)P(D \mid T, G)$$

The *Markov Blanket* of a node X, $MB(X)$, is the set of nodes that make it independent of all the other nodes in G, that is $P(X \mid G - X) = P(X \mid MB(X))$. For a BN, the Markov blanket of X is:

- the parents of X,
- the sons of X,
- and other parents of the sons of X.

For instance, in the BN of Fig. 7.2, the Markov blanket of R is T and the Markov blanket of T is C, R, F, D, G.

Mappings

Given a probability distribution P of $\mathbf{X}$, and its graphical representation G, there must be a correspondence between the conditional independence in P and in G; this is called a *mapping*. There are three basic types of mappings:

D-Map: all the conditional independence relations in P are satisfied (by D-Separation) in G.
I-Map: all the conditional independence relations in G are true in P.
P-Map: or perfect map, it is a D-Map and an I-Map.

In general, it is not always possible to have a *perfect mapping* of the independence relations between the graph (G) and the distribution (P), so we settle for what is called a *Minimal I-Map*: all the conditional independence relations implied by G are true in P, and if any arc is deleted in G this condition is lost [14].

Independence Axioms

Given some conditional independence relations between subsets of random variables, we can derive other conditional independence relations axiomatically, that is, without the need to estimate probabilities or independence measures. There are some basic rules to derive new conditional independence relations from other conditional independence relations, known as the *independence axioms*:

Symmetry: $I(X, Z, Y) \rightarrow I(Y, Z, X)$
Decomposition: $I(X, Z, Y \cup W) \rightarrow I(X, Z, Y) \wedge I(X, Z, W)$
Weak Union: $I(X, Z, Y \cup W) \rightarrow I(X, Z \cup W, Y)$
Contraction: $I(X, Z, Y) \wedge I(X, Z \cup Y, W) \rightarrow I(X, Z, Y \cup W)$
Intersection: $I(X, Z \cup W, Y) \wedge I(X, Z \cup Y, W) \rightarrow I(X, Z, Y \cup W)$

Graphical examples of the application of the independence axioms are illustrated in Fig. 7.4.

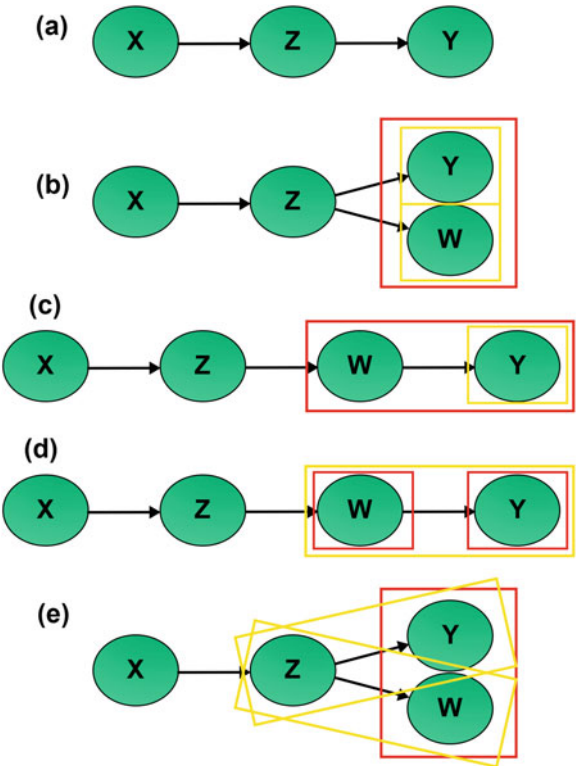


Fig. 7.4 Graphical examples of the independence axioms: **a** Symmetry, **b** Decomposition, **c** Weak Union, **d** Contraction, **e** Intersection

7.2.2 Parameters

To complete the specification of a BN, we need to define its parameters. In the case of a BN, these parameters are the conditional probabilities of each node given its parents in the graph. If we consider discrete variables:

- Root nodes: vector of marginal probabilities.
- Other nodes: conditional probability table (CPT) of the variable given its parents in the graph.

Figure 7.5 shows some of the CPTs of the BN in Fig. 7.2. In case of continuous variables, we need to specify a function that relates the density function of each variable to the density of its parents (for example, Kalman filters consider Gaussian distributed variables and linear functions).

In the case of discrete variables, the number of parameters in a CPT increases exponentially with the number of parents of a node. This can become problematic when there are *many* parents. The memory requirements can become very large, and it is also difficult to estimate so many parameters. Two main alternatives have been

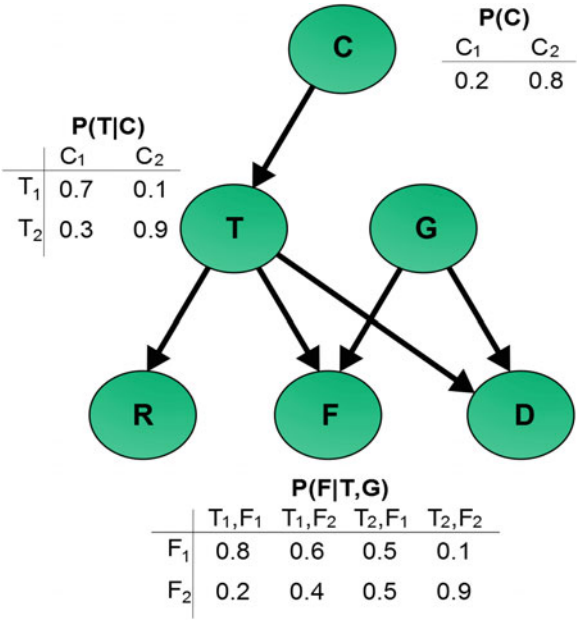


Fig. 7.5 Parameters for the BN in Fig. 7.2. It shows the CPTs for some of the variables in the example: $P(C)$; $P(T \mid C)$; and $P(F \mid T, G)$. We assume in this case that all variables are binary

proposed to overcome this issue, one is based on *canonical models* and the other on graphical representations of CPTs. Next we briefly present both schemes.

7.2.2.1 Canonical Models

Canonical models represent the relations between a set of random variables for particular interactions using few parameters. It can be applied when the probabilities of a random variable in a BN conform to certain *canonical* relations with respect to the configurations of its parents. There are several classes of canonical models, the most common are the *Noisy OR* and *Noisy AND* for binary variables, and their extensions for multivalued variables, *Noisy Max* and *Noisy Min*, respectively.

The Noisy OR is basically an extension of the OR relation in logic. Consider an OR logic gate, in which the output is *True* if any of its inputs are *True*. The Noisy OR model is based on the concept of the logic OR; the difference is that there is a certain (small) probability that the variable is not *True* even if one or more of its parents are *True*. In an analogous way, the Noisy And model is related to the logical AND. These models apply only when all the variables are binary; however, there are extensions for multivalued variables, which consider a set of *ordered* values for each variable. For example, consider a variable that represents a disease, D. In the case of the binary canonical models, it has two values, *True* and *False*. For a multivalued model, it could be defined as $D \in \{False, Mild, Intermediate, andSevere\}$, such that these

values follow a predefined order. The *Noisy Max* and *Noisy Min* models generalize the Noisy OR and Noisy AND models, respectively, for multivalued-ordered variables.

Next, we describe the Noisy OR model in detail; the other cases can be defined in a similar way.

Noisy OR

The Noisy OR model is applied when several variables or *causes* can produce an *effect* if any one of them is *True*, and as more of the *causes* are true, the probability of the effect increases. For instance, the effect could be a certain symptom, S, and the causes are a number of possible diseases, $D_1, D_2, \ldots, D_m$, that can produce the symptom, such that if none of the diseases is present (all *False*) the symptom does not appear; and when any disease is present (*True*) the symptom is present with high probability and increases as the number of $D_i = True$ increases. A graphical representation of a Noisy OR relation in a BN is depicted in Fig. 7.6.

Formally, the following two conditions must be satisfied for a Noisy OR canonical model to be applicable:

Responsibility: the effect is false if all the possible causes are false.
Independence of exceptions: if an effect is the manifestation of several causes, the mechanisms that inhibit the occurrence of the effect under one cause are independent of the mechanisms that inhibit it under the other causes.

The probability that the effect E is inhibited (it does not occur) under cause C_i is defined as:

$$q_i = P(E = False \mid C_i = True) \tag{7.3}$$

Given this definition and the previous conditions, the parameters in the CPT for a Noisy OR model can be obtained using the following expressions when all the m causes are *True*:

$$P(E = False \mid C_1 = True, \ldots C_m = True) = \prod_{i=i}^{m} q_i \tag{7.4}$$

$$P(E = True \mid C_1 = True, \ldots C_m = True) = 1 - \prod_{i=i}^{m} q_i \tag{7.5}$$

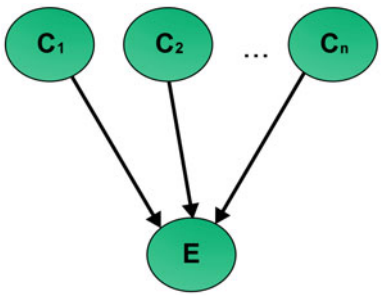


Fig. 7.6 Graphical representation of a Noisy OR structure. The n cause variables (C) are the parents of the effect variable (E)

Table 7.1 Conditional probability table for a Noisy OR variable with three parents and parameters $q_1 = q_2 = q_3 = 0.1$

C_1	0	0	0	0	1	1	1	1
C_2	0	0	1	1	0	0	1	1
C_3	0	1	0	1	0	1	0	1
$P(E = 0)$	1	0.1	0.1	0.01	0.1	0.01	0.01	0.001
$P(E = 1)$	0	0.9	0.9	0.99	0.9	0.99	0.99	0.999

In general, if k of m causes are *True*, then $P(E = False \mid C_1 = True, \ldots C_k = True) = \prod_{i=i}^{k} q_i$, so that if all the causes are *False* then the effect is *False* with a probability of one. Thus, only one parameter is required per parent variable to construct the CPT, the inhibition probability q_i. In this case, the number of parameters increases linearly with the number of parents, instead of exponentially.

As an example, consider a Noisy OR model with 3 causes, C_1, C_2, C_3, where the inhibition probability is the same for the three, $q_1 = q_2 = q_3 = 0.1$. Given these parameters, we can obtain the CPT for the effect variable, as shown in Table 7.1.

7.2.2.2 Other Representations

Canonical models apply in certain situations but do not provide a general solution for compact representations of CPTs. An alternative representation is based on the observation that in the probability tables for many domains, the same probability values tend to be repeated several times in the same table; for instance, it is common to have many zero entries in a CPT. Thus, it is not necessary to represent these repeated values many times, is should be sufficient to represent each different value once.

A representation that takes advantage of this condition is a *decision tree* (DT), such that it could be used for representing a CPT in a compact way. In a DT, each internal node corresponds to a variable in the CPT, and the branches from a node correspond to the different values a variable can take. The leaf nodes in the tree represent the different probability values. A trajectory from the root to a leaf specifies a probability value for the corresponding variables–values in the trajectory. If a variable is omitted in a trajectory, it means that the CPT has the same probability values for this variable.

For example, Table 7.2 depicts the CPT $P(X \mid A, B, C, D, E, F, G)$, assuming all variables are binary (F, T). Figure 7.7 shows a DT for the CPT in Table 7.2. In this example, the savings in memory are not significant; however, for *large* tables there could be a significant reduction in the memory space requirements.

A *decision diagram* (DD) extends a DT by considering a directed acyclic graph structure, such that it is not restricted to a tree. This avoids the need to duplicate repeated probability values in the leaf nodes, and in some cases provides an even more compact representation. An example of a decision diagram representation of the CPT of Table 7.2 is depicted in Fig. 7.8.

Table 7.2 Conditional probability table of $P(X \mid A, B, C, D, E, F, G)$, which has several repeated values

A	B	C	D	E	F	G	X
T	T/F	T/F	T/F	T/F	T/F	T/F	0.9
F	T	T/F	T	T/F	T	T	0.9
F	T	T/F	T	T/F	T	F	0.0
F	T	T/F	T	T/F	F	T/F	0.0
F	T	T	F	T	T/F	T	0.9
F	T	T	F	T	T/F	F	0.0
F	T	T	F	F	T/F	T/F	0.0
F	T	F	F	T/F	T/F	T/F	0.0
F	F	T	T/F	T	T/F	T	0.9
F	F	T	T/F	T	T/F	F	0.0
F	F	T	T/F	F	T/F	T/F	0.0
F	F	F	T/F	T/F	T/F	T/F	0.0

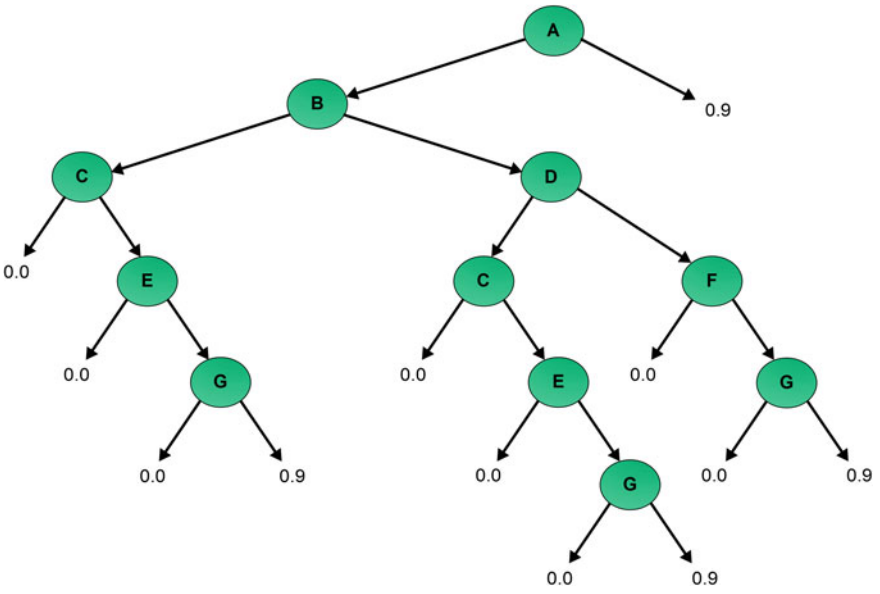


Fig. 7.7 Decision tree representation of a CPT. The DT represents the CPT shown in Table 7.2. For each variable (node in the tree), the *left arrow* corresponds to the value F and the *right arrow* to the value T

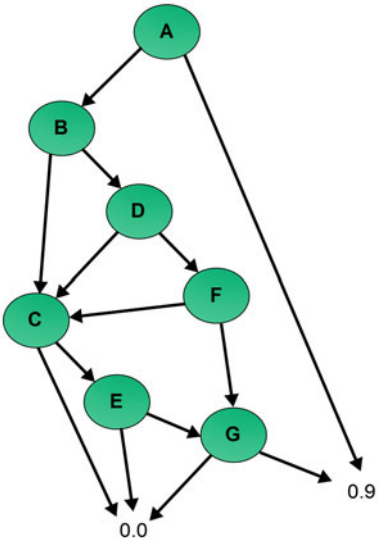


Fig. 7.8 Decision diagram representation of a CPT. The DD represents the CPT shown in Table 7.2. As in Fig. 7.7, the *left arrow* corresponds to the value F and the *right arrow* to the value T

7.3 Inference

Probabilistic inference consists in *propagating* the effects of certain evidence in a Bayesian network to estimate its effect on the unknown variables. That is, by knowing the values for some subset of variables in the model, the posterior probabilities of the other variables are obtained. The subset of unknown variables could be empty; in this case, we obtain the prior probabilities of all the variables.

There are basically two variants of the inference problem in BNs. One is obtaining the posterior probability of a single variable, H, given a subset of known (instantiated) variables, $\mathbf{E}$, that is, $P(H \mid \mathbf{E})$. Specifically, we are interested in the marginal probabilities of the unknown variables in the model. This is the most common application of BNs, and we will denominate it as *single query inference*.

The second variant consists in calculating the posterior probability of a set of variables, $\mathbf{H}$ given the evidence, $\mathbf{E}$, that is, $P(\mathbf{H} \mid \mathbf{E})$. This is known as *conjunctive query inference*. In principle, it can be solved using single query inference several times by applying the chain rule, making it a more complex problem. For example, $P(A, B \mid \mathbf{E})$ can be written as $P(A \mid \mathbf{E})P(B \mid A, \mathbf{E})$, which requires two single query inferences, and a multiplication. In some applications, it is of interest to know which are the most probable values in the set of hypothesis. That is, $ArgMax_H P(\mathbf{H} \mid \mathbf{E})$. When $\mathbf{H}$ includes all nonobserved variables, it is known as the *most probable explanation* (MPE) or the *total abduction* problem. When we are interested in the most likely joint state of some (not all) of the unobserved variables, it corresponds to the *maximum a posteriori assignment* (MAP) or *partial abduction* problem.

We will first focus on the single query inference problem, and later on the MPE and MAP problems. If we want to solve the inference problem using a direct (brute

force) computation (i.e., from the joint distribution), the computational complexity increases exponentially with respect to the number of variables, and the problem becomes intractable even with few variables. Many algorithms have been developed for making this process more efficient, which can be roughly divided into the following classes:

1. Probability propagation (Pearl's algorithm [13]).
2. Variable elimination.
3. Conditioning.
4. Junction tree.
5. Stochastic simulation.

The probability propagation algorithm only applies to singly connected graphs (trees and polytrees[1]), although there is an extension for general networks called *loopy propagation* which does not guarantee convergence. The other four classes of algorithms work for any network structure, the last one being an approximate technique, while the other three are exact.

In the worst case, the inference problem is *NP-hard* for Bayesian networks [1]. However, there are efficient (polynomial) algorithms for certain types of structures (singly connected networks); while for other structures it depends on the connectivity of the graph. In many applications, the graphs are *sparse* and in this case there are inference algorithms which are very efficient.

Next, we will describe probability propagation for singly connected networks, and then the most common techniques used for multiconnected BNs.

7.3.1 Singly Connected Networks: Belief Propagation

We now describe the tree propagation algorithm proposed by Pearl, which provides the basis for several of the more advanced and general techniques.

Given certain evidence, **E** (subset of instantiated variables), the posterior probability for a value i of any variable B, can be obtained by applying the Bayes rule:

$$P(Bi|\mathbf{E}) = P(Bi)P(\mathbf{E}|Bi)/P(\mathbf{E}) \tag{7.6}$$

Given that the BN has a tree structure, any node divides the network into two independent subtrees. Thus, we can separate the evidence into (see Fig. 7.9):

E-: Evidence in the tree rooted in B.
E+: All other evidence.

Then:

$$P(Bi|\mathbf{E}) = P(Bi)P(\mathbf{E}-, \mathbf{E}+|Bi)/P(\mathbf{E}) \tag{7.7}$$

[1] A polytree is a singly connected DAG in which some nodes have more than one parent; in a directed tree, each node has at most one parent.

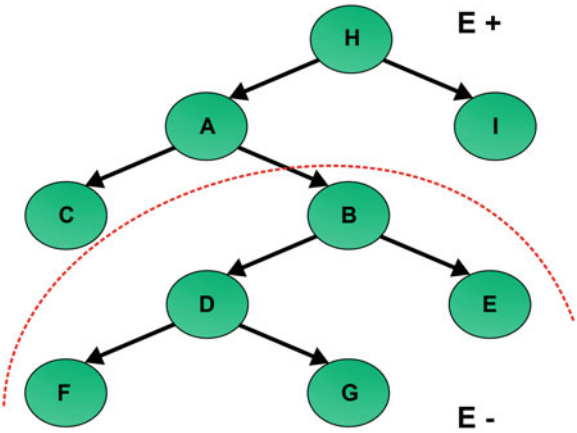


Fig. 7.9 In a tree–structured BN, every node (B) divides the network into two conditionally independent subtrees, $\mathbf{E}+$ and $\mathbf{E}-$

Given that $\mathbf{E}+$ and $\mathbf{E}-$ are independent, by applying the Bayes rule again, we obtain:

$$P(Bi|\mathbf{E}) = \alpha P(Bi|\mathbf{E}+)P(\mathbf{E}-|Bi) \tag{7.8}$$

where α is a normalization constant. If we define the following terms:

$$\lambda(Bi) = P(\mathbf{E}-|Bi) \tag{7.9}$$

$$\pi(Bi) = P(Bi|\mathbf{E}+) \tag{7.10}$$

Then Eq. (7.8) can be written as:

$$P(Bi|\mathbf{E}) = \alpha\pi(Bi)\lambda(Bi) \tag{7.11}$$

Equation (7.11) is the basis for a distributed propagation algorithm to obtain the posterior probability of all noninstantiated nodes. The computation of the posterior probability of any node B is decomposed into two parts: (i) the evidence coming from the sons of B in the tree (λ), and the evidence coming from the parent of B, (π). We can think of each node B in the tree as a simple processor that stores its vectors $\pi(B)$ and $\lambda(B)$, and its conditional probability table, $P(B \mid A)$. The evidence is propagated via a message passing mechanism, in which each node sends the corresponding messages to its parent and sons in the tree. A message sent from node B to its parent A:

$$\lambda_B(Ai) = \sum_j P(B_j \mid A_i)\lambda(B_j) \tag{7.12}$$

A message sent from node B to its son S_k:

$$\pi_k(Bi) = \alpha\pi(B_j)\prod_{l\neq k}\lambda_l(B_j) \tag{7.13}$$

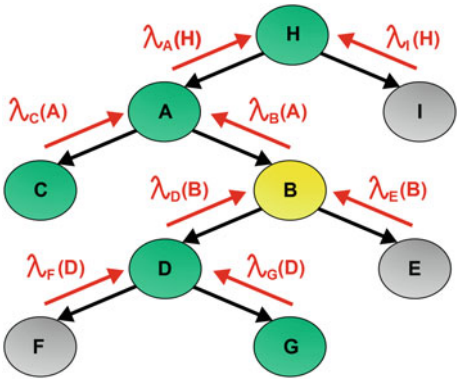


Fig. 7.10 Bottom-up propagation. λ messages are propagated from the leaf nodes to the root

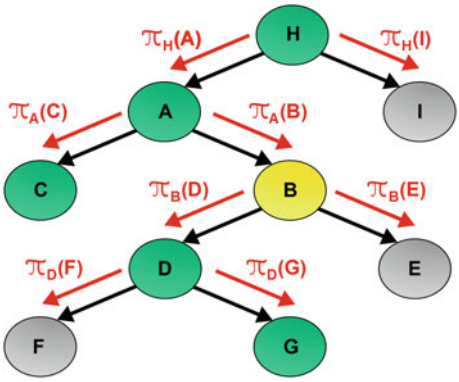


Fig. 7.11 Top-down propagation. π messages are propagated from the root node to the leaves

where l refers to each one of the sons of B.

Each node can receive several λ messages, which are combined via a term-by-term multiplication for the λ messages received from each son. Therefore, the λ for a node A with m sons is obtained as:

$$\lambda(Ai) = \prod_{j=1}^{m} \lambda_{Sj}(Ai) \tag{7.14}$$

The propagation algorithm starts by assigning the evidence to the known variables, and then propagating it through the message passing mechanism until the root of the tree is reached for the λ messages, and the leaves are reached for the π messages. Figures 7.10 and 7.11 illustrate the propagation scheme. At the end of the propagation, each node has its updated λ and π vectors. The posterior probability of any variable B is obtained by combining these vectors using Eq. (7.11) and normalizing.

For the root and leaf nodes we need to define some initial conditions:

Leaf nodes: If not known, $\lambda = [1, 1, \ldots, 1]$ (a uniform distribution). If known, $\lambda = [0, 0, \ldots, 1, \ldots, 0]$ (one for the assigned value and zero for all other values).

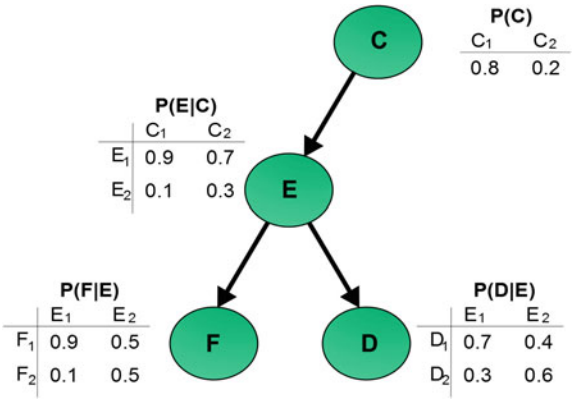


Fig. 7.12 A simple BN used in the belief propagation example

Root node: If not known, $\pi = P(A)$ (prior marginal probability vector). If known, $\pi = [0, 0, \ldots, 1, \ldots, 0]$ (one for the assigned value and zero for all other values).

We now illustrate the belief propagation algorithm with a simple example. Consider the BN in Fig. 7.12 with 4 binary variables (each with values *false* and *true*), C, E, F, D, with the CPTs shown in the figure.

Consider that the only evidence is $F = \textit{false}$. Then the initial conditions for the leaf nodes are: $\lambda_F = [1, 0]$ and $\lambda_D = [1, 1]$ (no evidence). Propagating to the parent node (E) is basically multiplying the λ vectors by the corresponding CPTs:

$$\lambda_F(E) = [1, 0] \begin{bmatrix} 0.9, 0.5 \\ 0.1, 0.5 \end{bmatrix} = [0.9, 0.5]$$

$$\lambda_D(E) = [1, 1] \begin{bmatrix} 0.7, 0.4 \\ 0.3, 0.6 \end{bmatrix} = [1, 1]$$

Then, $\lambda(E)$ is obtained by combining the messages from its two sons:

$$\lambda(E) = [0.9, 0.5] \times [1, 1] = [0.9, 0.5]$$

And now it is propagated to its parent, C:

$$\lambda_E(C) = [0.9, 0.5] \begin{bmatrix} 0.9, 0.7 \\ 0.1, 0.3 \end{bmatrix} = [0.86, 0.78]$$

In this case, $\lambda(C) = [0.86, 0.78]$, as C has only one son. In this way, we complete the bottom-up propagation; we will now do it top-down.

Given that C is not instantiated, $\pi(C) = [0.8, 0.2]$, we propagate to its son, E, which also corresponds to multiplying the π vector by the corresponding CPT:

$$\pi(E) = [0.8, 0.2] \begin{bmatrix} 0.9, 0.7 \\ 0.1, 0.3 \end{bmatrix} = [0.86, 0.14]$$

We now propagate to its son D; however, given that E has another son, F, we also need to consider the λ message from this other son, thus:

$$\pi(D) = [0.86, 0.14] \times [0.9, 0.5] \begin{bmatrix} 0.7, 0.4 \\ 0.3, 0.6 \end{bmatrix} = [0.57, 0.27]$$

This completes the top-down propagation (we do not need to propagate to F as this variable is known). Given the λ and π vectors for each unknown variable, we just multiply them term by term and then normalize to obtain the posterior probabilities:

$$P(C) = [0.86, 0.2] \times [0.86, 0.78] = \alpha[0.69, 0.16] = [0.815, 0.185]$$
$$P(E) = [0.86, 0.14] \times [0.9, 0.5] = \alpha[0.77, 0.07] = [0.917, 0.083]$$
$$P(D) = [0.57, 0.27] \times [1, 1] = \alpha[0.57, 0.27] = [0.67, 0.33]$$

This concludes the belief propagation example.

Probability propagation is a very efficient algorithm for tree-structured BNs. The time complexity to obtain the posterior probability of all the variables in the tree is proportional to the *diameter* of the network (the number of arcs in the trajectory from the root to the most distant leaf).

The message passing mechanism can be directly extended to polytrees, as these are also singly connected networks. In this case, a node can have multiple parents, so the λ messages should be sent from a node to all its parents. The time complexity is in the same order as for tree structures.

The propagation algorithm only applies to single connected networks. Next, we will present general algorithms that apply to any structure.

7.3.2 Multiple Connected Networks

There are several classes of algorithms for probabilistic inference on multiconneced BNs. Next, we review the main ones: (i) variable elimination, (ii) conditioning, and (iii) junction tree.

7.3.2.1 Variable Elimination

The variable elimination technique is based on the idea of calculating the probability by marginalizing the joint distribution. However, in contrast to the naive approach, it takes advantage of the independence conditions of the BN and the associative and distributive properties of addition and multiplication to do the calculations more efficiently.

Assume a BN representing the joint probability distribution of $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$. We want to calculate the posterior probability of a certain variable or

subset of variables, X_H, given a subset of evidence variables, X_E; the remaining variables are X_R, such that $\mathbf{X} = \{X_H \cup X_E \cup X_R\}$.

The posterior probability of X_H given the evidence is:

$$P(X_H \mid X_E) = P(X_H, X_E)/P(X_E) \tag{7.15}$$

We can obtain both terms via marginalization of the joint distribution:

$$P(X_H, X_E) = \sum_{X_R} P(\mathbf{X}) \tag{7.16}$$

and

$$P(X_E) = \sum_{X_H} P(X_H, X_E) \tag{7.17}$$

A particular case of interest is to obtain the marginal probability of the variables when there is no evidence; in this case $X_E = \emptyset$. Another calculation of interest is to obtain the probability of the evidence; this is given by the last equation.

The objective of the variable elimination technique is to perform these calculations efficiently. To achieve this, we can first represent the joint distribution as a product of local probabilities according to the network structure. Then, summations can be carried out only on the subset of terms which are a function of the variables being normalized. This approach takes advantage of the properties of summation and multiplication, resulting in the number of necessary operations being reduced. Next we will illustrate the method through an example.

Consider the BN in Fig. 7.13 where we want to obtain $P(A \mid D)$. In order to achieve this, we need to obtain $P(A, D)$ and $P(D)$. To calculate the first term, we must *eliminate* B, C, E from the joint distribution, that is:

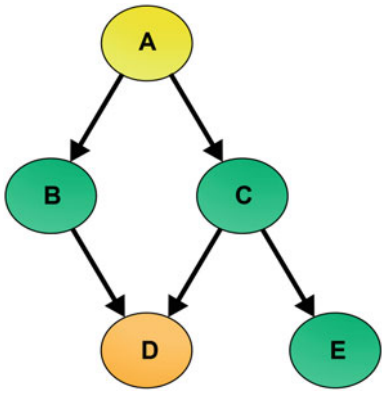


Fig. 7.13 A Bayesian network used to illustrate the variable elimination algorithm

$$P(A, D) = \sum_B \sum_C \sum_E P(A)P(B \mid A)P(C \mid A)P(D \mid B, C)P(E \mid C) \quad (7.18)$$

By *distributing* the summations we can arrive to the following equivalent expression:

$$P(A, D) = P(A) \sum_B P(B \mid A) \sum_C P(C \mid A)P(D \mid B, C) \sum_E P(E \mid C) \quad (7.19)$$

If we consider that all variables are binary, this implies a reduction from 32 operations to 9 operations; of course, this reduction will be more significant for larger models or when there are more values per variable.

As an example, consider the BN in Fig. 7.12 and that we want to obtain $P(E \mid F = f_1) = P(E, F = f_1)/P(F = f_1)$. Given the structure of the BN, the joint probability distribution is given by $P(C, E, F, D) = P(C)P(E \mid C)P(F \mid E)P(D \mid E)$. We first calculate $P(E, F)$; by reordering the operations:

$$P(E, F) = \sum_D P(F \mid E)P(D \mid E) \sum_C P(C)P(E \mid C)$$

We must do this calculation for each value of E, given $F = f_1$:

$$\begin{aligned}
P(e_1, f_1) &= \sum_D P(f_1 \mid e_1)P(D \mid e_1) \sum_C P(C)P(e_1 \mid C) \\
P(e_1, f_1) &= \sum_D P(f_1 \mid e_1)P(D \mid e_1)[0.9 \times 0.8 + 0.7 \times 0.2] \\
P(e_1, f_1) &= \sum_D P(f_1 \mid e_1)P(D \mid e_1)[0.86] \\
P(e_1, f_1) &= [0.9 \times 0.7 + 0.9 \times 0.3][0.86] \\
P(e_1, f_1) &= [0.9][0.86] = 0.774
\end{aligned}$$

In a similar way, we obtain $P(e_2, f_1)$; and then from these values we can calculate $P(f_1) = \sum_E P(E, f_1)$. Finally, we calculate the posterior probability of E given f_1: $P(e_1 \mid f_1) = P(e_1, f_1)/P(f_1)$ and $P(e_2 \mid f_1) = P(e_2, f_1)/P(f_1)$.

The critical aspect of the variable elimination algorithm is to select the appropriate order for eliminating each variable, as this has an important effect on the number of required operations. The different terms that are generated during the calculations are known as *factors* which are functions over a subset of variables, that map each instantiation of these variables to a non-negative number (these numbers are not necessarily probabilities). In general, a factor can be represented as $f(X_1, X_2, \ldots X_m)$. For instance, in the previous example, one of the factors is $f(C, E) = P(C)P(E \mid C)$, which is a function of two variables.

The computational complexity in terms of space and time of the variable elimination algorithm is determined by the size of the factors; that is, the number of variables, w, on which the factor is defined. Basically, the complexity for eliminating

(marginalize) any number of variables is exponential on the number of variables in the factor, $O(exp(w))$ [2]. Thus, the order in which the variables are eliminated should be selected so that the largest factor is kept to a minimum. However, finding the *best* order is in general a NP-Hard problem.

There are several heuristics that help to determine a *good* ordering for variable elimination. These heuristics can be explained based on the *interaction graph*—a undirected graph that is built during the process of variable elimination. The variable of each factor forms a clique in the interaction graph. The initial interaction graph is obtained from the original BN structure by eliminating the direction of the arcs, and adding additional arcs between each pair of nonconnected variables that have a common child. Then, each time a variable X_j is eliminated, the interaction graph is modified by: (i) adding an arc between each pair of neighbors of X_j that are not connected, (ii) deleting variable X_j from the graph.

We illustrate the interaction graphs that result from the BN in Fig. 7.13 by the following elimination ordering: E, C, B, D, depicted in Fig. 7.14.

Two popular heuristics for determining the elimination ordering, which can be obtained from the elimination graph, are the following:

Min-degree: eliminate the variable that leads to the smallest possible factor; which is equivalent to eliminating the variable with the smallest number of neighbors in the current elimination graph.

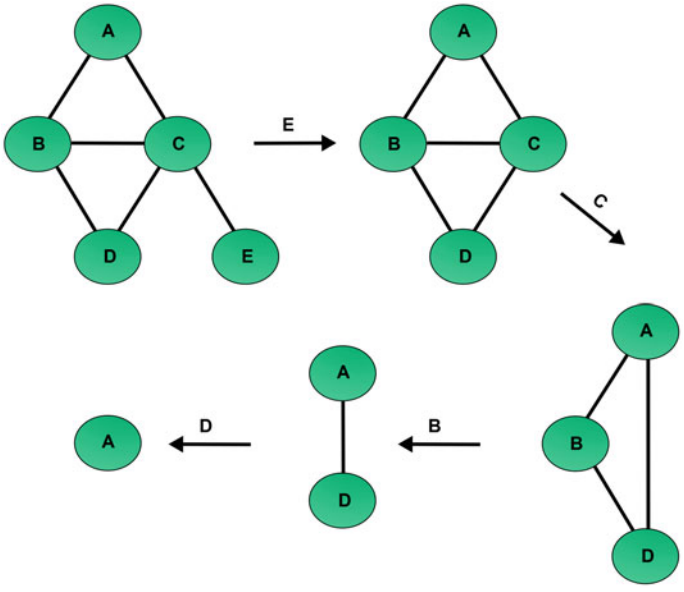


Fig. 7.14 Interaction graphs resulting from the elimination of variables with the following elimination ordering: E, C, B, D from the BN in Fig. 7.13

Min-fill: eliminate the variable that leads to adding the minimum number of edges to the interaction graph.

A disadvantage of variable elimination is that it only obtains the posterior probability of one variable (or subset of variables). To obtain the posterior probability of each noninstantiated variable in a BN, the calculations have to be repeated for each variable. Next, we describe two algorithms that calculate the posterior probabilities for all variables at the same time.

7.3.2.2 Conditioning

The conditioning method is based on the fact that an instantiated variable *blocks* the propagation of the evidence in a Bayesian network, Thus, we can *cut* the graph at an instantiated variable, and this can transform a multiconnected graph into a polytree, for which we can apply the probability propagation algorithm.

In general, a subset of variables can be instantiated to transform a multiconnected network into a singly connected graph. If these variables are not actually known, we can set them to each of their possible values, and then do probability propagation for each value. With each propagation, we obtain a probability for each unknown variable. Then, the final probability values are obtained as a weighted combination of these probabilities.

First we will develop the conditioning algorithm assuming we only need to partition a single variable and then we will extend it for multiple variables. Formally, we want to obtain the probability of any variable, B, given the evidence E, conditioning on variable A. By the rule of total probability:

$$P(B \mid E) = \sum_i P(B \mid E, a_i) P(a_i \mid E) \tag{7.20}$$

where:

$P(B \mid E, a_i)$ is the posterior probability of B which is obtained by probability propagation for each possible value of A.

$P(a_i \mid E)$ is a *weight*.

By applying the Bayes rule, we obtain the following equation to estimate the weights:

$$P(a_i \mid E) = \alpha P(a_i) P(E \mid a_i) \tag{7.21}$$

The first term, $P(a_i)$, can be obtained by propagating without evidence. The second term, $P(E \mid a_i)$, is calculated by propagation with $A = a_i$ to obtain the probability of the evidence variables. α is a normalizing constant.

For example, consider the BN in Fig. 7.13. This multiconnected network can be transformed into a polytree by assuming A is instantiated (see Fig. 7.15). If the

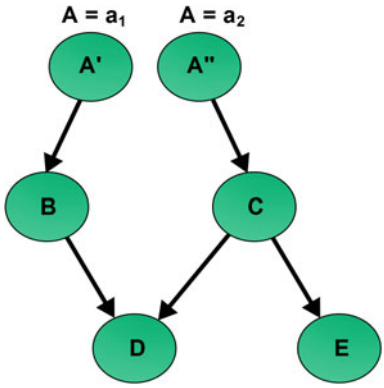


Fig. 7.15 The Bayesian network in Fig. 7.13 is transformed into a singly connected network by instantiating A

evidence is D, E, then probabilities for the other variables, A, B, C can be obtained via conditioning following these steps:

1. Obtain the prior probability of A (in this case it is already given as it is a root node).
2. Obtain the probability of the evidence nodes D, E for each value of A by propagation in the polytree.
3. Calculate the weights, $P(a_i \mid D, E)$, from (1) and (2) with the Bayes rule.
4. Estimate the probability of B and C for each value of A given the evidence by probability propagation in the polytree.
5. Obtain the posterior probabilities for B and C from (3) and (4) by applying Eq. (7.20).

In general, to transform a multiconnected BN to a polytree, we need to instantiate m variables. Thus, propagation must be performed for all the combinations of values (cross product) of the instantiated variables. If each variable has k values, then the number of propagations is k^m. The procedure is basically the same as described above for one variable, but the complexity increases.

7.3.2.3 Junction Tree Algorithm

The junction tree method is based on a transformation of the BN to a junction tree, where each node in this tree is a group or cluster of variables from the original network. Probabilistic inference is performed over this new representation. Next, we give a brief overview of the basic algorithm, for more details see the additional reading section.

The transformation proceeds as follows (see Fig. 7.16):

1. Eliminate the directionality of the arcs.
2. Order the nodes in the graph (based on *maximum cardinality*).
3. Moralize the graph (add an arc between pairs of nodes with common children).

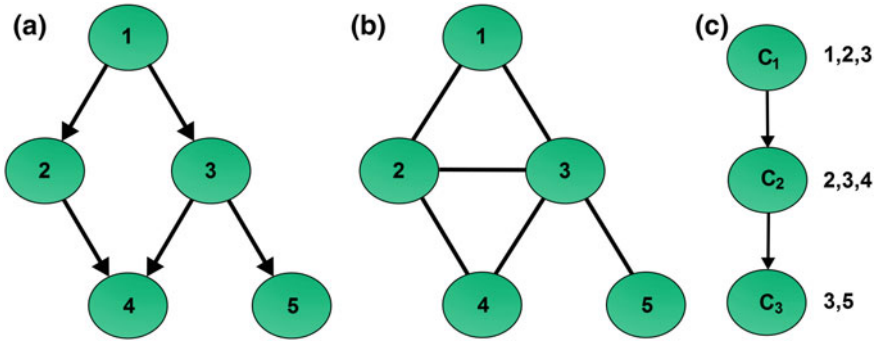


Fig. 7.16 Transformation of a BN to a junction tree: **a** original net, **b** triangulated graph, **c** junction tree

4. If necessary add additional arcs to make the graph *triangulated*.
5. Obtain the *cliques* of the graph (subsets of nodes that are fully connected and are not subsets of other fully connected sets).
6. Build a junction tree in which each node is a clique and its parent is any node that contains all common previous variables according to the ordering.

Once the junction tree is built, inference is based on probability propagation over the junction tree, in an analogous way as for tree-structured BNs. Initially, the joint probability (potential) of each macronode is obtained, and given some evidence, this is propagated to obtain the posterior probability of each junction. The individual probability of each variable is obtained from the joint probability of the appropriate junction via marginalization. We now describe the procedure in more detail.

The junction tree algorithm can be divided in two stages: preprocessing and propagation. In the preprocessing phase, the potentials of each clique are obtained following the next steps:

1. Determine the set of variables for each clique, C_i.
2. Determine the set of variables that are common with the previous (parent) clique, S_i.
3. Determine the variables that are in C_i but not in S_i: $R_i = C_i - S_i$.
4. Calculate the potential of each clique, clq_i, as the product of the corresponding CPTs: $\psi(clq_i) = \prod_j P(X_j \mid Pa(X_j))$; where X_j are the variables in clq_i.

For example, consider the BN in Fig. 7.16, with cliques: $clq_1 = \{1, 2, 3\}$, $clq_2 = \{2, 3, 4\}$, $clq_3 = \{3, 5\}$. Then the preprocessing phase is:

C: $C_1 = \{1, 2, 3\}$, $C_2 = \{2, 3, 4\}$, $C_3 = \{3, 5\}$.
S: $S_1 = \emptyset$, $S_2 = \{2, 3\}$, $S_3 = \{3\}$.
R: $R_1 = \{1, 2, 3\}$, $R_2 = \{4\}$, $R_3 = \{5\}$.
Potentials: $\psi(clq_1) = P(1)P(2 \mid 1)P(3 \mid 2)$, $\psi(clq_2) = P(4 \mid 3, 2)$, $\psi(clq_3) = P(5 \mid 3)$.

The propagation phase proceeds in a similar way to belief propagation for trees, by propagating λ messages bottom-up and π messages top-down.

Bottom-Up Propagation

1. Calculate the λ message to send to the parent clique: $\lambda(C_i) = \sum_R \psi(C_i)$.
2. Update the potential of each clique with the λ messages of its sons: $\psi(C_j)' = \lambda(C_i)\psi(C_j)$.
3. Repeat the previous two steps until reaching the root clique.
4. When reaching the root node obtain $P'(C_r) = \psi(C_r)'$.

Top-Down Propagation

1. Calculate the π message to send to each child node i: $\pi(C_i) = \sum_{C_j - S_i} P'(C_j)$.
2. Update the potential of each clique when receiving the π message of its parent: $P'(C_i) = \pi(Ci)\psi(C_i)'$.
3. Repeat the previous two steps until reaching the leaf nodes in the junction tree.

When there is evidence, the potentials for each clique are updated based on the evidence, and the same propagation procedure is followed.

Finally, the marginal posterior probabilities of each variable are obtained from the clique potentials via marginalization: $P(X) = \sum_{C_i - X} P'(C_i)$.

There are two main variations on the junction tree algorithm, which are known as the Hugin [6] and Shenoy–Shafer [16] architectures. The description above is based on the Hugin architecture. The main differences between them are in the information they store, and in the way they compute the messages. These differences have implications in their computational complexity. In general, the Shafer–Shenoy architecture will require less space but more time.

7.3.2.4 Complexity Analysis

In the worst case, probabilistic inference for Bayesian networks is NP-Hard [1]. The time and space complexity is determined by what is known as the *tree-width*, and has to do with how close the structure of the network is to a tree. Thus, a tree-structured BN (maximum one parent per variable) has a tree-width of one. A polytree with at most k parents per node has a tree-width of k. In general, the tree-width is determined by how *dense* the topology of the network is, and this has affects: (i) the size of the largest factor in the variable elimination algorithm; (ii) the number of variables that need to be instantiated in the conditioning algorithm, and (iii) the size of the largest clique in the junction tree algorithm.

In practice, BNs tend to be sparse graphs, and in this case the junction tree techniques are very efficient even for models with hundreds of variables. In the case of complex networks, an alternative is to use approximate algorithms. These are described next.

7.3.3 Approximate Inference

7.3.3.1 Loopy Propagation

This is simply the application of the probability propagation algorithm for multi-connected networks. Although in this case the conditions for this algorithm are not satisfied, and it only provides an approximate solution for the inference problem, making it very efficient. Given that the BN is not singly connected, as the messages are propagated, these can *loop* through the network. Consequently, the propagation is repeated several times. The procedure is the following:

1. Initialize the λ and π values for all nodes to random values.
2. Repeat until convergence or a maximum number of iterations:
   a. Do probability propagation according to the algorithm for singly connected networks.
   b. Calculate the posterior probability for each variable.

The algorithm converges when the difference between the posterior probabilities for all variables of the current and previous iterations is below a certain threshold. It has been found empirically that for certain structures this algorithm converges to the true posterior probabilities; however, for other structures it does not converge [11].

An important application of loopy belief propagation is in "Turbo Codes" which is a popular error detection and correction scheme used in data communications.

7.3.3.2 Stochastic Simulation

Stochastic simulation algorithms consist in *simulating* the BN several times, where each simulation gives a sample value for all noninstantiated variables. These values are chosen randomly according to the conditional probability of each variable. This process is repeated N times, and the posterior probability of each variable is approximated in terms of the frequency of each value in the sample space. This gives an estimate of the posterior probability which depends on the number of samples; however, the computational cost is not affected by the complexity of the network. Next, we present two stochastic simulation algorithms for BNs: logic sampling and likelihood weighting.

Logic Sampling

Logic sampling is a basic stochastic simulation algorithm that generates samples according to the following procedure:

1. Generate sample values for the root nodes of the BN according to their prior probabilities. That is, a random value is generated for each root variable X, following a distribution according to $P(X)$.

Table 7.3 Samples generated using logic sampling for the BN in Fig. 7.13

variables	A	B	C	D	E
$sample_1$	T	F	F	F	T
$sample_2$	F	T	T	F	F
$sample_3$	T	F	F	T	F
$sample_4$	F	F	T	F	T
$sample_5$	T	F	T	T	F
$sample_6$	F	F	F	F	T
$sample_7$	F	T	T	T	F
$sample_8$	F	F	F	F	F
$sample_9$	F	F	F	T	F
$sample_{10}$	T	T	T	T	F

All variables are binary with two possible values, $True = T$ or $False = F$

2. Generate samples for the next *layer*, that is the sons of the already sampled nodes, according to their conditional probabilities, $P(Y \mid Pa(Y))$, where $Pa(Y)$ are the parents of Y.
3. Repeat (2) until all the leaf nodes are reached.

The previous procedure is repeated N times to generate N samples. The probability of each variable is estimated as the fraction of times (frequency) that a value occurs in the N samples, that is, $P(X = x_i) \sim No(x_i)/N$; where $No(x_i)$ is the number of times that $X = x_i$ in all the samples.

The direct application of the previous procedure gives an estimate of the marginal probabilities of all the variables when there is no evidence. If there is evidence (some variables are instantiated), all samples that are not consistent with the evidence are discarded and the posterior probabilities are estimated from the remaining samples.

For example, consider the BN in Fig. 7.13, and 10 samples generated by logic sampling. Assuming all variables are binary, the 10 samples generated are shown in Table 7.3.

If there is no evidence, then given these samples, the marginal probabilities are estimated as follows:

- $P(A = T) = 4/10 = 0.4$
- $P(B = T) = 3/10 = 0.3$
- $P(C = T) = 5/10 = 0.5$
- $P(D = T) = 5/10 = 0.5$
- $P(E = T) = 3/10 = 0.3$

The remaining probabilities are just the complement, $P(X = F) = 1 - P(X = T)$.

In the case where there is evidence with $D = T$, we eliminate all the samples where $D = F$, and estimate the posterior probabilities from the remaining five samples:

- $P(A = T \mid D = T) = 3/5 = 0.6$
- $P(B = T \mid D = T) = 2/5 = 0.4$
- $P(C = T \mid D = T) = 3/5 = 0.6$
- $P(E = T \mid D = T) = 1/5 = 0.2$

A disadvantage of logic sampling when evidence exists is that many samples have to be discarded; this implies that a larger number of samples are required to have a *good* estimate. An alternative algorithm that does not waste samples is presented below.

Likelihood Weighting

Likelihood weighting generates samples in the same way as logic sampling; however, when there is evidence the nonconsistent samples are not discarded. Instead, each sample is given a weight according to the weight of the evidence for this sample. Given a sample s and the evidence variables $\mathbf{E} = \{E_1, \ldots, E_m\}$, the weight of sample s is estimated as:

$$W(\mathbf{E} \mid s) = P(E_1)P(E_2) \ldots P(E_m) \tag{7.22}$$

where $P(E_i)$ is the probability of the evidence variable E_i for that sample.

The posterior probability for each variable X taking value x_i is estimated by dividing the sum of the weights $W_i(X = x_i)$ for each sample where $X = x_i$ by the total weight for all the samples:

$$P(X = x_i) \sim \sum_i W_i(X = xi) / \sum_i W_i \tag{7.23}$$

7.3.4 Most Probable Explanation

The *most probable explanation* (MPE) or *abduction* problem consists in determining the most probable values for a subset of variables (explanation subset) in a BN given some evidence. There are two variants of this problem, *total abduction* and *partial abduction*. In the total abduction problem, the explanation subset is the set of all noninstantiated variables; while in partial abduction, the explanation subset is a proper subset of the noninstantiated variables. In general, the MPE is not the same as the union of the most probable value for each individual variable in the explanation subset.

Consider the set of variables $\mathbf{X} = \{X_E, X_R, X_H\}$, where X_E is the subset of instantiated variables; then we can formalize the MPE problems as follows:

Total abduction: $ArgMax_{X_H, X_R} P(X_H, X_R \mid X_E)$.
Partial abduction: $ArgMax_{X_H} P(X_H \mid X_E)$.

One way to solve the MPE problem is based on a modified version of the variable elimination algorithm. For the case of total abduction, we substitute the summations by maximizations:

$$max_{X_H, X_R} P(X_H, X_R \mid X_E)$$

For partial abduction, we sum over the variables that are not in the explanation subset and maximize over the explanation subset:

$$max_{X_H} \sum_{X_R} P(X_H, X_R \mid X_E)$$

The MPE problem is computationally more complex than the single query inference.

7.3.5 Continuous Variables

Up to now, we have considered BNs with discrete multivalued variables. When dealing with continuos variables, one option is to discretize them; however, this could result in a loss of information (few intervals) or in an unnecessary increase in computational requirements (many intervals). Another alternative is to operate directly on the continuous distributions. Probabilistic inference techniques have been developed for some distribution families, in particular for Gaussian variables. Next, we describe the basic propagation algorithm for linear, Gaussian BNs [14].

The basic algorithm makes the following assumptions:

1. The structure of the network is a polytree.
2. All the sources of uncertainty are Gaussians and uncorrelated.
3. There is a linear relationship between each variable and its parents:

$$X = b_1 U_1 + b_2 U_2 + \cdots + b_n U_n + W_X$$

where U_i are parents of variable X, b_i are constant coefficients and W_X represents Gaussian noise with a zero mean.

The inference procedure is analogous to belief propagation in discrete BNs, but instead of propagating probabilities, it propagates means and standard deviations. In the case of Gaussian distributions, the marginal distributions of all the variables are also Gaussians. Thus, in general, the posterior probability of a variable can be written as:

$$P(X \mid E) = N(\mu_X, \sigma_X)$$

where μ_X and σ_X are the mean and standard deviation of X given the evidence E, respectively.

Next, we describe how to calculate the mean and standard deviation via the propagation algorithm. Each variable sends to its parent variable i:

$$\mu_i^- = (1/b_i)\left[\mu_\lambda - \sum_{k\neq i} b_k \mu_k^+\right] \tag{7.24}$$

$$\sigma_i^- = (1/b_i^2)\left[\sigma_\lambda - \sum_{k\neq i} b_k^2 \sigma_k^+\right] \tag{7.25}$$

Each variable sends to its child node j:

$$\mu_j^+ = \frac{\sum_{k\neq j} \mu_k^-/\sigma_k + \mu_\pi/\sigma_\pi]}{\sum_{k\neq} 1/\sigma_k^- + \mu_\pi/\sigma_\pi} \tag{7.26}$$

$$\sigma_j^+ = \left[\sum_{k\neq j} 1/\sigma_k^- + 1/\sigma_\pi\right]^{-1} \tag{7.27}$$

Each variable integrates the messages it receives from its sons and parents via the following equations:

$$\mu_\pi = \sum_i b_i \mu_i^+ \tag{7.28}$$

$$\sigma_\pi = \sum_i b_i^2 \sigma_i^+ \tag{7.29}$$

$$\mu_\lambda = \sigma_\lambda \sum_j \mu_j^-/\sigma_j^- \tag{7.30}$$

$$\sigma_\lambda = \left[\sum_j 1/\sigma_j^-\right]^{-1} \tag{7.31}$$

Finally, each variable obtains its mean and standard deviation by combining the information from its parent and children nodes:

$$\mu_X = \frac{\sigma_\pi \mu_\lambda + \sigma_\lambda \mu_\pi}{\sigma_\pi + \sigma_\lambda} \tag{7.32}$$

$$\sigma_X = \frac{\sigma_\pi \sigma_\lambda}{\sigma_\pi + \sigma_\lambda} \tag{7.33}$$

Propagation for other distributions is more difficult, as they do not have the same properties of the Gaussian; in particular, the product of Gaussians is also a Gaussian. An alternative for other types of distributions is to apply stochastic simulation techniques.

7.4 Applications

Bayesian networks have been applied in many domains, including medicine, industry, education, finance, biology, etc. To exemplify the application of BNs, in this chapter we will describe: (i) a technique for information validation and (ii) a methodology for system reliability analysis. In the following chapters, we will illustrate their application in other areas.

7.4.1 Information Validation

Many systems use information to make decisions; if this information is erroneous it could lead to nonoptimal decisions, and in some cases decisions made based on erroneous data could be dangerous. Consider, for example, an intensive care unit of a hospital in which sensors monitor the status of an operated patient so that the body temperature is kept beneath certain levels. Given that the sensors are working constantly, there is potential for them to produce erroneous readings. If this happens, two situations may arise:

- the temperature sensor indicates no changes in temperature even if it has increased to dangerous levels,
- the temperature sensor indicates a dangerous level even if it is normal.

The first situation may cause severe damage to the patient's health. The second situation may cause an emergency treatment of the patient that can also worsen his/her condition.

In many applications, there are different sources of information, i.e., sensors, which are not independent; the information from one source gives us clues about the other sources. If we can represent these dependencies between the different sources, then we can use it to detect possible errors and avoid erroneous decisions. This section presents an information validation algorithm based on Bayesian networks [4]. The algorithm starts by building a model of the dependencies between sources of information (variables) represented as a Bayesian network. Subsequently, the validation is done in two phases. In the first phase, potential faults are detected by comparing the actual value with the one predicted from the related variables via propagation in the Bayesian network. In the second phase, the real faults are isolated by constructing an additional Bayesian network based on the Markov blanket property.

7.4.1.1 Fault Detection

It is assumed that it is possible to build a probabilistic model relating all the variables in the application domain. Consider, for example, the network shown in Fig. 7.17 which represents the most basic function of a gas turbine.

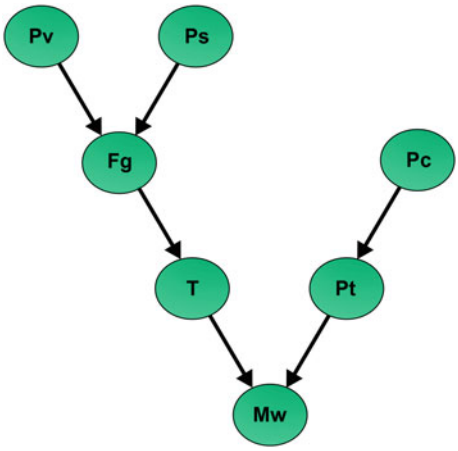


Fig. 7.17 A basic probabilistic model of a gas turbine. The mega watts generated in a gas turbine (node Mw) depends on the temperature (node T) and pressure in the turbine (node Pt). Temperature depends on the flow of gas (node Fg) and this flow depends on the valve of gas position (node Pv) and the gas fuel pressure supply (node Ps). The pressure for the turbine depends on the pressure at the output of the compressor (node Pc)

Suppose it is required to validate the temperature measurements in the turbine. By reading the values of the rest of the sensors, and applying probability propagation, it is possible to calculate a posterior probability distribution of the temperature given all the evidence, i.e., $P(T \mid Mw, P, Fg, Pc, Pv, Ps)$. Assuming that all the variables are discrete or discretized if continuous, by propagation we obtain probability distributions for each value of T. If the real observed value coincides with a *valid* value—that has a *high* probability, then the sensor is considered correct; otherwise, it is considered faulty.

This procedure is repeated for all the sensors in the model. However, if a validation of a single sensor is made using a faulty sensor, then a faulty validation can be expected. In the example above, what happens if T is validated using a faulty Mw sensor? How do we know which of the sensors is faulty? Thus, by applying this validation procedure, we may only detect a faulty condition, but we are not able to identify which is the real faulty sensor. This is called an *apparent fault*. An isolation stage is needed.

7.4.1.2 Fault Isolation

The isolation phase is based on the *Markov Blanket* (MB) property. For example, in the network of Fig. 7.17, the $MB(T) = \{Mw, Fg, Pt\}$, and the $MB(Pv) = \{Fg\}$. The set of nodes that constitute the MB of a sensor can be seen as a protection of the sensor against changes outside its MB. Additionally, we define the *Extended Markov Blanket* of a node X ($EMB(X)$) as the set of sensors formed by the sensor itself plus its MB. For example, $EMB(Fg) = \{Fg, Pv, Ps, T\}$.

Utilizing this property, if a fault exists in one of the sensors, it will be revealed in all of the sensors in its EMB. On the contrary, if a fault exists outside a sensors' EMB, it will not affect the estimation of that sensor. It can be said then, that the EMB of a sensor acts as its protection against others faults, and also protects others from its own failure. We utilize the EMB to create a *fault isolation* module that distinguishes the *real faults* from the apparent faults. The full theory is developed in [5].

After a cycle of basic validations of all sensors is completed, a set S of apparent faulty sensors is obtained. Thus, based on the comparison between S and the EMB of all sensors, the theory establishes the following situations:

1. If $S = \phi$ there are no faults.
2. If S is equal to the EMB of a sensor X, and there is no other EMB which is a subset of S, then there is a *single real fault* in X.
3. If S is equal to the EMB of a sensor X, and there are one or more EMBs which are subsets of S, then there is a real fault in X, and possibly, real faults in the sensors whose EMBs are subsets of S. In this case, there are possibly *multiple indistinguishable* real faults.
4. If S is equal to the union of several EMBs and the combination is unique, then there are *multiple distinguishable* real faults in all the sensors whose EMB are in S.
5. If none of the above cases is satisfied, then there are multiple faults but they cannot be distinguished. All the variables whose EMBs are subsets of S could have a real fault.

For example, considering the Bayesian network model in Fig. 7.17, some of the following situations may occur (among others):

- $S = \{T, Pt, Mw\}$, which corresponds to case 2, and confirms a single real fault in Mw,
- $S = \{T, Pc, Pt, Mw\}$, which corresponds to case 3, and as such, there is a real fault in Pt and possibly in Pv and Mw,
- $S = \{Pv, Ps, Fg\}$, which corresponds to case 4, and as such, there are real faults in Pv and Ps.

The isolation of a real fault is carried out in the following manner. Based on the EMB property described above, there will be a real fault in sensor X if an apparent fault is detected in its entire EMB. Also, we can say that an apparent fault will be revealed if a real fault exists in any sensor of its EMB. With these facts, we define the isolation network formed by two levels. The root nodes represent the real faults, where there is one per sensor or variable. The lower level is formed by one node representing the apparent fault for each variable. Each node is a binary variable with two values: {correct, faulty}. Notice that the arcs are defined by the EMB of each variable. Figure 7.18 shows the isolation network for the detection network of Fig. 7.17. For instance, the apparent fault node corresponding to variable Mw (node A_{mw}) is connected with nodes R_{mw}, R_T and R_P, which represent the real faults of the EMB nodes of Mw. At the same time, node R_{mw} is connected with all the apparent faults that this real fault causes, i.e., to nodes A_{mw}, A_T, and A_P. Fault isolation is

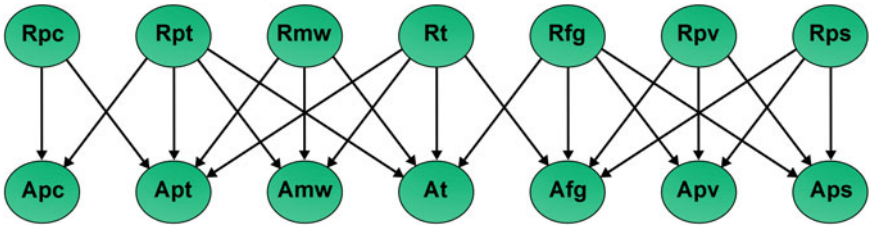


Fig. 7.18 Isolation network of the example in Fig. 7.17. The upper level nodes represent the real faults and the lower level nodes the apparent faults

Algorithm 7.1 Function isolate

Require: A sensor n and the state of sensor n.
1: Assign a value (instantiate) to the apparent fault node corresponding to n
2: Propagate probabilities and obtain a posterior probability of all nodes *Real fault*
3: Update vector $P_f(sensors)$

carried out by the isolation procedure described in Algorithm 7.1. The faulty sensors will be determined by the real fault nodes with "high" probability.

7.4.2 Reliability Analysis

In the reliability analysis of a complex system, a common approach is to divide the system into smaller elements, units, subsystems, or components. The main assumption is that every entity has two states: success and failure. This subdivision generates a "block diagram" that is similar to the description of the system in operation. For each element, the *failure rate* is specified, and based on these, the reliability of the complete system is obtained.

Traditionally, fault trees are used for reliability analysis; however, this technique has its limitations, as it assumes independent events, thus it is difficult to model dependency between events or faults. Dependent events can be found in reliability analysis in the following cases: (i) common causes—condition or event which provokes multiple elemental failures; (ii) mutually exclusive primary events—the occurrence of one basic event precludes another; (iii) standby redundancies—when an operating component fails, a standby component is put into operation, and the redundant configuration continues to function; and (iv) components supporting loads—failure of one component increases the load supported by the other components. Using Bayesian networks, we can explicitly represent dependencies between failures, and in this way model complex systems that are difficult for traditional techniques [17].

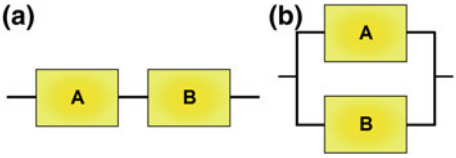


Fig. 7.19 Reliability block diagrams for the basic reliability structures with two components: **a** serial, **b** parallel

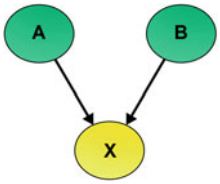


Fig. 7.20 Bayesian network structure for the two basic reliability block diagrams from Fig. 7.19

Table 7.4 Conditional probability table, $P(X \mid A, B)$, for two components with a serial structure

X	A, B	$A, \neg B$	$\neg A, B$	$\neg A, \neg B$
Success	1	0	0	0
Failure	0	1	1	1

A means that the component A is operational and $\neg A$ that it has failed

7.4.2.1 Reliability Modeling with Bayesian Networks

Reliability analysis starts by representing the structure of the system in terms of a reliability block diagram. In this representation, there are two basic structures: serial and parallel components (see Fig. 7.19). A serial structure implies that the two components should operate correctly for the system to function; or in other words, if one fails the entire system fails (this corresponds to an *AND* gate in fault trees). In parallel structures, it is sufficient for one of the components to operate for the system to function (*OR* gate in fault tress).

We can represent the previous basic block diagrams with a Bayesian network as is depicted in Fig. 7.20. The structure is the same in both cases, the difference is the conditional probability matrix. The CPTs for both cases are depicted in Tables 7.4 and 7.5. In both cases, the prior probabilities of the basic components (A, B) will represent there failure rate. Thus, by applying probabilistic inference in the BN representation, we obtain the failure rate of the system, X.

The BN representation of the basic serial/parallel cases can be directly generalized to represent any block diagram that can be reduced to a set of serial and parallel combinations of components, which, in practice is the case for most systems. There are some structures that cannot be decomposed to a serial/parallel combination, such as a *bridge*. However, it is also possible to model these cases using BNs [17].

Table 7.5 Conditional probability table, $P(X \mid A, B)$, for two components in parallel

X	A, B	$A, \neg B$	$\neg A, B$	$\neg A, \neg B$
Success	1	1	1	0
Failure	0	0	0	1

A means that the component A is operational and $\neg A$ that it has failed

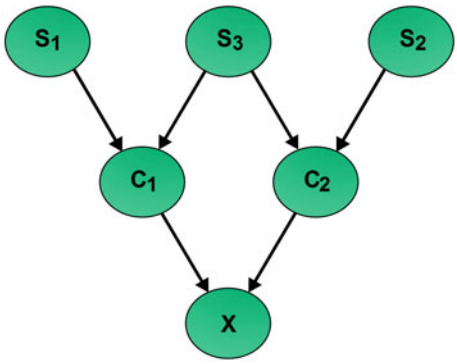


Fig. 7.21 Bayesian network structure for the system with a common cause failures

7.4.2.2 Modeling Dependent Failures

The main advantage of reliability modeling with BNs is that we can model dependent failures. We will illustrate this for the case of a system with common cause failures.

Suppose that a system has two components that are affected by three possible failure sources. Source S_1 affects component C_1, source S_2 affects component C_2, and source S_3 affects both components (common cause). For instance, the system could be a power plant with two subsystems; each subsystem has elements that can fail, but an earthquake can make both fail. A Bayesian network model for this example of dependent failures is depicted in Fig. 7.21. In this model, the CPT for all three nonroot nodes (C_1, C_2, X) is equivalent to that of a serial component combination. X represents the failure rate of the system, which can be obtained by probability propagation given the failure rates for the three failure sources.

7.5 Additional Reading

An introduction to Bayesian networks is given in the classic book by Judea Pearl [14]. Other general books on BNs are [7, 12]. A more recent account with emphasis on modeling and inference is given by [2]; it includes a complexity analysis for the different inference techniques. Other books with more emphasis on applications are [8, 15]. An overview of canonical models is presented in [3]. The junction tree algorithm was initially introduced by [9], and the two main architectures are described in [6, 16]. An analysis of loopy propagation can be seen in [11]. Inference for

continuous Gaussian variables was introduced in [14], and a more general approach based on truncated exponentials is presented in [10].

7.6 Exercises

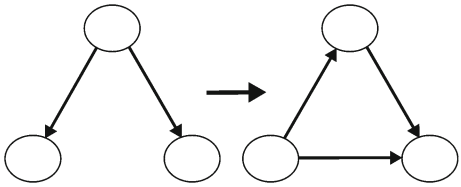
1. For the BN in Fig. 7.2 determine: (a) the contour of each variable, (b) the Markov blanket of each variable, (c) all the conditional independence relations implied by the network structure.
2. Deduce some of the independence relations in the previous problem using the independence axioms.
3. Complete the CPTs for the BN in Fig. 7.5 assuming all the variables are binary.
4. Investigate the Noisy AND model and obtain the CPT for a variable with three causes with inhibition probabilities equal to 0.05, 0.1, and 0.2, respectively.
5. Consider the belief propagation example in Sect. 7.3.1, obtain the posterior probabilities of all the variables via belief propagation considering that the only evidence is $C = true$.
6. Repeat the previous problem using the variable elimination procedure.
7. Estimate the posterior probabilities of the example in Sect. 7.3.1 under the same conditions as the previous two problems ($C = true$) using the logic sampling method for different numbers of samples (10, 20, …) and compare the results using exact inference.
8. For the BN in Fig. 7.18: (a) moralize the graph, (b) triangulate the graph, (c) determine the cliques and obtain a junction tree, (d) obtain the sets C, S and R for each clique according to the junction tree algorithm.
9. *** Develop a program based on the Bayes ball procedure to illustrate D-Separation. Given a BN structure, the user selects two nodes and a separation subset. The program should find all the trajectories between the two nodes, and then determines if these are independent given the separation subset by applying the Bayes ball procedure, illustrating graphically if the ball goes through a trajectory or is blocked.
10. *** Develop a general program for the belief propagation algorithm for polytrees considering discrete variables. Develop a parallel version of the previous program, establishing how the processors are assigned for an efficient parallelization of the algorithm. Extend the previous programs for loopy belief propagation.

References

1. Cooper, G.F.: The computational complexity of probabilistic inference using Bayesian networks. Artif. Intell. **42**, 393–405 (1990)
2. Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press, New York (2009)

3. Díez, F.J., Druzdzel, M.J.: Canonical probabilistic models for knowledge engineering. Technical Report CISIAD-06-01. Universidad Nacional de Educación a Distancia, Spain (2007)
4. Ibargüengoytia, P.H., Sucar, L.E., Vadera, S.: A probabilistic model for sensor validation. In: Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence UAI-96, pp. 332–339. Morgan Kaufmann Publishers Inc. (1996)
5. Ibargüengoytia, P.H., Vadera, S., Sucar, L.E.: A probabilistic model for information validation. Br. Comput. J. **49**(1), 113–126 (2006)
6. Jensen, F.V., Andersen, S.K.: Approximations in Bayesian belief universes for knowledge based systems. In: Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence UAI-90, pp. 162–169. Elsevier, New York (1990)
7. Jensen, F.V.: Bayesian Networks and Decision Graphs. Springer, New York (2001)
8. Korb, K.B., Nicholson, A.E.: Bayesian Artificial Intelligence, 2nd edn. CRC Press, Boca Raton (2010)
9. Lauritzen, S., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. R. Stat. Soc. Ser. B. **50**(2), 157–224 (1988)
10. Moral, S., Rumi, R., Salmerón, A.: Mixtures of truncated exponentials in hybrid Bayesian networks. Symb. Quant. Approaches Reason. Uncertain. **2143**, 156–167 (2001)
11. Murphy, K.P., Weiss, Y., Jordan, M.: Loopy belief propagation for approximate inference: an empirical study. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 467–475. Morgan Kaufmann Publishers Inc. (1999)
12. Neapolitan, R.E.: Probabilistic Reasoning in Expert Systems. Wiley, New York (1990)
13. Pearl, J.: Fusion, propagation and structuring in belief networks. Artif. Intell. **29**, 241–288 (1986)
14. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)
15. Pourret, O., Naim, P., Marcot, B. (eds.): Bayesian Belief Networks: A Practical Guide to Applications. Wiley, New Jersey (2008)
16. Shenoy, P., Shafer, G.: Axioms for probability and belief-function propagation. Uncertainty in Artificial Intelligence, vol. 4, pp. 169–198. Elsevier, New York (1990)
17. Torres-Toledano, J.G., Sucar, L.E.: Bayesian networks for reliability analysis of complex systems. In: Coelho, H. (ed.) IBERAMIA'98. Lecture Notes in Computer Science, vol. 1484, pp. 195–206. Springer, Berlin (1998)

Chapter 8
Bayesian Networks: Learning



8.1 Introduction

Learning a Bayesian network includes two aspects: learning the structure and learning the parameters. When the structure is known, parameter learning consists in estimating the conditional probability tables (CPTs) from data. For structure learning there are two main types of methods: (i) global methods based on search and score and (ii) local methods that use conditional independence tests. Next we describe both aspects, starting with parameter learning.

8.2 Parameter Learning

If we have *sufficient* and complete data for all the variables, and we assume the topology of the BN is known, parameter learning is straight forward. The CPT for each variable can be estimated from the data based on the frequency of each value (or combination of values) obtaining a *maximum likelihood* (ML) estimator of the parameters. For example, to estimate the CPT of B given it has two parents, A, C:

$$P(B_i \mid A_j, C_k) \sim NB_i A_j C_k / NA_j C_k \tag{8.1}$$

where $NB_i A_j C_k$ is the number of cases in the database in which $B = B_i$, $A = A_j$ and $C = C_k$, and $NA_j C_k$ is the total number of cases in which $A = A_j$ and $C = C_k$.

8.2.1 Smoothing

When we estimate probabilities from data, it can sometimes happen that a particular event never occurs in the dataset. This leads to the corresponding probability value being zero, implying an *impossible* case; if in the inference process this probability

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_8

is considered, it will also make the result zero. This situation occurs, in many cases, because there is insufficient data to have a robust estimate of the parameters, and not because it is an impossible event.

The previous situation can be avoided by using some type of *smoothing* for the probabilities, eliminating zero probability values. There are several smoothing techniques, one of the most common and simplest being *Laplacian* smoothing.

Laplacian smoothing consists in initializing the probabilities to a uniform distribution, and then updating these values based on the data. Consider a discrete variable, X, with k possible values. Initially, each probability will be set to $P(x_i) = 1/k$. Then, consider a dataset with N samples, in which the value x_i occurs m times; the estimate of its probability will be the following:

$$P(x_i) = (1 + m)/(k + N) \tag{8.2}$$

8.2.2 Parameter Uncertainty

If there is not sufficient data, a situation common in practice, we have uncertainty in the parameters. This uncertainty can be modeled using a second-order probability distribution, and could be propagated in the inference process so we have an estimate of the uncertainty in the resulting probabilities. For binary variables, the uncertainty in the parameters can be modeled using a Beta distribution:

$$\beta(a, b) = \frac{(a + b + 1)!}{a!b!} x^a (1 - x)^b \tag{8.3}$$

For multivalued variables, uncertainty in the parameters can be represented by an extension of the Beta known as the Dirichlet distribution.

For the binary case, the expected value of the Beta distribution is given by: $P(b_i) = a + 1/a + b + 2$, where a and b are the parameters of the Beta distribution. This representation could also be used when we have experts' estimates of the probabilities. The parameters of the Beta distribution can represent a measure of *confidence* in the expert's estimates, expressed by varying the term $a + b$ with the same probability value. For instance:

- Complete ignorance: $a = b = 0$.
- Low confidence: $a + b$ *small* (10).
- Medium confidence: $a + b$ *intermediate* (100).
- High confidence: $a + b$ large (1000).

This representation could be used to combine experts' estimations with data. For example, to approximate the probability value of a binary variable, b_i we can use:

$$P(b_i) = k + a + 1/n + a + b + 2 \tag{8.4}$$

where $a/a+b$ represents the expert's estimate and k/n is the probability obtained from the data (k is the number of times b_i occurs in n samples).

For example, let us assume an expert gives an estimate of 0.7 for a certain parameter, and that the experimental data provides 40 positive cases among 100 samples. The parameter estimation for different confidences assigned to the expert will be the following:

Low confidence ($a+b=10$): $P(b_i) = \frac{40+7+1}{100+10+2} = 0.43$

Medium confidence ($a+b=100$): $P(b_i) = \frac{40+70+1}{100+100+2} = 0.55$

High confidence ($a+b=1000$): $P(b_i) = \frac{40+700+1}{100+1000+2} = 0.67$

We observe that in the first case the estimate is dominated by the data, while in the third case the probability is closer to the expert's estimate; the second case provides a compromise.

8.2.3 Missing Data

Another common situation is to have incomplete data. There are two basic cases:

Missing values: In some registers there are missing values for one or more variables.

Hidden nodes: A variable or set of variables in the model for which there is no data at all.

For dealing with missing values, there are several alternatives:

1. Eliminate the registers with missing values.
2. Consider a special "unknown" value.
3. Substitute the missing value by the most common value (mode) of the variable.
4. Estimate the missing value based on the values of the other variables in the corresponding register.

The first and second alternatives are acceptable if there is sufficient data, otherwise we could be discarding useful information. The third alternative does not consider the other variables and as a result, it could bias the model. In general the best alternative is the fourth option. In this case, we first learn the parameters of the BN based on the complete registers, and then complete the data and re-estimate the parameters, applying the following process. For each register with missing values:

1. Instantiate all the known variables in the register.
2. Through probabilistic inference obtain the posterior probabilities of the missing variables.
3. Assign to each variable the value with highest posterior probability.
4. Add this completed register to the database and re-estimate the parameters.

An alternative to the previous process is that instead of assigning the value with the highest probability, we assign a *partial* case for each value of the variable proportional to the posterior probability.

For hidden nodes, the approach to estimate their parameters is based on the *Expectation–Maximization* (EM) technique.

8.2.3.1 Hidden Nodes: EM

The EM algorithm is a statistical technique used for parameter estimation when there are non-observable variables. It consists of two phases which are repeated iteratively:

E step: the missing data values are estimated based on the current parameters.
M step: the parameters are updated based on the estimated data.

The algorithm starts by initializing the missing parameters with random values.

Given a database with one or more hidden nodes, $H_1, H_2, \ldots, H_k$, the EM algorithm to estimate their CPTs is the following:

1. Obtain the CPTs for all the *complete* variables (the values of the variable and all its parents are in the database) based on an ML estimator.
2. Initialize the unknown parameters with random values.
3. Considering the actual parameters, estimate the values of the hidden nodes based on the known variables via probabilistic inference.
4. Use the estimated values for the hidden nodes to complete/update the database.
5. Re-estimate the parameters for the hidden nodes with the updated data.
6. Repeat 3–5 until converge (no significant changes in the parameters).

The EM algorithm optimizes the unknown parameters and gives a *local maximum* (the final estimates depend on the initialization).

8.2.3.2 Example

We now illustrate how to handle missing values and hidden variables using data from the *Golf* example (see Table 8.1). In this dataset, there are some missing values for the variable *Temperature* (registers 1 and 9), and there is no information about *Wind*, which is a hidden node. We first illustrate how to fill-in the missing values for temperature and then how to manage the hidden node.

Assume that we learn a naive Bayes classifier (an NBC is a particular type of BN) based on the available data (12 complete registers without the wind variable), considering *Play* as the class variable and the other variables as attributes. Then, based on this model, we can estimate the probability of temperature for the registers in which it is missing, via probabilistic inference using the values of the other variables in the corresponding registers as evidence. That is:

Register 1: $P(Temperature \mid sunny, high, N)$
Register 9: $P(Temperature \mid sunny, normal, P)$

Table 8.1 Data for the golf example with missing values for *Temperature* and a hidden variable, *Wind*

Outlook	Temperature 1	Humidity	Wind	Play
Sunny	xxx	High	–	N
Sunny	High	High	–	N
Overcast	High	High	–	P
Rainy	Medium	High	–	P
Rainy	Low	Normal	–	P
Rainy	Low	Normal	–	N
Overcast	Low	Normal	–	P
Sunny	Medium	High	–	N
Sunny	xxx	Normal	–	P
Rainy	Medium	Normal	–	P
Sunny	Medium	Normal	–	P
Overcast	Medium	High	–	P
Overcast	High	Normal	–	P
Rainy	Medium	High	–	N

Then we can select as the value of temperature the one with highest posterior probability, and fill-in the missing values, as shown in Table 8.2.

For the case of the hidden node, *Wind*, we cannot obtain the corresponding CPT from the NBC, $P(Wind \mid Play)$, as there are no values for wind. However, we can apply the EM procedure, where we first pose initial random parameters for the CPT, which could be, for example, a uniform distribution:

$$P(Wind \mid Play) = \begin{matrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{matrix}$$

Given this CPT we have a complete, initial model for the NBC, and can estimate the probability of wind for each register based on the values of the other variables in the register. By selecting the highest probability value for each register, we can fill-in the table, as depicted in Table 8.2. Based on this new data table, we re-estimate the parameters, and obtain a new CPT:

$$P(Wind \mid Play) = \begin{matrix} 0.60 & 0.44 \\ 0.40 & 0.56 \end{matrix}$$

This completes one cycle of the EM algorithm; the process is then repeated until all parameters in the CPT have *almost* no change from the previous iteration. At this point, the EM procedure has converged, and we have an estimate of the missing parameters of the BN.

Table 8.2 Data for the golf example after completing the missing values for *Temperature* and one iteration of the EM procedure to estimate the values of *Wind*

Outlook	Temperature 1	Humidity	Wind	Play
Sunny	Medium	High	No	N
Sunny	High	High	No	N
Overcast	High	High	No	P
Rainy	Medium	High	No	P
Rainy	Low	Normal	Yes	P
Rainy	Low	Normal	Yes	N
Overcast	Low	Normal	Yes	P
Sunny	Medium	High	No	N
Sunny	Medium	Normal	No	P
Rainy	Medium	Normal	No	P
Sunny	Medium	Normal	Yes	P
Overcast	Medium	High	Yes	P
Overcast	High	Normal	Yes	P
Rainy	Medium	High	Yes	N

8.2.4 Discretization

Usually Bayesian networks consider discrete or nominal values. Although there are some developments for continuous variables, these are restricted to certain distributions, in particular Gaussian variables and linear relations. An alternative to include continuous variables in BNs is to discretize them. Discretization methods can be (i) unsupervised and (ii) supervised.

8.2.4.1 Unsupervised Discretization

Unsupervised techniques do not consider the task for which the model is going to be used (e.g., classification), such that the intervals for each variable are determined independently. The two main types of unsupervised discretization approaches are: equal width and equal data.

Equal width consists in dividing the range of a variable, $[Xmin; Xmax]$, into k equal bins; such that each bin has a size of $[Xmin; Xmax]/k$. The number of intervals k is usually set by the user.

Equal data divides the range of the variable in k intervals, such that each interval includes the same number of data points from the training data. In other words, if there are n data points, each interval will contain n/k data points; this means that the intervals will not necessarily have the same width.

8.2.4.2 Supervised Discretization

Supervised discretization considers the task to be performed with the model, such that the variables are discretized to optimize this task, for instance classification accuracy. If we consider a BN for classification, i.e. a Bayesian classifier, then the supervised approach can be directly applied. Assuming continuous attribute variables, these are discretized according to the class values. This can be posed as an optimization problem.

Consider the attribute variable X with range $[Xmin; Xmax]$ and a class variable C with m values $c_1, c_2, \ldots, c_m$. Given n training samples, so that each one has a value for C and X, the problem is to determine the *optimal* partition of X such that the classifier precision is maximized. This a combinatorial problem that is computationally complex, and can be solved using a search process as follows:

1. Generate all potential divisions in X which correspond to a value in $[Xmin; Xmax]$ where there is a change in the class value.
2. Based on the potential division points generate an initial set of n intervals.
3. Test the classification accuracy of the Bayesian classifier (usually on a different set of data known as a validation set) according to the current discretization.
4. Modify the discretization by partitioning an interval or joining two intervals.
5. Repeat (3) and (4) until the accuracy of the classifier cannot be improved or some other termination criteria occurs.

Different search approaches can be used, including basic ones such as *hill-climbing* or more sophisticated methods like *genetic algorithms*.

The previous algorithm does not apply for the general case of a Bayesian network which can be used to predict different variables based on different evidence variables. In this case, there is a supervised method [6] that discretizes continuous attributes while it learns the structure of the BN. The method is based on the *Minimum Description Length* (MDL) principle—described in Sect. 8.3.3. For each continuous variable, the number of intervals is determined according to its neighbors in the network. The objective is to minimize the MDL (a compromise between the precision and complexity of the model), using a search and test approach analogous to the process for Bayesian classifiers. This is repeated iteratively for all continuous variables in the network.

8.3 Structure Learning

Structure learning consists in obtaining the topology of the BN from the data. This is a complex problem because: (i) the number of possible structures is *huge* even with a few variables (it is super-exponential on the number of variables; for example, for 10 variables the number of possible DAGs is in the order of 4×10^{18}), and (ii) a very large database is required to obtain good estimates of the statistical measures on which all methods depend.

For the particular case of a tree structure, there is a method that guarantees the *best* tree. For the general case several methods have been proposed, which can be divided into two main classes:

1. Global methods: these [4, 5] perform a heuristic search over the space of network structures, starting from some initial structure, and generating a variation of the structure at each step. The *best* structure is selected based on a score that measures how well the model represents the data. Common scores are BIC [4] and MDL [5].
2. Local methods: these are based on evaluating the (in)dependence relations between subsets of variables given the data, to sequentially obtain the structure of the network. The most well-known variant of this approach is the PC algorithm [10].

Both classes of methods obtain similar results with *enough* data. Local methods tend to be more sensitive when there are few data samples, and global methods tend to be more computationally complex.

Next we review the tree learning algorithm developed by Chow and Liu [1] and its extension to polytrees. Then we present the techniques for learning a general structure.

8.3.1 Tree Learning

Chow and Liu [1] developed a method for approximating any multivariable probability distribution as a product of second-order distributions, which is the basis for learning tree-structured BNs. The joint probability of n random variables can be approximated as:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_{j(i)}) \tag{8.5}$$

where $X_{j(i)}$ is the parent of X_i in the tree.

The problem consists in obtaining the *best* tree, that is, the tree structure that best approximates the real distribution. A measure of how close the approximation is based on the information difference between the real distribution (P) and the tree approximation (P^*) is as follows:

$$DI(P, P^*) = \sum_{X} P(X) log(P(X)/P^*(X)) \tag{8.6}$$

Thus, now the problem consists in finding the tree that minimizes DI.

The mutual information between any pair of variables is defined as:

$$I(X_i, X_j) = \sum_{X_i, X_j} P(X_i, X_j) log(P(X_i, X_j)/P(X_i)P(X_j)) \tag{8.7}$$

Given a tree-structured BN with variables $X_1, X_2, \ldots, X_n$, we define its *weight*, W, as the sum of the mutual information of the arcs (pairs of variable) that constitute the tree:

$$W(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n-1} I(X_i, X_j) \tag{8.8}$$

where X_j is the parent of X_i in the tree (a tree with n nodes has $n - 1$ arcs).

It can be shown [1] that minimizing *DI* is equivalent to maximizing W. Therefore, obtaining the optimal tree is equivalent to finding the *maximum weight spanning tree*, using the following algorithm:

1. Obtain the mutual information (I) between all pairs of variables (for n variables, there are $n(n-1)/2$ pairs).
2. Order the mutual information values in descending order.
3. Select the pair with maximum I and connect the two variables with an arc, this constitutes the initial tree.
4. Add the pair with the next highest I to the tree, while they do not make a cycle; otherwise skip it and continue with the following pair.
5. Repeat 4 until all the variables are in the tree ($n - 1$ arcs).

This algorithm obtains the *skeleton* of the tree; that is, it does not provide the direction of the arcs in the BN. The directions of the links have to be obtained using external semantics or using higher order dependency tests (see below).

To illustrate the tree learning method consider the classic *golf* example with 5 variables: *play, outlook, humidity, temperature, wind*. Given some data, we obtain the mutual information shown in Table 8.3.

Table 8.3 Mutual information in descending order for the golf example

No.	Var 1	Var 2	Mutual information
1	Temp.	Outlook	0.2856
2	Play	Outlook	0.0743
3	Play	Humidity	0.0456
4	Play	Wind	0.0074
5	Humidity	Outlook	0.0060
6	Wind	Temp.	0.0052
7	Wind	Outlook	0.0017
8	Play	Temp.	0.0003
9	Humidity	Temp.	0
10	Wind	Humidity	0

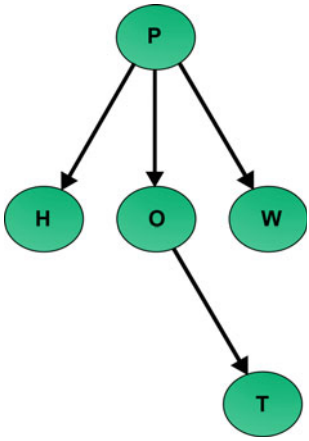


Fig. 8.1 Tree structure obtained for the golf example (P is play, O is outlook, H is humidity, T is temperature, and W is wind). Arc directions are set arbitrarily

In this case, we select the first four pairs (arcs) and obtain the tree in Fig. 8.1, where the directions were assigned arbitrarily.

8.3.2 Learning a Polytree

Rebane and Pearl [9] developed a method that can be used to direct the arcs in the skeleton, and in general, learn a *polytree* BN. The algorithm is based on independence tests for variable triplets, and in this way it can distinguish *convergent* substructures; once one or more substructures of this type are detected in the skeleton, it can direct additional arcs by applying the independence tests to neighboring nodes. However, there is no guarantee for obtaining the direction for all the arcs in the tree. This same idea is used in the PC algorithm for learning general structures.

The algorithm begins with the skeleton (undirected structure) obtained with the Chow and Liu algorithm. Subsequently, the direction of the arcs is learned using independence tests for variable triplets. Given three variables, there are three possibilities:

1. Sequential arcs: $X \rightarrow Y \rightarrow Z$.
2. Divergent arcs: $X \leftarrow Y \rightarrow Z$.
3. Convergent arcs: $X \rightarrow Y \leftarrow Z$.

The first two cases are indistinguishable under statistical independence testing; that is, they are equivalent. In both cases, X and Z are independent given Y. However, the third case is different, since X and Z are NOT independent given Y. Consequently, this case can be used to determine the directions of the two arcs that connect these three variables; additionally, we can apply this knowledge to learn the directions of other arcs using independence tests. With this in mind, the following algorithm can be used for learning polytrees:

1. Obtain the skeleton using the Chow and Liu algorithm.
2. Iterate over the network until a convergent variable triplet is found. We will call the variable to which the arcs converge a *multi-parent node*.
3. Starting with a multi-parent node, determine the directions of other arcs using independence tests for variable triplets. Continue this procedure until it is no longer possible (causal base).
4. Repeat 2–3 until no other directions can be determined.
5. If any arcs are left undirected, use the external semantics to infer their directions.

To illustrate this algorithm, let us consider the golf example again, with the obtained skeleton (undirected structure). Suppose the variable triplet H, P, W falls in the convergent case. Then, the arcs will be directed such that H points to P and W points to P. Subsequently, the dependence between H and W is measured with respect to O given P. If H and W are independent from O given P then there will be an arc that points from P to O. Finally, the dependence relation between P and T given O is tested, and if they are again found to be independent, then the arc points from O to T. Figure 8.2 shows the resulting structure.

8.3.3 Search and Score Techniques

The previous methods are restricted to tree and polytree structures; in this and the following section we will cover general structure learning techniques, starting with global approaches.

Global methods search for the *best* structure based on a global metric. That is, different structures are generated and these are evaluated with respect to the data using some scoring method. There are different variants of these methods, all of which basically depend on two aspects: (i) a fitness measure between the structure and the data, and (ii) a method for searching for the best structure.

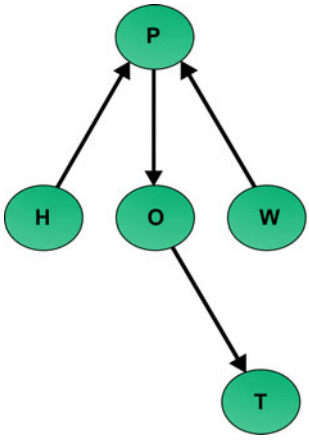


Fig. 8.2 A polytree obtained for the golf example using the Rebane and Pearl algorithm

8.3.3.1 Scoring Functions

There are several possible fitness measures or scoring functions. Two desirable properties for scoring functions are [3]:

Decomposability: a scoring function is decomposable if the value assigned to each structure can be expressed as a sum (in the logarithmic space) of local values that depend only on each node and its parents. This is important for efficiency reasons during the search process; given this property when a local change is made to the structure, only a part of the score has to be re-evaluated.

Score Equivalence: a scoring function S is score equivalent if it assigns the same value to all DAGs that are represented by the same essential graph. In this way, the result of evaluating an equivalence class will be the same regardless of the DAG that is selected from this class. The structures of two BNs correspond to the same essential graph if they are equivalent in terms of the independence relations they represent.

Next we describe some common scoring functions, including: the maximum likelihood (ML), the Bayesian information criterion (BIC), the Bayesian score (BD), and the minimum description length (MDL) criterion.

The maximum likelihood score selects the structure that maximizes the probability of the data, D, given the structure, G:

$$G^* = ArgMax_G[P(D \mid \Theta_G, G_i)] \tag{8.9}$$

where G_i is the candidate structure and Θ_G the corresponding vector of parameters (probability of each variable given its parents according to the structure).

The direct application of the ML score might result in a highly complex network, which usually implies overfitting the data (poor generalization) and also makes inference more complex. Therefore, a way to penalize complex models is required.

A commonly used scoring function that includes a penalty term is the Bayesian Information Criterion or BIC defined as:

$$BIC = logP(D \mid \Theta_G, G_i) - \frac{d}{2} logN \tag{8.10}$$

where d is the number of parameters in the BN and N the number of cases in the data. An advantage of this metric is that it does not require a prior probability specification and it is related to the MDL measure, compromising between the precision and complexity of the model. However, given the high penalty on the complexity of the model, it tends to choose structures that are too simple.

Bayesian scores

An alternative metric is obtained by following a Bayesian approach, obtaining the posterior probability of the structure given the data with the Bayes rule:

$$P(G_i \mid D) = P(G_i)P(D \mid G_i)/P(D) \tag{8.11}$$

Given that P(D) is a constant that does not depend on the structure, it can be discarded from the metric to obtain de Bayesian or BD score:

$$BD = P(G_i)P(D \mid G_i) \tag{8.12}$$

$P(G_i)$ is the prior probability of the model. This can be specified by an expert or defined such that simpler structures are preferred; or just set to a uniform distribution.

The BDe score is a variation of the BD score which makes the following assumptions: (i) the parameters are independent and have a prior Dirichlet distribution, (ii) equivalent structures have the same score, (iii) the data samples are independent and identically distributed (iid). Under these assumptions the *virtual counts* required to compute the score can be estimated as:

$$N_{ijk} = P(X_i = k, Pa(X_i) = j \mid G_i, \Theta_G) \times N' \tag{8.13}$$

This is the estimated count of a certain *configuration*: $X_i = k$ given $Pa(X_i) = j$; N' is the equivalent sample size.

By assuming that the hyperparameters of the priors are one, we can further simplify the calculation of the Bayesian score, and obtain what is known as the K2 metric.[1] This score is decomposable and it is calculated for each variable X_i given its parents $Pa(X_i)$:

$$S_i = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}! \tag{8.14}$$

where r_i is the number of values of X_i, q_i is the number of possible configurations for the parents of X_i, α_{ijk} is the number of cases in the database where $X_i = k$ and $Pa(X_i) = j$, and N_{ij} is the number of cases in the database where $Pa(X_i) = j$.

This metric provides a practical alternative for evaluating a BN. Another common alternative that is based on the MDL principle is described next.

MDL

The MDL measure makes a compromise between accuracy and model complexity. Accuracy is estimated by measuring the mutual information between the attributes and the class. Model complexity is evaluated by counting the number of parameters. A constant, α within [0, 1], is used to balance the weight of each aspect, that is, accuracy against complexity. The fitness measure is given by the following equation:

$$MC = \alpha(W/Wmax) + (1 - \alpha)(1 - L/Lmax) \tag{8.15}$$

where W represents the accuracy of the model, and L the complexity. $Wmax$ and $Lmax$ represent the maximum accuracy and complexity, respectively. To determine the maximums, usually an upper bound is set on the number of parents each node is

[1] K2 is an algorithm for learning BNs described below.

allowed to have. A value of $\alpha = 0.5$ gives equal importance to the model complexity and accuracy, while a value near 0 gives more importance to the complexity, and a value near 1 more importance to accuracy.

Complexity is given by the number of parameters required for representing the model, which can be measured with the following equation:

$$L = S_i[k_i log_2 n + d(S_i - 1)F_i] \tag{8.16}$$

where n is the number of nodes, k is the number of parents per node, S_i is the average number of values per variable, F_i is the average number of values per parent variable, and d the number of bits per parameter.

The accuracy can be estimated based on the 'weight' of each node; this is analogous to the weights in the methodology for learning trees. In this case, the weight of each node, X_i, is estimated based on its mutual information with its parents, $Pa(X_i)$:

$$w(X_i, Pa(X_i)) = \sum_{xi} P(X_i, Pa(X_i)) log[P(X_i, Pa(X_i))/P(X_i)P(Pa(X_i))] \tag{8.17}$$

and the weight (accuracy) total is given by the sum of the weights for each node:

$$W = \sum_i w(X_i, Pa(X_i)) \tag{8.18}$$

8.3.3.2 Search Algorithms

Once a fitness measure for the structure has been established, we need to establish a method for choosing the 'best' structure among the possible options. Since the number of possible structures is exponential on the number of variables, it is impossible to evaluate every structure. To limit the number of structures that are evaluated, a heuristic search is carried out. Several different search methods can be applied. One common strategy is to use a *hill climbing* approach, where we begin with a simple tree structure that is improved until we obtain the 'best' structure. A basic greedy algorithm to search for the best structure is the following:

1. Generate an initial structure-tree.
2. Calculate the fitness measure of the initial structure.
3. Add/invert an arc from the current structure.
4. Calculate the fitness measure of the new structure.
5. If the fitness improves, keep the change; if not, return to the previous structure.
6. Repeat 3–5 until no further improvements exist.

The previous algorithm is not guaranteed to find the optimum structure, since it is possible to reach only a local maximum. Figure 8.3 illustrates the search procedure for the golf example, starting with a tree structure that is improved until the final

structure is obtained. Other search methods, such as genetic algorithms, simulated annealing, bidirectional searches, etc., can also be applied to obtain the best structure.

An alternative to reduce the number of potential structures to be evaluated, is to set an ordering on the variables, known as a *causal ordering*. Given this ordering, the arcs in the network are restricted to follow this order; that is, there could be NO arc from V_j to V_i if $j > i$ according to the ordering. The K2 algorithm [4] takes advantage of this, providing and efficient and popular method for learning BNs.

8.3.3.3 The K2 Algorithm

Given a causal ordering for all the variables, learning the best structure is equivalent to selecting the best set of parents for each node independently. Initially, each variables has no parents. Then, the K2 algorithm incrementally adds parents to each node, as long as it increases the global score. When adding parents to any node does not increase the score, the search stops. Also, given a causal ordering it guarantees that there are not cycles in the graph.

Algorithm 8.1 provides a summary of the K2 procedure. The inputs to the algorithm are the set of n variables with a causal ordering, $X_1, X_2, \ldots X_n$, a database D containing m cases, and, usually, a restriction on the maximum number of parents for each variable, u. The output is the set of parents, $Pa(X_i)$, for each variable, which defines the structure of the network. Starting from the first variable according to the ordering, the algorithm tests all possible parents of a variable that have not been added, and includes the one that makes the maximum increment in the score of the network. This is repeated until there is no additional parent that increases the score; for every node on the network.

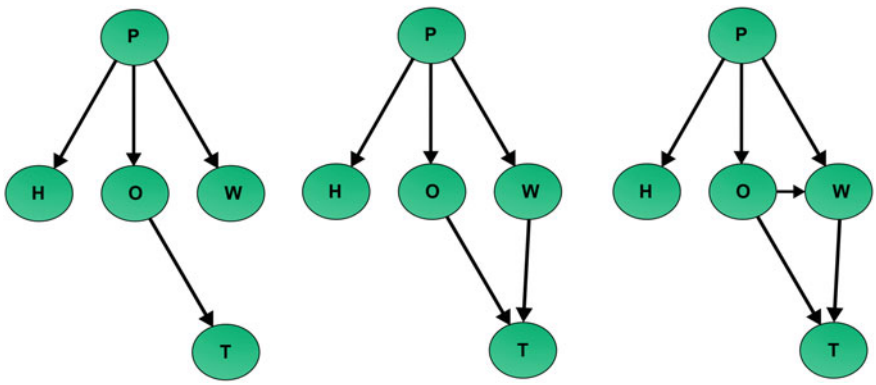


Fig. 8.3 A few steps in the procedure for learning the structure of the "golf" example, beginning with a tree structure (*left*) until the final structure (*right*) is obtained

Algorithm 8.1 The K2 Algorithm

Require: Set of variables X with a causal ordering, scoring function S, and maximum parents u
Ensure: Set of parents for each variable, $Pa(X_i)$

```
for i = 1 to n do
  oldScore = S(i, Pa(X_i))
  incrementScore = true
  Pa(X_i) = ∅
  while incrementScore and |Pa(X_i)| < u do
    let Z be the node in Predecessors(X_i) − Pa(X_i) that maximizes S
    newScore = S(i, Pa(X_i) ∪ Z)
    if newScore > oldScore then
      oldScore = newScore
      Pa(X_i) = Pa(X_i) ∪ Z
    else
      incrementScore = false
    end if
  end while
end for
return Pa(X_1), Pa(X_2) ... Pa(X_n)
```

8.3.4 Independence Tests Techniques

The other class of structure learning techniques use a *local* approach instead of the *global* one used by the score and search techniques. The basic idea is to apply independence tests to sets of variables to recover the structure of the BN. An example of this type of techniques is the Chow and Liu algorithm for trees. Next we present a method for learning general structures, the PC algorithm.

8.3.4.1 The PC Algorithm

The PC algorithm [10] first recovers the skeleton (underlying undirected graph) of the BN, and then it determines the orientation of the edges.

To determine the skeleton, it starts from a fully connected undirected graph, and determines the conditional independence of each pair of variables given some subset of the other variables. For this it assumes that there is a procedure that can determine if two variables, X, Y, are independent given a subset of variables, $\mathbf{S}$, that is, $I(X, Y \mid \mathbf{S})$. An alternative for this procedure is the conditional cross entropy measure. If this measure is below a threshold value set according to a certain confidence level, the edge between the pair of variables is eliminated. These tests are iterated for all pairs of variables in the graph.

In the second phase the direction of the edges are set based on conditional independence tests between variable triplets. It proceeds by looking for substructures in the graph of the form $X - Z - Y$ such that there is no edge $X - Y$. If X, Y are not independent given Z, it orients the edges creating a V-structure $X \rightarrow Z \leftarrow Y$. Once

all the V-structures are found, it attempts to orient the other edges based on independence tests and avoiding cycles. Algorithm 8.2 summarizes the basic procedure.[2]

Algorithm 8.2 The PC algorithm.

Require: Set of variables **X**, Independence test I
Ensure: Directed Acyclic Graph G
Initialize a complete undirected graph G'
i=0
repeat
  for $X \in \mathbf{X}$ **do**
    for $Y \in ADJ(X)$ **do**
      for $S \subseteq ADJ(X) - \{Y\}, \mid S \mid = i$ **do**
        if $I(X, Y \mid S)$ **then**
          Remove the edge $X - Y$ from G'
        end if
      end for
    end for
  end for
  i=i + 1
until $\mid ADJ(X) \mid \leq i, \forall X$
Orient edges in G'
Return G

If the set of independencies are faithful to a graph[3] and the independence tests are perfect, the algorithm produces a graph equivalent to the original one; that is, the BN structure that generated the data.

The independence test techniques rely on having *enough* data for obtaining good estimates from the independence tests. Search and score algorithms are more robust with respect to the size of the dataset, however their performance is also affected by the size and quality of the available data. An alternative for when there is not *sufficient* data, is to combine expert knowledge and data.

8.4 Combining Expert Knowledge and Data

When domain expertise is available, this can be combined with learning algorithms to improve the model. In the case of parameter learning, we can combine data and expert estimates based on the Beta or Dirichlet distributions as described in Sect. 8.2.

[2] $ADJ(X)$ is the set of nodes adjacent to X in the graph.

[3] The *Faithfulness Condition* can be thought of as the assumption that conditional independence relations are due to causal structure rather than to accidents in parameter values [10].

For structure learning, there are two basic approaches to combine expert knowledge and data:

- Use expert knowledge as *restrictions* to reduce the search space for the learning algorithm.
- Start from a structure proposed by an expert and use data to validate and improve this structure.

There are several ways to use expert knowledge to aid the structure learning algorithm, such as:

1. Define an ordering for the variables (causal order), such that there could be an arc from X_i to X_j only if X_j is after X_i according to the specified ordering.
2. Define restrictions in terms of directed arcs that must exist between two variables, i.e., $X_i \rightarrow X_j$.
3. Define restrictions in terms of an arc between two variables that could be directed either way.
4. Define restrictions in terms of pairs of variables that are not directly related, that is, there must be no arc between X_i and X_j.
5. Combinations of the previous restrictions.

Several variants of both types of techniques, search and score and independence tests, incorporate the previous restrictions.

In the case of the second approach, an example was presented in Chap. 4, with the structural improvement algorithm. This technique starts from a naive Bayes structure which is improved by eliminating, joining or inserting variables. This idea can be extended to general BN structures, in particular for tree-structured BNs.

8.5 Applications

There are many domains in which learning Bayesian networks has been applied to get a better understanding of the domain or make predictions based on partial observations; for example medicine, finance, industry and the environment, among others. Next we present an example for modeling the air pollution in Mexico City.

8.5.1 Air Pollution Model for Mexico City

Air quality in Mexico City is a major problem. There, air pollution is one of the highest in the world, with a high average of daily emissions of several primary pollutants, such as hydrocarbons, nitrogen oxides, carbon monoxide, and others. The pollution is due primarily to transportation and industrial emissions. When the primary pollutants are exposed to sunshine, they undergo chemical reactions and yield a variety of secondary pollutants, ozone being the most important. Besides the

health problems it may cause, ozone is considered an indicator of the air quality in urban areas.

The air quality is monitored in 25 stations within Mexico City, with five of these being the most complete. Nine variables are measured in each of the five main stations, including: wind direction and velocity, temperature, relative humidity, sulfur dioxide, carbon monoxide, nitrogen dioxide and ozone. These are measured every minute 24 hours a day, and are averaged every hour.

It is important to be able to forecast the pollution level several hours, or even a day in advance for several reasons, including:

1. To be able to take emergency measures if the pollution level is going to be above a certain threshold.
2. Helping industry make contingency plans in advance in order to minimize the cost of the emergency measures.
3. To estimate the pollution in an area where there are no measurements.
4. To take preventive actions in some places, such as schools, in order to reduce the health hazards produced by high pollution levels.

In Mexico City, the ozone level is used as a global indicator for the air quality within the different parts of the city. The concentrations of ozone are given in IMECA (Mexican air quality index). It is important to predict the daily ozone, or at least, predict it several hours in advance using the other variables measured at different stations.

It is useful to know the dependencies between the different variables that are measured, and specially their influence in the ozone concentration. This will provide a better understanding of the problem with several potential benefits:

- Determine which factors are more important for the ozone concentration in Mexico City.
- Simplify the estimation problem, by taking into account only the relevant information.
- Discover the most critical primary causes of pollution in Mexico City; these could help in future plans to reduce pollution.

We started by applying a learning algorithm to obtain an initial structure of the phenomena [11]. For this we considered 47 variables: 9 measurements for each of the 5 stations, plus the hour and month in which they were recorded. We used nearly 400 random samples, and applied the Chow and Liu algorithm to obtain the tree structure that best approximates the data distribution. This tree-structured Bayesian network is shown in Fig. 8.4.

We then considered the ozone in one station (Pedregal) as unknown, and we estimate it, one hour in advance, using the other measurements. Thus we make *ozone-Pedregal* the hypothesis variable and consider it as the root in the probabilistic tree, as shown in Fig. 8.4. From this initial structure we can get an idea of the relevance or

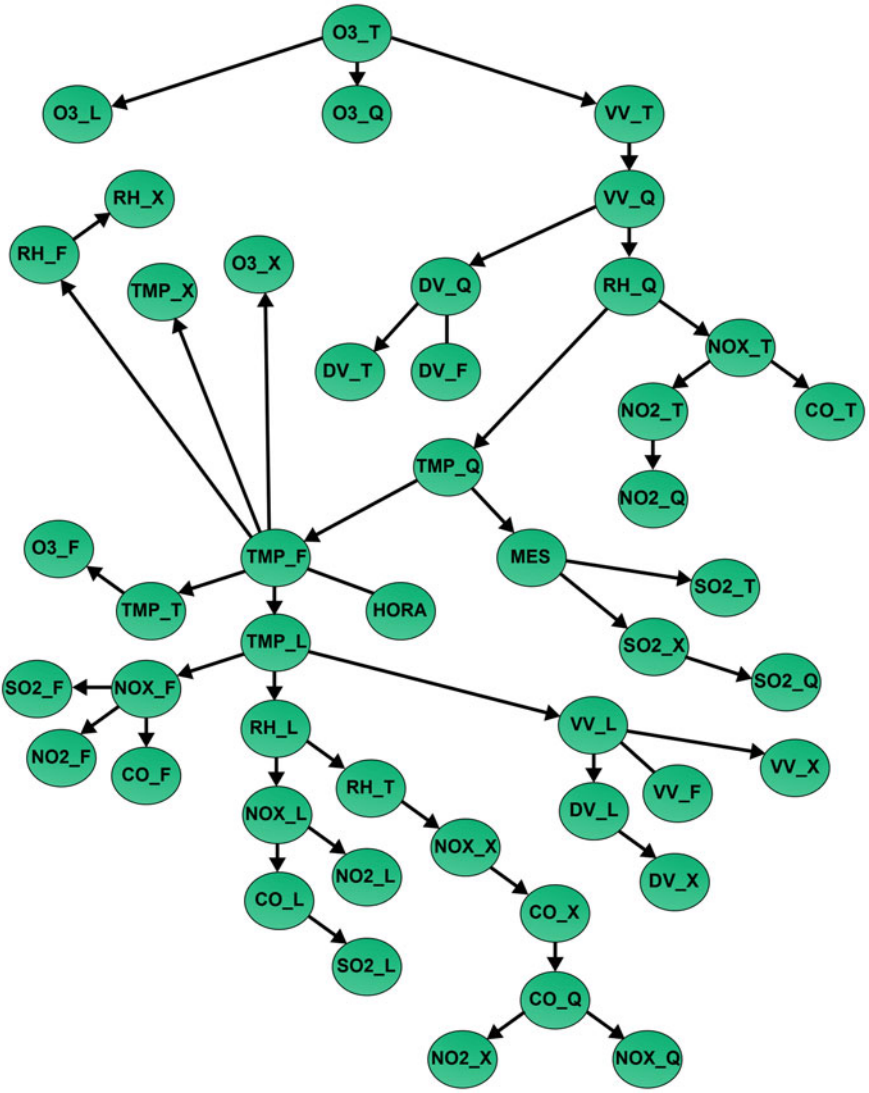


Fig. 8.4 A Bayesian tree that represents the ozone phenomena in five stations in Mexico City. The nodes represent the 47 variables according to the following nomenclature. For the measured variables, each name is formed by two parts, "measurement-station," using the following abbreviations. The measurements: O3-ozone, SO2-sulfur dioxide, CO-carbon monoxide, NO2-nitrogen dioxide, NOX-nitrogen oxides, VV-wind velocity, DV-wind direction, TMP-temperature, RH-relative humidity; the monitoring stations, T-Pedregal, F-Tlanepantla, Q-Merced, L-Xalostoc, X-Cerro de la Estrella. The other two variables correspond to the time when the measurements were taken, and they are: HORA-hour, MES-month

influence of the other variables for estimating *ozone-Pedregal*. The nodes "closest" to the root are the most important ones, and the "far-away" nodes are less important.

In this case we observe that there are three variables (ozone-Merced, ozone-Xalostoc, and wind velocity in Pedregal) that have the greatest influence in *ozone-Pedregal*. Furthermore, if the tree structure is a good approximation to the "real" structure, these three nodes make *ozone-Pedregal* independent from the rest of the variables (see Fig. 8.4). Thus, as a first test of this structure, we estimated *ozone-Pedregal* using only these three variables. We carried out two experiments: (1) estimate *ozone-Pedregal* using 100 random samples taken from the training data, and (2) estimate *ozone-Pedregal* with another 100 samples taken from separate data, not used for training. We observe that even with only three parameters, the estimations are quite good. For training data the average error (absolute difference between the real and the estimated ozone concentration) is 11 IMECA or 12 %, and for non-training data it is 26 IMECA or 22 %.

An interesting observation from the obtained structure is that *ozone-Pedregal* (located in the south of the city) depends basically on three variables, *ozone-Merced* and *ozone-Xalostoc* (located in the center and north of the city), and the wind velocity in Pedregal. Otherwise stated, pollution in the south depends basically on the pollution in the center and north of the city (where there is more transit and industry) and the wind velocity—which carries the pollution from the north to the south. This phenomenon was already known, but it was discovered automatically by learning a BN. Other, not so well-known relations, can also be discovered and could be useful for making decisions to control pollution and take emergency measures.

8.6 Additional Reading

A general book on learning Bayesian networks is [7]; Heckerman [4] has comprehensive tutorial on learning BNs. The tree and polytree learning algorithms are described in [8]. A general introduction to learning BNs from a statistical perspective is given in [10]. An analysis of different scoring functions is presented in [3].

8.7 Exercises

1. The table below gives the original data for the golf example using numeric values for some of the variables. Discretize these variables using three intervals for each variable: (a) use equal width discretization, (b) use equal data.

Outlook	Temperature	Humidity	Wind	Play
Sunny	19	High	5	N
Sunny	25	High	3	N
Overcast	26	High	3	P
Rainy	17	High	6	P
Rainy	11	Normal	15	P
Rainy	7	Normal	17	N
Overcast	8	Normal	11	P
Sunny	20	High	7	N
Sunny	19	Normal	1	P
Rainy	22	Normal	5	P
Sunny	21	Normal	20	P
Overcast	22	High	18	P
Overcast	28	Normal	16	P
Rainy	18	High	Yes	3

2. Using the discretized data from the previous problem, obtain the CPTs for the Bayesian network structure given in Fig. 8.2.
3. Continue the EM algorithm for the example in Sect. 8.2.3.2 until convergence. Show the final CPT, $P(Wind \mid Play)$, and the final data table.
4. Based on the data for the golf example in Table 8.2, learn the skeleton of a tree BN using Chow and Liu's algorithm.
5. Obtain the directions for the arcs of the skeleton from the previous problem by applying the polytree learning technique.
6. Based on the same dataset from the previous problem, Table 8.2, learn a BN using the PC algorithm. Use the cross entropy measure (seen in Chap. 2) for testing conditional independence.
7. Given the dataset in the following table, (a) learn a naive Bayesian classifier considering C as the class, (b) learn a tree-structured BN and fix the directions of the arcs considering C as the root, (c) compare the structures of both models.

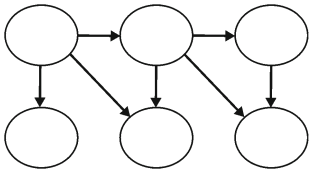
A1	A2	A3	C
0	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0
0	1	1	0
1	1	0	1
0	0	0	1
0	1	0	0
0	1	1	0

8. For the dataset in the previous problem, use the Laplacian smoothing technique to obtain the conditional probability tables for both, the NBC and the tree BN. Compare the tables to the ones obtained without smoothing.
9. *** Develop a program that implements the polytree learning algorithm. Apply it to the golf data and compare with the results from Exercise 2.
10. *** Implement a program for learning a BN from data using a score and search technique based on the MDL scoring function, and another based on independence tests (PC algorithm). Apply both to different datasets and compare the results.

References

1. Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Trans. Inf. Theory **14**, 462–467 (1968)
2. Cooper, G.F., Herskovitz, E.: A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**(4), 309–348 (1992)
3. De Campos, L.M.: A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. J. Mach. Learn. Res. **7**, 2149–2187 (2006)
4. Heckerman, D.: A Tutorial on Learning with Bayesian Networks. Innovations in Bayesian Networks, pp. 33–82. Springer, Netherlands (2008)
5. Lam, W., Bacchus, F.: Learning Bayesian belief networks: an approach based on the MDL principle. Comput. Intell. **10**, 269–293 (1994)
6. Martínez, M, Sucar, L.E.: Learning an optimal naive Bayes classifier. In: 18th International Conference on Pattern Recognition (ICPR), vol. 3, pp. 1236–1239 (2006)
7. Neapolitan, R.E.: Learning Bayesian Networks. Prentice Hall, New Jersey (2004)
8. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)
9. Rebane, G., Pearl, J.: The recovery of causal poly-trees from statistical data. In: Kanal, Laveen N., Levitt, Tod S., Lemmer, John F. (eds.) Uncertainty in Artificial Intelligence, pp. 175–182 (1987)
10. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. Springer, Berlin (1993)
11. Sucar, L.E., Ruiz-Suarez, J.C.: Forecasting air pollution with causal probabilistic networks. In: Barnett, V., Turkman, K.F. (eds.) Statistics for the Environment 3: Statistical Aspects of Pollution, pp. 185–197. Wiley, Chichester (2007)

Chapter 9
Dynamic and Temporal Bayesian Networks



9.1 Introduction

Bayesian networks usually represent the *state* of certain phenomena at an instant in time. However, in many applications, we want to represent the temporal evolution of a certain process, that is, how the different variables evolve over time, known also as time series.

There are two basic types of Bayesian network models for dynamic processes: state based and event based. State-based models represent the state of each variable at discrete time intervals, so that the networks consist of a series of *time slices*, where each time slice indicates the value of each variable at time t; these models are known as dynamic Bayesian networks. Event-based models represent the changes in state of each state variable; each temporal variable will then correspond to the time in which a state change occurs. These types of models are known as event networks or temporal networks.

In this chapter we will review both, dynamic Bayesian networks and temporal networks, including representation, inference, and learning.

9.2 Dynamic Bayesian Networks

Dynamic Bayesian networks (DBNs) are an extension of Bayesian networks to model dynamic processes. A DBN consists of a series of *time slices* that represent the state of all the variables at a certain time, t; a kind of snapshot of the evolving temporal process. For each temporal slice, a dependency structure between the variables at that time is defined, called the *base network*. It is usually assumed that this structure is duplicated for all the temporal slices (except the first slice, which can be different). Additionally, there are edges between variables from different slices, with their directions following the direction of time, defining the *transition network*. Usually, DBNs are restricted to have directed links between consecutive temporal slices, known as a first-order Markov model; although, in general, this is not necessary. An example of a DBN with 3 variables and 4 time slices is depicted in Fig. 9.1.

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_9

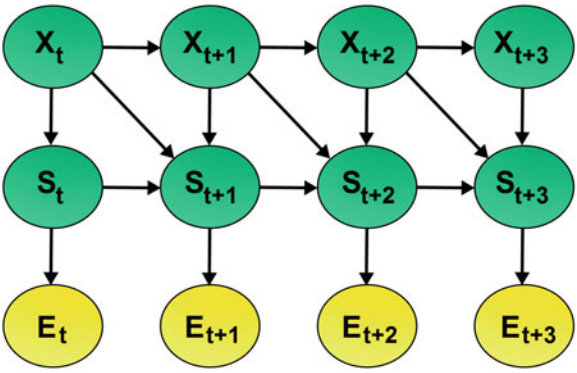


Fig. 9.1 An example of a DBN with 3 variables and 4 time slices. In this case, the base structure is $X \rightarrow S \rightarrow E$ which is repeated across the 4 time slices

Most of the DBNs considered in practice satisfy the following conditions:

- First-order Markov model. The state variables at time t depend only on the state variables at time $t-1$ (and other variables at time t).
- Stationary process. The structure and parameters of the model do not change over time.

DBNs can be seen as a generalization of Markov chains and hidden Markov models (HMMs). A Markov chain is the simplest DBN, in which there is only one variable, X_t, per time slice, directly influenced only by the variable in the previous time. In this case, the joint distribution can be written as:

$$P(X_1, X_2, \ldots, X_T) = P(X_1)P(X_2 \mid X_1) \ldots P(X_T \mid X_{T-1}) \tag{9.1}$$

A hidden markov model has two variables per time stage, one that is known as the *state variable*, S; and the other as the *observation variable*, Y. It is usually assumed that S_t depends only on S_{t-1} and Y_t depends only on S_t. Thus, the joint probability can be factored as follows:

$$P(\{S_{1:T}, Y_{1:T}\}) = P(S_1)P(Y_1 \mid S_1) \prod_{t=2}^{T} P(S_t \mid S_{t-1})P(Y_t \mid S_t) \tag{9.2}$$

Markov chains and HMMS are particular cases of DBNs, which in general can have N variables per time step, with any base and transition structures. Another particular variant of DBNs are Kalman Filters, which also have one state and one observation variable, but both variables are continuous. The basic Kalman filter assumes Gaussian distributions and linear functions for the transition and observation models.

Table 9.1 Learning dynamic Bayesian networks: 4 basic cases

Structure	Observability	Method
Known	Full	Maximum likelihood estimation
Known	Partial	Expectation–maximization (EM)
Unknown	Full	Search (global) or tests (local)
Unknown	Partial	EM and global or local

9.2.1 Inference

There are several classes of inferences that can be performed with DBNs. In the following, we briefly mention the main types of inference, where $\mathbf{X}$ are the unobserved (hidden) variables, and $\mathbf{Y}$ are the observed variables [9]:

- *Filtering*. Predict the next state based on past observations: $P(X_{t+1} \mid Y_{1:t})$.
- *Prediction.* Predict future states based on past observations: $P(X_{t+n} \mid Y_{1:t})$.
- *Smoothing.* Estimate the current state based on past and future observations (useful for learning): $P(X_t \mid Y_{1:T})$.
- *Decoding.* Find the most likely sequence of hidden variables given the observations: $\text{ArgMax}(X_{1:T})\ P(X_{1:T} \mid Y_{1:T})$.

Efficient inference methods have been developed for particular types of models, such as HMMs [10] (see Chap. 5). However, for more complex models, inference becomes computationally intractable. In these cases, we can apply approximate methods based on sampling, such as Markov chain Monte Carlo [6]. A popular approximate method is *particle filters*, which approximate the state probability distribution (belief state) with a set of weighted *particles* or samples [9].

9.2.2 Learning

As with BNs, learning dynamic Bayesian networks involves two aspects: (i) learning the structure or graph topology and (ii) learning the parameters or CPTs for each variable. Additionally, we can consider two cases in terms of the observability of the variables: (a) full observability, when there is data for all the variables and (b) partial observability, when some variables are unobserved or hidden, or we have missing data. There are four basic cases for learning DBNs, see Table 9.1.

For all the cases, we can apply extensions of the methods for parameter and structure learning for Bayesian networks that we reviewed in Chap. 8. We describe one of these extensions below, for the case of unknown structure and full observability.

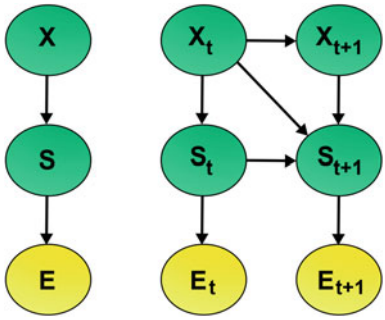


Fig. 9.2 Learning a DBN: first, we obtain the base structure (*left*), and then the transition structure (*right*)

Assuming that the DBN is stationary (time invariant), we can consider that the model is defined by two structures: (i) the base structure and (ii) the transition structure. Thus, we can divide the learning of a DBN into two parts, first learn the base structure, and then, given the base structure, learn the transition structure, see Fig. 9.2.

For learning the base structure, we can use all the available data for each variable, ignoring the temporal information. This is equivalent to learning a BN, so we can apply any of the methods used for learning BNs (see Chap. 8).

For learning the transition network, we consider the temporal information, in particular the data for all variables in two consecutive time slices, $\mathbf{X_t}$ and $\mathbf{X_{t+1}}$. Considering the base structure, we can then learn the dependencies between the variables at time t and $t + 1$ (assuming a first–order Markov model), and restricting the direction of the edges from the past to the future.

Here, we have described in a simple, general way the two phases for learning a DBN; however, there are several variants of this idea that have been developed (see the additional readings section).

Aside from DBNs, there are some alternative BN representations for describing temporal processes that have been developed. An example of a different type of temporal Bayesian network is *event networks* [1, 5].

9.3 Temporal Event Networks

Temporal event networks (TENs) are an alternative to DBNs for modeling dynamic processes. In a temporal event network, a node represents the *time* of occurrence of an event or state change of certain variable, in contrast to a node in a DBN that represents the state value of a variable at a certain time. For some problems, in which there are few state changes in the temporal range of interest, event networks provide a simpler and more efficient representation; however, for other applications such as monitoring or filtering, DBNs are more appropriate.

Several variants of TENs have been proposed, such as *time nets* and *temporal nodes Bayesian networks* (TNBNs). In the rest of this section we will focus on TNBNs.

9.3.1 Temporal Nodes Bayesian Networks

A Temporal Nodes Bayesian Network (TNBN) [1, 5] is composed of a set of Temporal Nodes (TNs). TNs are connected by edges, where each edge represents a causal–temporal relationship between TNs. There is at most one state change for each variable (TN) in the temporal range of interest. The value taken by the variable represents the interval in which the event occurs. Time is discretized in a finite number of intervals, allowing a different number and duration of intervals for each node (multiple granularity). Each interval defined for a child node represents the possible delays between the occurrence of one of its parent events (cause) and the corresponding child event (effect). Some Temporal Nodes do not have temporal intervals, these correspond to Instantaneous Nodes. Root nodes are instantaneous by definition [1]. Formally, a TNBN is defined as follows.

A TNBN is defined as a pair $B = (G, \Theta)$. G is a Directed Acyclic Graph, $G = (\mathbf{V}, \mathbf{E})$. G is composed of $\mathbf{V}$, a set of Temporal and Instantaneous Nodes; $\mathbf{E}$ a set of edges between Nodes. The Θ component corresponds to the set of parameters that quantify the network. Θ contains the values $\Theta_{v_i} = P(v_i|Pa(v_i))$ for each $v_i \in \mathbf{V}$; where $Pa(v_i)$ represents the set of parents of v_i in G.

A Temporal Node, v_i, is defined by a set of states $\mathbf{S}$, each state is defined by an ordered pair $S = (\lambda, \tau)$, where λ is the value of a random variable and $\tau = [a, b]$ is the interval associated, with an initial value $\mathtt{a}$ and a final value $\mathtt{b}$. These values correspond to the time interval in which the state change occurs. In addition, each Temporal Node contains an extra default state $s =$ ("no change", $\emptyset$), which has no interval associated. If a Node has no intervals defined for any of its states, then it receives the name of Instantaneous Node.

The following is an example based on [1]:

Example 9.1 Assume that at time $t = 0$, an automobile accident occurs, that is, a Collision. This kind of accident can be classified as *severe*, *moderate*, or *mild*. To simplify the model, we will consider only two immediate consequences for the person involved in the collision: Head Injury and Internal Bleeding. A Head Injury can bruise the brain and chest injuries can lead to internal bleeding. These are all instantaneous events that may generate subsequent changes; for example, the head injury event might generate dilated pupils and unstable vital signs. Suppose that we gathered information about accidents that occurred in a specific city. The information indicates that there is a strong causal relationship between the severity of the accident and the immediate effect of the patient's state. Additionally, a physician domain expert provided some important temporal information: If a head injury occurs, the brain will start to swell and if left unchecked the swelling will cause the pupils to dilate within 0–60 min. If internal bleeding begins, the blood volume will start to fall, which will tend to destabilize vital signs. The time required to destabilize vital signs will depend on the severity of the bleeding: if the bleeding is gross, it will take from 0 to 15 min; if the bleeding is slight it will take from 15 to 45 min. A head injury also tends to destabilize vital signs, taking from 0 to 15 min to make them unstable.

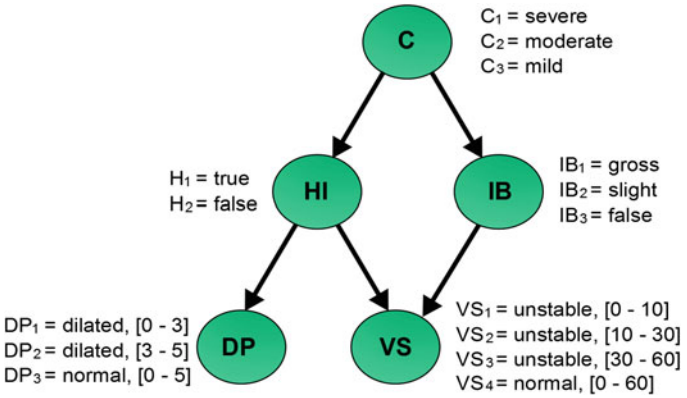


Fig. 9.3 A TNBN for the accident example. C = Collision, HI = Head Injury, IB = Internal Bleeding, DP = Dilated Pupils, and VS = Vital Signs

The graphical representation of a TNBN for the accident example is shown in Fig. 9.3. The model presents three instantaneous nodes: Collision, Head Injury and Internal Bleeding. These events will generate subsequent changes that are not immediate: Dilated Pupils and unstable Vital Signs, which depend on the severity of the accident, and therefore have temporal intervals associated to them. This TNBN contains only 5 nodes (in contrast to the 25 nodes that would be required for the equivalent DBN).

In TNBNs, each variable represents an event or state change. So, only one (or a few) instance(s) of each variable is required, assuming there is one (or a few) change(s) of a variable's state in the temporal range of interest. No copies of the model are needed, and no assumption about the Markovian nature of the process is made. TNBNs can deal with multiple granularity, because the number and the size of the intervals for each node can be different.

9.3.1.1 Inference

A TNBN allows for reasoning about the probability of occurrence of certain events, for diagnosis (i.e., finding the most probable cause of a temporal event) or prediction (i.e., determining the probable future events that will occur given a certain event). For this, standard probability propagation techniques for standard BNs (see Chap. 7) can be applied. However, given that a TNBN represents relative times between events, the cases of prediction and diagnosis have to be differentiated for doing probabilistic inference:

Prediction. In the case where at least one of the root (instantaneous) nodes of the TNBN is part of the evidence, then the time reference for the model is fixed and probability propagation can be performed directly, obtaining the posterior probability of the subsequent events (a probability for each temporal interval

of each non-instantiated temporal node). For example, considering the example in Fig. 9.3, if we know the severity of the *Collision* (e.g. moderate), then via probability propagation we obtain the posterior probability of the event *Dilated Pupils* occurring during each temporal interval and not occurring at all (False). The posterior probability of other temporal and instantaneous nodes is obtained in a similar manner. Note that the time intervals of the TNs will be with respect to the occurrence of the *Collision*, which will correspond to $t = 0$.

Diagnosis. In the case where none of the instantaneous nodes are known, and the evidence is given only for temporal nodes, then several *scenarios* need to be considered, as we do not know to which interval to assign the evidence given that there is no time reference. In this case, all the n possible intervals for the TN have to be considered, performing inference n times, one for each interval. The results for each scenario have to be maintained, until there is additional evidence, such as the occurrence of another event, that allows for discarding some scenarios. Considering the accident example of Fig. 9.3, assume that a paramedic arrives and finds that the person has his *Pupils Dilated*. As the time of the accident is unknown, then the evidence must be given to all the three temporal intervals for this variable, generating 3 scenarios. If later on, the time of the accident is determined, then the appropriate scenario is kept and the others are discarded.

9.3.1.2 Learning

Learning a TNBN involves three aspects: (i) learning the temporal intervals for the temporal nodes, (ii) learning the structure of the model, and (iii) learning the parameters of the model. As these three components are interrelated, an iterative procedure is required, that learns an initial estimate of one (or more) of these aspects, and then improves these initial estimates iteratively. Next, we present an algorithm for learning the three components of a TNBN [7].

The algorithm assumes that the root nodes are instantaneous nodes and it obtains a finite number of non overlapping intervals for each temporal node. It uses the times (delays) between parent events and the current node as input for learning the intervals. With this top–down approach the algorithm is guaranteed to arrive at a local maximum in terms of predictive score. The algorithm initially assumes that the structure of the network is known, and later the structure learning component is incorporated.

A learning algorithm (known as *LIPS*) [7] for TNBNs is summarized as follows:

1. First, it performs an initial discretization of the temporal variables, for example using an Equal-Width discretization. With this process it obtains an initial approximation of the intervals for all the Temporal Nodes.
2. Then it performs standard BN structural learning. Specifically, the K2 learning algorithm [3] (see Chap. 8) is used to obtain an initial structure and corresponding parameters.

Table 9.2 Initial sets of intervals obtained for the node *Dilated Pupils*

Partition	Intervals
Head Injury = true	[11-35]
	[11–27] [32–53]
	[8–21] [25–32] [45–59]
Head Injury = false	[3–48]
	[0–19] [39–62]
	[0–14] [28–40] [47–65]

There are three sets of intervals for each partition

3. The interval learning algorithm refines the intervals for each temporal node (TN) by means of clustering. For this, it uses the information of the configurations of the parent nodes. To obtain a set of intervals a Gaussian mixture model is used as a clustering algorithm for the temporal data. Each cluster corresponds, in principle, to a temporal interval. The intervals are defined in terms of the mean and the standard deviation of the clusters. The algorithm obtains different sets of intervals that are merged and combined, this process generates different interval sets that will be evaluated in terms of the predictive accuracy. The algorithm applies two pruning techniques in order to remove sets of intervals that may not be useful and also to keep a low complexity for the TNBN. The best set of intervals (which may not be those obtained in the first step) for each TN is selected based on predictive accuracy. When a TN has as parents other Temporal Nodes, the configurations of the parent nodes are not initially known. In order to solve this problem, the intervals are sequentially selected in a top–down fashion according to the TNBN structure.
4. Finally, the parameters (CPTs) are updated according to the new set of intervals for each TN.

The algorithm then iterates between structure learning and interval learning.

We illustrate the process of obtaining the intervals for the TN *Dilated Pupils* (DP) of Fig. 9.3 (the intervals in this example are different from those shown in the figure). We can see that its parent node (Head Injury) has two configurations, *true* and *false*. Thus, the temporal data of Dilated Pupils is divided into two partitions, one for each configuration of the parent node. Then, for each partition, the first approximation of the interval learning step of the previous algorithm is applied. The expectation-maximization algorithm is applied to get Gaussian mixture models with parameters 1, 2, and 3 as the number of clusters. That gives six different sets of intervals, as shown in Table 9.2. Then each set of intervals is evaluated in terms of its prediction performance to measure its quality and the set of intervals with the best score is selected.

Now we present a complete example of the TNBN learning algorithm considering that we have data for the accident example illustrated in Fig. 9.3. First we assume we have data from other accidents that is similar to the one presented in the upper part of Table 9.3. The first three columns have nominal data; however, the last two columns

Table 9.3 Collected data to learn the TNBN of Fig. 9.3

Collision	Head Injury	Internal Bleeding	Dilated Pupils	Vital Signs
Severe	True	Gross	14	20
Moderate	True	Gross	25	25
Mild	False	False	-	-
…	…	…	…	…
Severe	True	Gross	[10–20]	[15–30]
Moderate	True	Gross	[20–30]	[15–30]
Mild	False	False	-	-
…	…	…	…	…

Top original data showing the time of occurrence of the temporal events. *Bottom* temporal data after the initial discretization. For dilated pupils and vital signs the temporal data represents the minutes after the collision occurred

have temporal data which represent the occurrence of those events after the collision. Those two columns would correspond to Temporal Nodes of the TNBN. We start by applying the equal-width discretization on the numerical data, so it would yield results similar to the ones presented in the lower part of Table 9.3.

Using the discretized data, we can apply a structure learning algorithm (like K2) using the partial ordering of the temporal events: {Collision}, {Head Injury, Internal Bleeding}, and {Dilated Pupils, Vital Signs}. Now we have an initial TNBN; however, the obtained intervals are somewhat naive, so the interval learning step of the algorithm can be applied to improve the initial temporal intervals. This process will learn a TNBN similar to the one presented in Fig. 9.3.

9.4 Applications

We illustrate the application of dynamic BN models in two domains. First, dynamic Bayesian networks are used for dynamic gesture recognition. Then, temporal event networks are used for predicting HIV mutational pathways.

9.4.1 DBN: Gesture Recognition

Dynamic Bayesian networks provide an alternative to HMMs for dynamic gesture recognition. They have the advantage of greater flexibility in terms of the structure of the models. In this section we consider a particular type of DBN known as *dynamic Bayesian network classifier* (DBNC) [2]. Similarly to HMMs, a DBNC has a hidden state variable for each time instant, S_t; however, the observation variable is decomposed into m attributes, $A_t^1, \ldots, A_t^m$, which are assumed to be conditionally

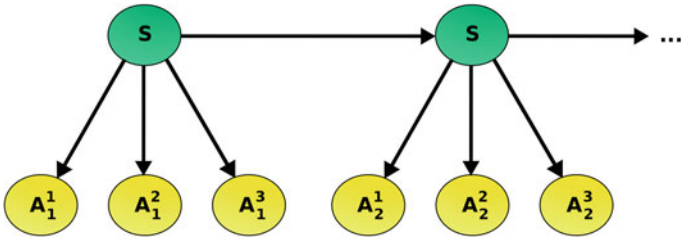


Fig. 9.4 Graphical Representation of a DBNC with 3 attributes unrolled two times

independent given S_t. Thus, the *base structure* for a DBNC has a star-like structure with the directed links from S_t to each attribute A_t^i (see Fig. 9.4).

The joint probability of a DBNC can be factored as follows:

$$P(\{S_{1:T}, \mathbf{A}_{1:T}\}) = P(S_1)\left[\prod_{m=1}^{M} P(A_1^m \mid S_1)\right]\prod_{t=2}^{T} P(S_t \mid S_{t-1})\left[\prod_{m=1}^{M} P(A_t^m \mid S_t)\right] \tag{9.3}$$

where $\mathbf{A} = A^1, \ldots, A^m$.

The difference with the joint probability of a HMM is that instead of having $P(A_t \mid S_t)$, we have the product of each attribute given the state, $\prod_{m=1}^{M} P(A_t^m \mid S_t)$.

Parameter learning and classification with DBNCs are done in a similar manner as with HMMs, using a modified version of the Baum-Welch and Forward algorithms, respectively.

9.4.1.1 Gesture Recognition with DBNCs

DBNC have been applied for recognizing different hand gestures oriented to command a mobile robot. A set of 9 different gestures were considered; a key frame for each type of gesture is illustrated in Fig. 9.5. Initially, the hand of the person performing the gesture is detected and tracked using a camera and specialized vision software. A rectangle approximates the position of the hand in each image in the sequence, and from these rectangles a set of features is extracted which are the observations for the DBNCs.

The features include motion and posture information, including in total seven attributes: (i) three features to describe motion and (ii) four to describe posture. Motion features are Δarea—or changes in the hand area-, Δx and Δy -or changes in hand position on the XY-plane of the image. The conjunction of these three attributes allows us to estimate hand motion in the Cartesian space, *XYZ*. Each one of these features takes only one of three possible values: $+, -, 0$, that indicate increment, decrement, or no change, depending on the area and position of the hand in the previous image of the sequence.

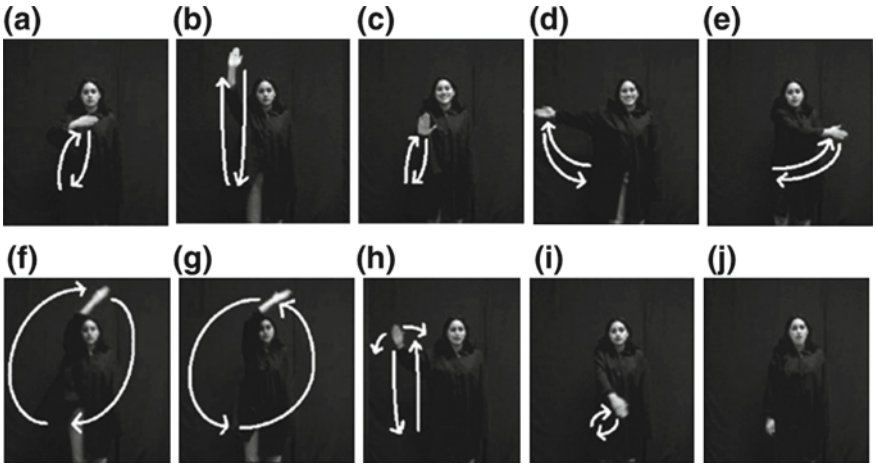


Fig. 9.5 Types of gestures considered in the experiments: **a** come, **b** attention, **c** stop, **d** right, **e** left, **f** turn left, **g** turn right, **h** waving-hand and **i** pointing; **j** initial and final position for each gesture

Posture features named *form, right, above, and torso* describe hand orientation and spatial relations between the hand and other body parts, such as the face and torso. Hand orientation is represented by *form*. This feature is discretized into one of three values: + if the hand is vertical, − if the hand is horizontal, or 0 if the hand is leaning to the left or right over the XY plane. *right* indicates if the hand is to the right of the head, *above* if the hand is above the head, and *torso* if the hand is in front of the torso. These three latter attributes take binary values, true or false, that represent if their corresponding condition is satisfied or not. An example of posture extraction in terms of these variables is depicted in Fig. 9.6.

As with HMMs, a DBNC is trained for each type of gesture; and for classification the probability of each model is evaluated and the one with the highest probability is selected as the recognized gesture.

9.4.1.2 Experiments

Three experiments were done to compare classification and learning performances of DBNCs and HMMs. In the first experiment, gestures taken from the same person are used for recognition. In the second experiment, the generalization capabilities of the classifiers are evaluated by training and testing with gestures from different people. Experiment three considers gestures with variations on distance and rotation. Additionally, the DBNC and HMM models using only motion, and using motion and posture information were compared. The number of hidden states in each model was varied between 3 and 18.

For the first experiment, 50 executions of each type of gesture by one individual were recorded; 20 samples were used for training and 30 for testing. In the second

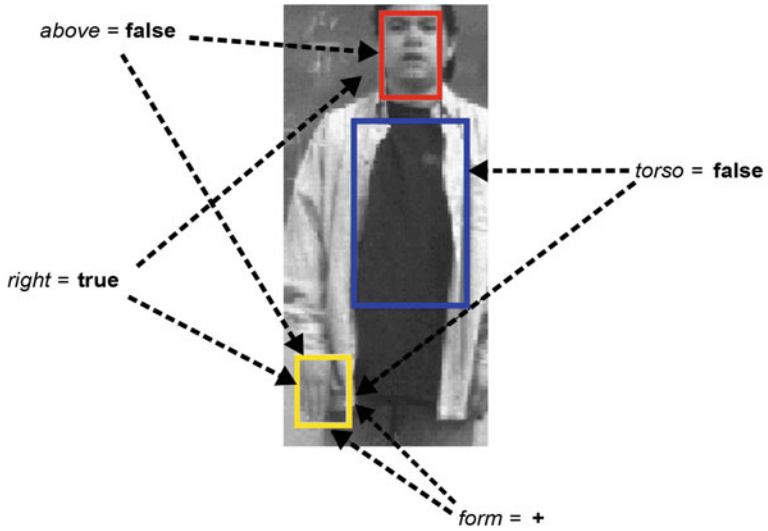


Fig. 9.6 The image illustrates the posture features. It shows that the hand has a vertical position, below the head, to the right of the user and not over the user's torso. Thus, the attribute values are *above* = *false*, *right* = *true*, *torso* = *false*, and *form* = +

experiment, the models learned for one person were evaluated with gestures executed by other 14 persons, 2 samples per person for each gesture class. For the experiment on distance variation, 15 samples were randomly extracted for each gesture performed at 2 and 4 m, giving a test set of 30 samples per gesture. Similarly, for rotation variation, 15 random samples of each gesture at +45 and −45 degrees were used.

An important difference between the DBNCs and the HMMs is the number of parameters that are required for each model. The number of parameters to specify state observation distributions of HMMs with posture-motion features is 648 and with only motion data is 27. With DBNCs, parameters are 21 in the former case, and 12 in the latter case. This significant reduction in the number of parameters for the DBNC has an important impact in the training time, as well as in the classification accuracy when the number of training instances is low.

In the case of the experiments with the same person using motion and posture attributes, both HMMs and DBNCs obtain a very good performance, with recognition rates between 96 and 99 %, depending on the number of states (best results with 12–15 states). DBNCs obtain slightly better recognition results, but with a significant reduction in training time, about ten times faster than HMMs.

For the experiments with multiple people, as expected the performance of both, HMMs and DBNCs decreases, to about 86 % recognition with motion-posture attributes. If only motion attributes are used, the performance is in the order of 65 %, about 20 points less than when incorporating posture! As with most classification problems, the selection of the appropriate set of attributes is critical.

In the case of the third experiments, variations in distance and orientation also have an impact in the recognition rates, with the second aspect having a greater effect. Varying the distance from the camera to the user between 2 and 4 m, reduces the recognition rate to about 90 % (single user); if the orientation is varied in the interval between +45 and −45 degrees, the performance goes down to approx. 70 %. In both situations, the results with HMMs and DBNCs are similar in terms of performance.

Experimental results show the competitiveness in terms of recognition rates of DBNCs in comparison to standard HMMs in various issues in gesture recognition. Attribute factorization allows an important decrease on training time for a DBNC in comparison with equivalent HMMs, this allows online learning of gestures. Additionally, DBNCs require less training examples to achieve a similar performance as the corresponding HMMs, in particular when the number of attributes increases.

9.4.2 TNBN: Predicting HIV Mutational Pathways

In this section, we explore the application of TNBNs to uncover the temporal relationships between drug-resistance mutations of the HIV virus and antiretroviral drugs, unveiling possible mutational pathways and establishing their probabilistic-temporal sequence of appearance [8].

The human immunodeficiency virus (HIV) is one of the fastest evolving organisms on the planet. Its remarkable variation capability makes HIV able to escape from multiple evolutionary forces naturally or artificially acting on it, through the development and selection of adaptive mutations. This is the case of antiretroviral therapy (ART), a strong selective pressure acting on HIV that, under suboptimal conditions, readily selects for mutations that allow the virus to replicate even in the presence of highly potent antiretroviral drug combinations. In particular, we address the problem of finding mutation–mutation and drug–mutation associations in individuals receiving antiretroviral therapy. We have focused on protease inhibitors (PIs), a family of antiretroviral drugs widely used in modern antiretroviral therapy. The development of drug-resistant viruses compromises HIV control, with a consequent further deterioration of the patient's immune system. Hence, there is interest in having a profound understanding of the dynamics of appearance of drug-resistance mutations.

Historical data from HIV patients was used to learn a TNBN, and then this model was evaluated in terms of the discovered relationships according to a domain expert and its predictive power.

9.4.2.1 Data

Clinical data from 2373 patients with HIV subtype B was retrieved from the HIV Stanford Database (HIVDB) [11]. The isolates in the HIVDB were obtained from longitudinal treatment profiles reporting the evolution of mutations in individual sequences. For each patient, data consisted of an initial treatment

(a combination of drugs) administered to the patient and a list of laboratory resistance tests at different times (in weeks). Each test included a list of the most frequent mutations in the viral population within the host at a specific time after the initiation of treatment. An example of the data is presented in Table 9.4. The number of studies available varied from 1 to 10 studies per patient history.

Antiretrovirals are usually classified according to the enzyme that they target. We focused on the viral protease, as this is the smallest of the viral enzymes in terms of number of aminoacids. Nine protease inhibitors are currently available, namely: Amprenavir (APV), Atazanavir (ATV), Darunavir (DRV), Lopinavir (LPV), Indinavir (IDV), Nelfinavir (NFV), Ritonavir (RTV), Tripanavir (TPV), and Saquinavir (SQV).

To test the ability of the model for predicting clinically relevant data, a subset of patients from the original dataset was selected, including individuals that received ART regimes including LPV, IDV, and SQV. Relevant major drug-resistance mutations associated with the selected drugs were included. The mutations selected were: V32I, M46I, M46L, I47V, G48V, I54V, V82A, I84V, and L90M. Since we used a subset of drugs, the number of patients in the final dataset was reduced to 300 patients.

9.4.2.2 Model and Evaluation

A TNBN was learned from the reduced HIVDB with the learning algorithm described in Sect. 9.3. Two modifications to the original algorithm were made, one to measure the strength of the temporal-probabilistic relations, and another to vary the variable order given to the structure learning algorithm (K2), so the results are not biased by a particular predefined order.

In order to evaluate the models and to measure the statistical significance of edge strengths, non-parametric bootstrapping was used (obtaining several models). Two thresholds were defined for considering a relation as important. A strong relation was defined as one that appeared in at least 90 % of the graphs, and a suggestive relation was defined as one that occurred with values between 70 and 90 %. Since the approach for selecting the drugs and mutations is based on experts opinions, we used a more elaborate way to obtain the order for the K2 algorithm. For this experiment,

Table 9.4 An example of the data

Patient	Initial Treatment	List of Mutations	Time (Weeks)
Pat_1	LPV, FPV, RTV	L63P, L10I	15
		V77I	25
		I62V	50
Pat_2	NFV, RTV, SQV	L10I	25
		V77I	45

Patient Pat_1 with 3 temporal studies, and patient Pat_2 with two temporal studies

different orderings for the K2 algorithm were considered and the one with the highest predictive accuracy was selected.

Figure 9.7 depicts the TNBN obtained. The green nodes represent the antiretroviral drugs, and the yellow nodes the mutations. For each mutation (temporal nodes), the associated temporal intervals with the probability of occurrence of the mutation in that interval are shown. Arcs that suggest a strong relation are marked with a $*$.

The model was able to predict clinically relevant associations between the chosen drugs and mutations. Indeed, a strong association between SQV, G48V, and I84V was readily predicted in the model, although no temporal associations were observed between the two mutations. All three drugs showed direct associations with L90M reflecting the fact that this mutation causes cross-resistance to many members of the PI family. Remarkably, the two possible mutational pathways for LPV resistance [8] were predicted:

- I54V $\rightarrow$ V32I $\rightarrow$ I47V
- L90M $\rightarrow$ M46IL $\rightarrow$ I84V

Whether the temporal order of mutations is relevant, still needs to be further evaluated. Also, the shared mutational pathway between IDV and LPV was observed, involving mutations L90M, M46IL, I54V, V82A, and I84V.

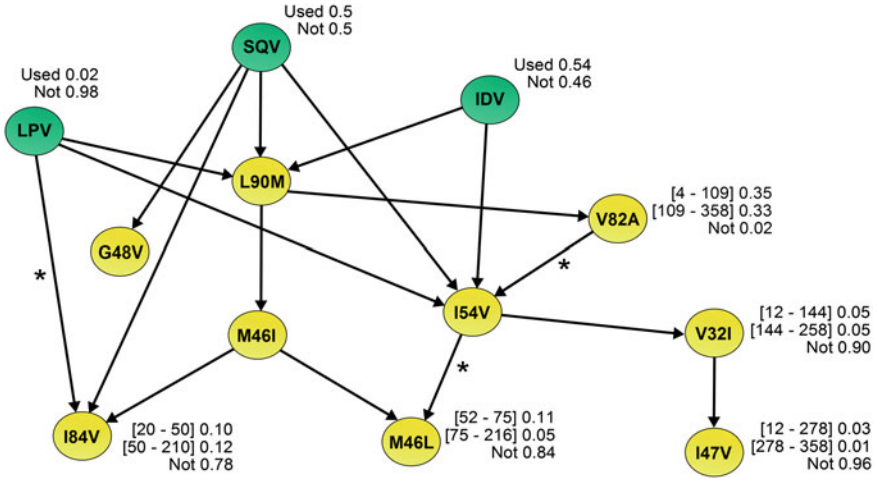


Fig. 9.7 The learned TNBN model that depicts the temporal-probabilistic relations between a set of antiretroviral drugs (*upper* 3 nodes) and relevant HIV mutations (*other* nodes). An arc labeled with a * represents a strong relation. Only a few sets of intervals associated with their respective temporal nodes are shown

9.5 Additional Reading

A comprehensive review of dynamic Bayesian networks is presented in [9]. Friedman et al. [4, 6] include techniques for learning DBNs. Temporal nodes Bayesian networks are introduced in [1], and extended with canonical temporal models in [5]. The algorithm for learning a TNBN is described in [7].

9.6 Exercises

1. Given the DBN in Fig. 9.1, consider that it is a stationary model. Which parameters are required for the complete specification of the model?
2. Assuming that all the variables are binary (false, true), specify the CPTs for the previous exercise (define the probabilities arbitrarily, just satisfying the probability axioms).
3. Given the DBN in Fig. 9.1, and the CPTs for the previous exercise: (a) obtain the posterior probability of X_{t+1} given $X_t = S_t = true$ (filtering), (b) obtain the posterior probability of X_{t+2} and X_{t+3} given $X_t = S_t = true$ (prediction), (c) obtain the posterior probability of X_{t+1} given $E_t = E_{t+1} = E_{t+2} = false$ (smoothing).
4. Consider the TNBN in Fig. 9.3. Define the CPTs for all the variables according to the values/intervals shown in the figure. Specify the parameters according to your intuition (subjective estimates).
5. Considering the structure and parameters for the TNBN of the previous exercise, obtain the posterior probability of all the variables given the evidence $C = moderate$ using probabilistic inference (you can apply any of the inference techniques for BNs).
6. Repeat the previous problem considering as evidence $DP = dilated$. As the relative time of occurrence of this event is not known, consider the different possible scenarios. Which scenario will apply if we later find out that the pupils became dilated 4 time units after the accident?
7. Modify the inference (forward) and learning (Baum–Welch) algorithms for HMMs (see Chap. 5) so that they can be applied to dynamic Bayesian network classifiers.
8. *** Search for data sets in different dynamic domains to learn dynamic Bayesian models (see next exercises). For example, stock exchange data, weather data, medical longitudinal studies, etc. Determine, according to each application, which type of model, state based or event based, is more appropriate.
9. *** Develop a program that learns a DBN considering a two phase procedure: first learn the initial structure and parameters (for $t = 0$); then learn the transition structure and parameters (for $t = k + 1$ given $t = k$).
10. *** Develop a program that implements the *LIPS* algorithm for learning TNBNs.

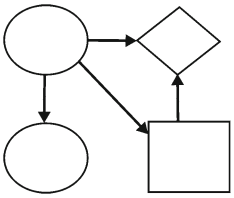
References

1. Arroyo-Figueroa, G., Sucar, L.E.: A temporal Bayesian network for diagnosis and prediction. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI). Morgan-Kaufmann, San Mateo, pp. 13–20 (1999)
2. Avilés-Arriaga, H.H., Sucar, L.E., Mendoza-Durán, C.E., Pineda-Cortés, L.A.: Comparison of dynamic naive Bayesian classifiers and hidden Markov models for gesture recognition. J. Appl. Res. Technol. **9**(1), 81–102 (2011)
3. Cooper, G.F., Herskovitz, E.: A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**(4), 309–348 (1992)
4. Friedman, N., Murphy, K., Russell, S.: Learning the Structure of Dynamic Probabilistic Networks. In: Proceedings of the Fourteenth Conference on Uncertainty in Artificial (UAI). Morgan Kaufmann Publishers Inc., pp. 139–147 (1998)
5. Galán, S.F., Arroyo-Figueroa, G., Díez, F.J., Sucar, L.E.: Comparison of two types of event Bayesian networks: a case study. Appl. Artif. Intell. **21**(3), 185–209 (2007)
6. Ghahramani, Z.: Learning Dynamic Bayesian Networks. Lecture Notes in Computer Science **1387**, 168–197 (1998)
7. Hernández-Leal, P., González, J.A., Morales, E.F., Sucar, L.E.: Learning temporal nodes Bayesian networks. Int. J. Approx. Reason. **54**(8), 956–977 (2013)
8. Hernández-Leal, P., Rios-Flores, A., Ávila-Rios, S., Reyes-Terán, G., González, J.A., Fiedler-Cameras, L., Orihuela-Espina, F., Morales, E.F., Sucar, L.E.: Discovering HIV mutational pathways using temporal bayesian networks. Artif. Intell. Med. **57**(3), 185–195 (2013)
9. Murphy, K.: Dynamic Bayesian networks: representation, inference and learning. dissertation, University of California, Berkeley (2002)
10. Rabiner, R., Juang, B.: Fundamentals of Speech Recognition. Prentice Hall, New Jersey (1993)
11. Shafer, R.: Rationale and uses of a public HIV drug-resistance database. J. Infect. Dis. **194**(1), 51–58 (2006)

Part III
Decision Models

Probabilistic graphical models that involve decisions are presented in this part. These models involve, besides random variables, decision variables and utilities; and their aim is to help decision makers to take the *best* decisions under uncertainty. One chapter is dedicated to models that have one or few decisions; while the other chapter focuses on sequential decision problems in which many decisions have to be taken over time.

Chapter 10
Decision Graphs



10.1 Introduction

The models that were covered in Part II have only random variables, so they can be used for estimating the posterior probability of a set of variables given some evidence; for example, for classification, diagnosis, or prediction. They can also provide the most probable combination of values for a subset of variables (most probable explanation) or the global probability of the model given some observations. However, they cannot be directly used to make decisions.

In this, and the following chapter, we will present *decision models*, whose aim is to help the decision-maker to choose the *best* decisions under uncertainty. We will consider that the best decisions are those that maximize the expected utility of an agent, given its current knowledge (evidence) and its objectives, under a decision-theoretic framework. These types of agents are known as *rational agents*.

After a brief introduction to decision theory, in this chapter we will describe two types of modeling techniques for problems with one or few decisions: decision trees and influence diagrams. As in the case of probabilistic models, these techniques take advantage of the dependency structure of the problem to have a more compact representation and a more efficient evaluation.

10.2 Decision Theory

Decision Theory provides a normative framework for decision-making under uncertainty. It is based on the concept of *rationality*, that is, that an agent should try to maximize its utility or minimize its costs. This assumes that there is some way to assign utilities (usually a number that can correspond to monetary value or any other scale) to the result of each alternative action, such that the best decision is the one that has the highest utility. In general, an agent is not sure about the results of each of its possible decisions, so it needs to take this into account when it estimates the value of each alternative. In decision theory we consider the *expected utility*, which makes an

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_10

average of all the possible results of a decision, weighted by their probability. Thus, in a nutshell, a rational agent must select the decision that maximizes its expected utility.

Decision theory was initially developed in economics and operations research [11], but in recent years has attracted the attention of artificial intelligent (AI) researchers interested in understanding and building intelligent agents. These intelligent agents, such as robots, financial advisers, intelligent tutors, etc., must deal with similar problems such as those encountered in economics and operations research, but with two main differences. One difference has to do with the size of the problems, which in artificial intelligence tend to be much larger, in general, than in traditional applications in economics. The other main difference has to do with knowledge about the problem domain. In many AI applications a model is not known in advance, and could be difficult to obtain.

10.2.1 Fundamentals

The principles of decision theory were initially developed in the classic text by Von Neuman and Morgensten, *Theory of Games and Economic Behavior* [11]. They established a set of intuitive constraints that should guide the preferences of a rational agent, which are known as the axioms of utility theory. Before we list these axioms, we need to establish some notation. In a decision scenario there are four elements:

Alternatives: Are the choices that the agent has and are under his control. Each decision has at least two alternatives (e.g., to do or not do some action).

Events: Are produced by the environment or by other agents; they are outside of the agent's control. Each random event has at least two possible results, although we do not know in advance which result will occur, we can assign a probability to each one.

Outcomes: Are the results of the combination of the agent's decisions and the random events. Each possible outcome has a different preference (utility) for the agent.

Preferences: These are established according to the agent's goals and objectives and are assigned by the agent to each possible outcome. They establish a value for the agent for each possible result of its decisions.

In utility theory, the different scenarios are called *lotteries*. In a lottery each possible outcome or *state*, A, has a certain probability, p, and an associated preference to the agent which is quantified by a real number, U. For instance, a lottery L with two possible outcomes, A with probability p, and B with probability $1 - p$, will be denoted as:

$$L = [p, A; 1 - p, B]$$

If an agent prefers A rather than B it is written as $A \succ B$, and if it is indifferent between both outcomes it is denoted as $A \sim B$. In general a lottery can have any number of outcomes; an outcome can be an atomic state or another lottery.

Based on these concepts, we can define utility theory in an analogous way as probability theory by establishing a set of reasonable constraints on the preferences for a rational agent; these are the **axioms of utility theory**:

Order: Given two states, an agent prefers one or the other or it is indifferent between them.

Transitivity: If an agent prefers outcome A to B and prefers B to C, then it must prefer A to C.

Continuity: If $A \succ B \succ C$, then there is some probability p such that the agent is indifferent between getting B with probability one, or the lottery $L = [p, A; 1 - p, C]$.

Substitutability: If an agent is indifferent between two lotteries A and B, then the agent is indifferent between two more complex lotteries that are the same except that B is substituted for A in one of them.

Monotonicity: There are two lotteries that have the same outcomes, A and B. If the agent prefers A, then it must prefer the lottery in which A has higher probability.

Decomposability: Compound lotteries can be decomposed into simple ones using the rules of probability.

Then, the definition of a utility function follows from the axioms of utility.

Utility Principle: If an agent's preferences follow the axioms of utility, then there is a real-valued utility function U such that:

1. $U(A) \succ U(B)$ if and only if the agent prefers A over B,
2. $U(A) = U(B)$ if and only if the agent is indifferent between A and B.

Maximum Expected Utility Principle: The utility of a lottery is the sum of the utilities of each outcome multiplied by its probability:

$$U[P_1, S_1; P_2, S_2; P_3, S_3; ...] = \sum_j P_j U_j$$

Based on this concept of a utility function, we can now define the expected utility (EU) of a certain decision D taken by an agent, considering that there are N possible results of this decision, each with probability P:

$$EU(D) = \sum_{j=1}^{N} P(result_j(D))U(result_j(D))$$

The principle of **Maximum Expected Utility** states that a rational agent should choose an action that maximizes its expected utility.

10.2.1.1 Utility of Money

In many cases it seems natural to measure utility in monetary terms; the more money we make based on our decisions, the better. Thus, we can think of applying the maximum expected utility principle measuring utility in terms of its monetary value. But this is not as straightforward as it seems.

Suppose that you are participating in a game, such as those typical TV shows, and that you have already won one million dollars. The host of the game asks you if you want to keep what you have already won and finish your participation in the game, or continue to the next stage and gain $3,000,000. Instead of asking some difficult question, the host will just flip a coin and if it lands on *heads* you will get three million, but if it lands on *tails* you will loose all the money you have already won. You have to make a decision with two options: (D1) keep the money you have already won, (D2) go to the next stage, with a possibility of winning three million (or loosing everything). What will your decision be?

Let us see what the principle of maximum expected utility will advise us if we measure utility in dollars (known as *Expected Monetary Value* or EMV). We calculate the EMV for both options:

$$\text{D1: } \text{EMV(D1)} = 1 \times \$1{,}000{,}000 = \$1{,}000{,}000$$
$$\text{D2: } \text{EMV(D2)} = 0.5 \times 0 + 0.5 \times \$3{,}000{,}000 = \$1{,}500{,}000$$

Thus, it seems that if we want to maximize the expected utility in terms of dollars we must take the bet. However, most of us would probably select to keep the one million that we already have and not take the risk of loosing it! What is the reasoning behind this? Are we not being *rational*?

The relation between utility and monetary value is not linear for most people; instead they have a logarithmic relation which denotes *risk aversion* (see Fig. 10.1). It is approximately linear for low values of money (for instance, if we have $10 instead of $1,000,000 on the line, we will probably go for the bet); but once we have a large amount of money (the amount will depend on each individual), the increase in utility given more money is no longer linear.

The utility–monetary value relation varies from person to person (and organizations) depending on their perception of risk; there are three basic types: risk aversion, risk neutral, and risk seeking; these are depicted in Fig. 10.1.

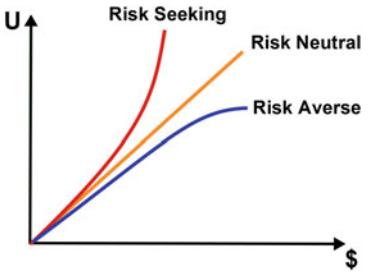


Fig. 10.1 The graphs show typical relations between utility (U) and monetary value ($). *Top*: risk seeking, *middle*: neutral, *bottom*: risk averse

Although it seems straightforward to apply the principle of maximum expected utility to determine the best decision, as the decision problems become more complex, involving several decisions, events and possible outcomes, it is not as easy as it seems and a systematic approach is required to model and solve complex decision problems. One of the earliest modeling tools developed for solving decision problems are **decision trees** [1].

10.3 Decision Trees

A decision tree is a graphical representation of a decision problem, which has three types of elements or nodes that represent the three basic components of a decision problem: decisions, uncertain events, and results.

A *decision node* is depicted as a rectangle which has several *branches*, each branch represents each of the possible alternatives present at this decision point. At the end of each branch there could be another decision point, an event, or a result.

An *event node* is depicted as a circle, and also has several branches, each branch represents one of the possible outcomes of this uncertain event. These outcomes correspond to all the possible results of this event, that is, they should be mutually exclusive and exhaustive. A probability value is assigned to each branch, such that the sum of the probabilities for all the branches is equal to one. At the end of each branch there could be another event node, a decision node or a result.

The *results* are annotated with the utility they express for the agent, and are usually at the end of each branch of the tree (the leaves).

Decision trees are usually drawn from left to right, with the root of the tree (a decision node) at the extreme left, and the leaves of the tree to the right. An example of a hypothetical decision problem (based on an example in [6]) is shown in Fig. 10.2. It represents an investment decision with 3 alternatives: (i) Stocks, (ii) Gold, and (iii) No investment. Assuming that the investment is for one year, if we invest in stock, depending on how the stock market behaves (uncertain event), we could gain $1000 or loose $300, both with equal probability. If we invest in gold, we have to make another decision, to have insurance or not. If we get insurance, then we are sure to gain $200; otherwise we win or loose depending on if the price of gold is up, stable, or down; this is represented as another event. Each possible outcome has a certain value and probability assigned, as shown in Fig. 10.2. What should the investor decide?

To determine the best decision for each decision point, according to the maximum expected utility principle, we need to *evaluate* the decision tree. The evaluation of a decision tree consists in determining the values of both types of nodes, decision and event nodes. It is done from right to left, starting from any node that has only results for all its branches:

- The value of a decision node D is the maximum value of all the branches that emanate from it:

$$V(D) = max_j U(result_j(D)).$$

- The value of an event node E is the expected value of all the branches that emanate from it, obtained as the weighted sum of the result values multiplied by their probabilities:

$$V(E) = \sum_j P(result_j(E))U(result_j(E))$$

Following this procedure we can evaluate the decision tree of Fig. 10.2:

Event 1—Market Price: $V(E_1) = 1000 \times 0.5 - 300 \times 0.5 = 350$.
Event 2—Gold Price: $V(E_2) = 800 \times 0.7 + 100 \times 0.2 - 200 \times 0.1 = 560$.
Decision 2—Insurance: $V(D_2) = max(200, 560) = 560$ – No insurance.
Decision 1—Investment: $V(D_1) = max(150, 560, 0) = 560$ – Invest in Gold.

Thus, in this case the best decisions are to invest in gold without insurance.

Decision trees are a tool for modeling and solving sequential decision problems, as decisions have to be represented in sequence as in the previous example. However, the size of the tree (number of branches) grows exponentially with the number of decision and event nodes, so this representation is practical only for *small* problems. An alternative modeling tool is the *Influence Diagram* [3, 8], which provides a compact representation of a decision problem.

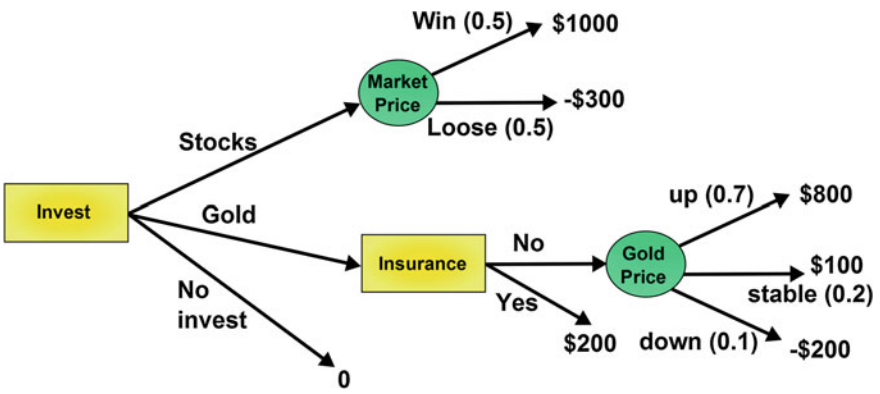


Fig. 10.2 An example of a decision tree (see text for details)

10.4 Influence Diagrams

Influence Diagrams (IDs) are a tool for solving decision problems that were introduced by Howard and Matheson [3] as an alternative to decision trees to simplify modeling and analysis. From another perspective, we can view IDs as an extension of Bayesian networks that incorporates decision and utility nodes. In the following sections we present a brief introduction to IDs, including their representation and basic inference techniques.

10.4.1 Modeling

An influence diagram is a directed acyclic graph, G, that contains nodes that represent random, decision, and utility variables:

Random nodes (X): represent random variables as in BNs, with an associated CPT. These are represented as ovals.

Decision nodes (D): represent decisions to be made. The arcs pointing toward a decision node are *informational*; that is, it means that the random or decision node at the origin of the arc must be known before the decision is made. Decision nodes are represented as rectangles.

Utility nodes (U): represent the costs or utilities associated to the model. Associated to each utility node there is a function that maps each permutation of its parents to a utility value. Utility nodes are represented as diamonds. Utility nodes can be divided into *ordinary* utility nodes, whose parents are random and/or decision nodes; and *super-value* utility nodes, whose parents are ordinary utility nodes. Usually, the super-value utility node is the (weighted) sum of the ordinary utility nodes.

There are three types of arcs in an ID:

Probabilistic: they indicate probabilistic dependencies, pointing toward random nodes.

Informational: they indicate information availability, pointing toward decision nodes. That is, $X \rightarrow D$ indicates that value of X is known before the decision D is taken.

Functional: they indicate functional dependency, pointing toward utility nodes.

An example of an ID is depicted in Fig. 10.3, which gives a simplified model for the problem of determining the location of a new airport, considering that the probability of accidents, the noise level and the estimated construction costs are the factors that directly affect the *utility*.

In an ID there must be a directed path in the underlying directed graph that includes all the decision nodes, indicating the order in which the decisions are made. This order induces a partition on the random variables in the ID, such that if there are n decision variables, the random variables are partitioned into $n + 1$ subsets. Each

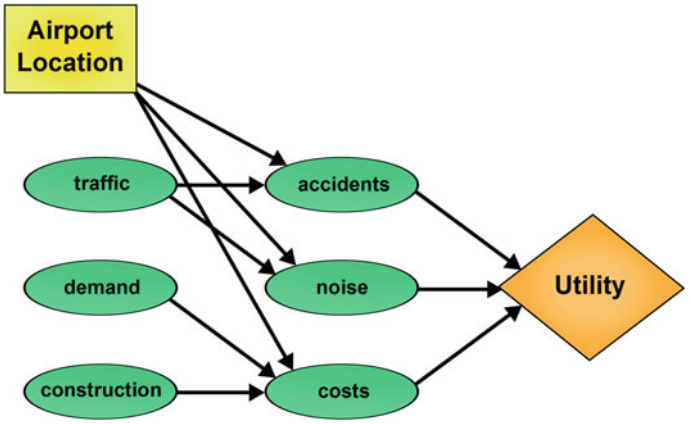


Fig. 10.3 A simplified model of the airport location problem represented as a simple ID. The decision node represents the different options for the location of a new airport, and the utility node represents the utility (or cost) which depends on several factors, which in turn depend on other random variables

subset, $\mathbf{R_i}$, contains all the random variables that are known before decision D_i and unknown for previous decisions. Some of the algorithms for evaluating influence diagrams take advantage of these properties to make the evaluation more efficient.

IDs are used to aid a decision-maker in finding the decisions that maximize its expected utility. That is, the goal in decision analysis is to find an *optimal policy*, $\pi = \{d_1, d_2, ..., d_n\}$, which selects the best decisions for each decision node to maximize the expected utility, $E_\pi(U)$. If there are several utility nodes, in general we consider that we have additive utility, so we will maximize the sum of these individual utilities:

$$E_\pi(U) = \sum_{u_i \in U} E_\pi(u_i) \tag{10.1}$$

10.4.2 Evaluation

Evaluating an influence diagram is finding the sequence of best decisions or optimal policy. First, we will see how we can solve a simple influence diagram with only one decision; then we will cover general techniques for solving IDs.

We define a *simple* influence diagram as one that has a single decision node and a single utility node. For this case we can simple apply BN inference techniques to obtain the optimal policy following this algorithm:

1. For all $d_i \in D$:
   a. Set $D = d_i$.
   b. Instantiate all the known random variables.

c. Propagate the probabilities as in a BN.
d. Obtain the expected value of the utility node, U.

2. Select the decision, d_k, that maximizes U.

For more complex decision problems in which there are several decision nodes, the previous algorithm becomes impractical. In general, there are three main types of approaches for solving IDs:

- Transform the ID to a decision tree and apply standard solution techniques for decision trees.
- Solve the ID directly by variable elimination, applying a series of transformations to the graph.
- Transform the ID to a Bayesian network and use BN inference techniques.

Next, we describe the second and third approaches.

10.4.2.1 Variable Elimination

The variable elimination algorithm [8] is based on evaluating the decision nodes one by one according to a certain order. Decision nodes that have been evaluated can be eliminated from the model, and this process continues until all the decision nodes have been evaluated. To apply this technique the influence diagram must be *regular*; that is, it satisfies the following conditions:

1. The structure of the ID is a directed acyclic graph.
2. The utility nodes do not have successors.
3. There is a directed path in the underlying directed graph that includes all the decision nodes, indicating the order in which the decisions are made.

In general, to evaluate the decision nodes, it is necessary to perform a series of transformations to the ID; these transformations are guaranteed to preserve the optimal series of decisions or *optimal policy*. The possible transformations are the following:

- Eliminate *barren* nodes, random or decision nodes that are leaf nodes in the graph—they do not affect the decisions.
- Eliminate random nodes that are parents of the utility node and do not have other children—the utility is updated according to the value of the node (if the node is not instantiated, then the expected utility is calculated).
- Eliminate decision nodes that are parents of the utility node where their parents are also a parent to the utility node—evaluate the decision node and take the decision that maximizes the expected utility; modifying accordingly the utility function.
- In case none of the previous operations can be applied, invert an arc between two random variables. To invert an arc between nodes i and j it is required that there be no other trajectory between these nodes. Then the arc $i \rightarrow j$ is inverted and each node inherits the parents of the other node.

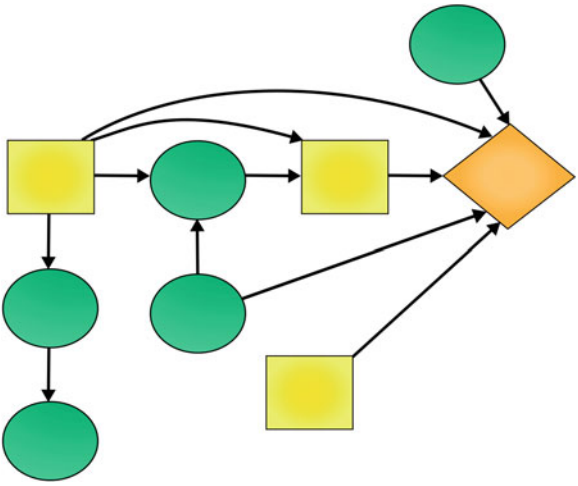


Fig. 10.4 An example of the variable elimination algorithm: initial influence diagram

We illustrate graphically the variable elimination algorithm with an example. Assume that we initially have the ID depicted in Fig. 10.4. We can eliminate the barren random node at the bottom left; when this node is eliminated, its parent also becomes a barren node, which is also eliminated and we obtain the ID shown in Fig. 10.5.

Next we eliminate the random node on the top, parent of the utility node *absorbing* it in the utility, and we then evaluate the first decision, that is, the decision node at the bottom, resulting in the ID seen in Fig. 10.6.

Then we invert the arc between the two remaining random nodes, so we can eliminate the bottom random node, obtaining the model shown in Fig. 10.7. From this graph we can evaluate the decision node on the right, then eliminate the remaining random node, and finally evaluate the decision node to the left.

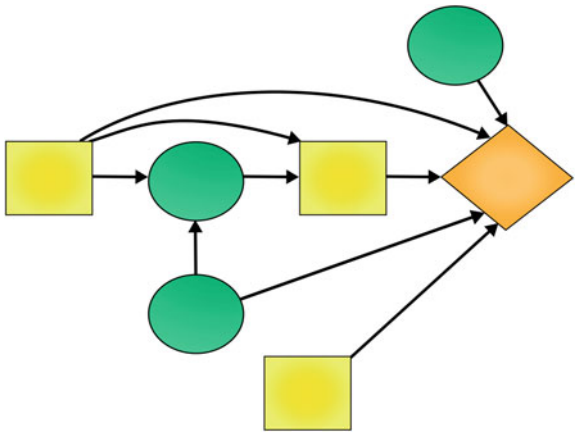


Fig. 10.5 An example of the variable elimination algorithm: after eliminating two barren nodes

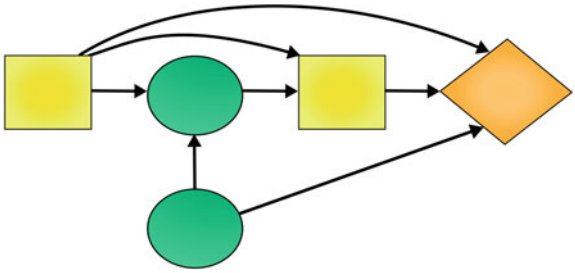


Fig. 10.6 An example of the variable elimination algorithm: after the evaluation of the first decision

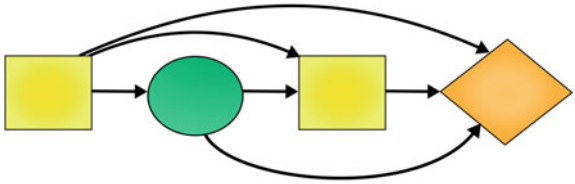


Fig. 10.7 An example of the variable elimination algorithm: after arc inversion and elimination of a random node

Next, we describe an alternative evaluation technique for IDs that take advantage of the efficient inference algorithms that have been developed for BNs.

10.4.2.2 Transformation to a BN

The idea to reduce an ID to a BN was originally proposed by Cooper [2]. To transform an ID to a BN, the basic idea is to transform decision and utility nodes to random nodes, with an associated probability distribution. A decision node is converted to a discrete random variable by considering each decision, d_i, as a value for this variable, and using a uniform distribution as a CPT (a decision node has no parents as all incoming arcs are informational). A utility node is transformed to a binary random variable by *normalizing* the utility function so it is in the range from 0 to 1, that is:

$$P(u_i = 1 \mid Pa(u_i)) = val(Pa(u_i))/maximum(val(Pa(u_i))) \tag{10.2}$$

where $Pa(u_i)$ are the parents of the utility node in the ID, and *val* is the value assigned to each combination of values of the parent nodes.

After the previous transformation, and considering a single utility node, the problem of finding the optimal policy is reduced to finding the values of the decision nodes that maximize the probability of the utility node: $P(u = 1 \mid D, R)$, where D is the set of decision nodes, and R is the set of the other random variables in the ID. This probability can be computed using standard inference techniques for BNs; however, it will require an exponential number of inference steps, one for each permutation of D.

Given that in a *regular* ID the decision nodes are ordered, a more efficient evaluation can be done by evaluating the decisions in (inverse) order [9]. That is, instead of maximizing $P(u = 1 \mid D, R)$, we maximize $P(D_j \mid u = 1, R)$. We can recursively optimize each decision node, D_j, starting from the last decision, continuing with the previous decision, and so on, until we reach the first decision. This gives a much more efficient evaluation procedure. Additional improvements have been proposed to the previous algorithms based on *decomposable* IDs (see the additional readings section).

Traditional techniques for solving IDs make two important assumptions:

Total ordering: all the decisions follow a total ordering according to a directed path in the graph.
Non-forgetting: all previous observations are remembered for future decisions.

These assumptions limit the applicability of IDs to some domains, in particular temporal problems that involve several decisions at different times. In some domains, such as in medical decision-making, a total ordering of the decisions is an unrealistic assumption since there are situations in which the decision-maker does not know in advance what decision should be made first to maximize the expected utility. For a system that evolves over a large period of time, the number of observations grows linearly with the passing of time, so the non-forgetting requirement implies that the size of policies grows exponentially.

10.4.3 Extensions

10.4.3.1 Limited Memory Influence Diagrams

In order to avoid the previous limitations of IDs, Lauritzen and Nilsson [5] proposed limited memory influence diagrams (LIMIDs) as an extension of influence diagrams. The term limited memory reflects the property that a variable known when making a decision is not necessarily remembered when making a posterior decision. Eliminating some variables reduces the complexity of the model so it is solvable with a computer, although at the price of obtaining a suboptimal policy.

10.4.3.2 Dynamic Decision Networks

Another extension is applied for sequential decision problems that involve several decisions over time. As with BNs, we can consider decision problems in which a series of decisions have to be taken at different time intervals; this type of problem is known as a *sequential decision problem*. A sequential decision problem can be modeled as a *dynamic decision network* (DDN)—also known as a *dynamic influence diagram* which can be seen as an extension of a DBN, with additional decision and utility nodes for each time step, see Fig. 10.8.

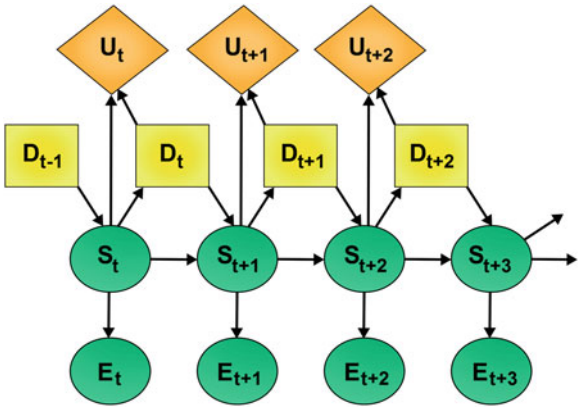


Fig. 10.8 An example of a dynamic decision network with 4 decision epochs

In principle, we can evaluate a DDN in the same way as an ID, considering that the decisions have to be ordered in time. That is, each decision node D_t has incoming informational arcs from all previous decision nodes, D_{t-1}, D_{t-2}, etc. However, as the number of time epochs increases, the complexity increases and can become computationally intractable. Additionally, in some applications we do not know in advance the number of decision epochs, so in principle there might be an *infinite* number of decisions.

DDNs are closely related to *Markov decision processes* which are the topic of the next chapter.

10.5 Applications

The application of decision graphs is exemplified with a system for assisting elderly or handicapped persons in washing their hands.

10.5.1 Decision-Theoretic Caregiver

The objective of the *caregiver* is to guide a person in completing a task using an adequate selection of prompts. We consider the particular task of *cleaning one's hands*. The system acts as a caregiver that guides an elderly or handicapped person in performing this task correctly [7].

The relevant objects in the washstand environment are: soap, faucet, and towel. The system detects the behavior of the user when interacting with these objects. Then it chooses an action (we use audible prompts) to guide the user to complete a task, or it may simply say nothing (null action) if the user is performing the sequence of steps required correctly.

10.5.1.1 Model

A DDN is used to model the user's behavior and make the optimal decisions at each time step based on the user's behavior (observations) and the objectives of the system (utilities). As optimal actions might involve evaluating the model *many* steps in advance until the task of washing the hands is completed, a compromise between optimality and efficiency is obtained by analyzing k steps in advance (lookahead). For this, the optimal decisions for k decision nodes are obtained by solving the DDN with one of the techniques presented previously in this chapter. In this case the DDN was transformed to a DBN and solved using BN inference.

The DDN for this scenario is illustrated in Fig. 10.9. In this model, the state variables (S) represent the activity of the user at each time step, which are not directly observable. The observation nodes (O) represent the information obtained from a vision system used to recognize the activity being performed by the user. The action nodes (A) correspond to the different actions (prompts) that can be selected by the controller at each time. Finally, the rewards (R) represent the immediate reward that depends on the current state and preferred action. These elements of the model are described in detail below.

States. The state space is characterized by the activities (hand gestures) carried out by a person. In this case, the state variable has 6 possible values: s_1 = *opening the faucet*, s_2 = *closing the faucet*, s_3 = *using the soap*, s_4 = *drying the hands*, s_5 = *taking the towel*, and s_6 = *washing the hands*.

Observations. These correspond to the information obtained by a visual gesture recognition system [7] that tries to recognize the activity performed by the person while washing their hands. The observation values are the same as the state values.

Actions. Actions are audible prompts to help the person complete the task. There are 8 actions that correspond to the possible prompts considered by the system:

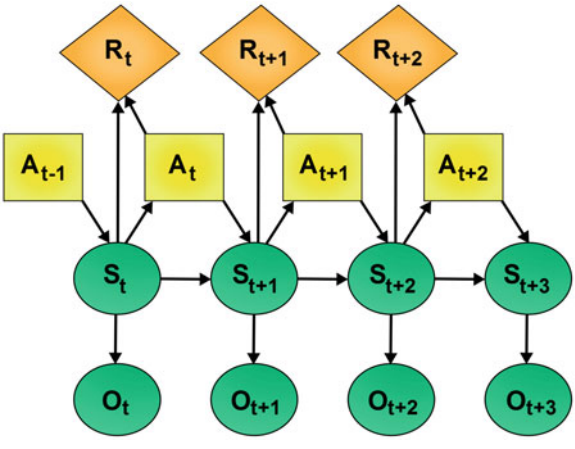


Fig. 10.9 A 4 stage dynamic decision network that models the caregiver scenario. S_t represents the activity of the user, O_t corresponds to the observed activity, A_t is the action selected, and R_t the immediate reward

$a_1 =$ *open the faucet*, $a_2 =$ *close the faucet*, $a_3 =$ *put soap on the hands*, $a_4 =$ *wash the hands*, $a_5 =$ *take the towel*, $a_6 =$ *dry the hands*, $a_7 =$ *Null*, and $a_8 =$ *call for help*.

Rewards. Rewards are associated with the preferences of the different actions selected by the system. In the caregiver setting, we must consider the prompts, clarity of the prompts and user's response. Three different reward values were used: $+3$ indicates a preference, -3 indicates a penalty, and -6 is used for selecting the action *call for help*. The idea is that asking for help should be the last option.

Additionally, the model requires two conditional probability tables: the transition function, $P(S_{t+1} \mid S_t)$ and the observation function, $P(O_t \mid S_t)$:

Transition function. The transition function defines the probability of the next state (next gesture) given the current state and action. In this setting this is predictable, with some degree of uncertainty. The transition functions were defined subjectively.

Observation function. In this case, the observation function consists of the probability of the observed gesture given the state, that is, the *actual* gesture. It can be easily obtained from the confidence (confusion matrix) of the gesture recognition system.

It is assumed that the model is time invariant, that is, that the previous CPTs do not change over time.

10.5.1.2 Evaluation

The model was evaluated in terms of its: (i) sensitivity relative to the number of stages or lookahead, (ii) efficiency in terms of the time required to solve the model for selecting the next action, and (iii) performance, comparing the actions selected by the system with a human caregiver.

The DDN was solved with 2 to 11 stages, and the actions selected under different scenarios were compared for the different model sizes. The expected utility increases as the lookahead is increased, tending to stabilize after 6 or 7 stages. However, the selected actions do not vary after a lookahead of 4, so this value was selected.

The model with 4 times stages (4 decision nodes) was evaluated in terms of its response time, and it could be solved in about 3 seconds with a standard personal computer. Thus, this number of stages provides a good compromise between performance and efficiency.

To evaluate the quality of the actions selected by the system, its decisions were compared to those of a human performing the same task. A preliminary evaluation was done with normal persons simulating that they had problems washing their hands. Ten adults participated in the experiment, divided in two groups, 5 each (test and control). The first group was guided to complete the task of washing their hands by verbal prompts given by the system, the control group was guided by verbal instructions given by a human assistant. The aspects evaluated in a questionnaire given to each participant were: (i) clarity of the prompt, (ii) detail of the prompt,

Table 10.1 Results obtained in the caregiver setting, where a person simulating memory problems is guided to complete the activity of cleaning their hands

	Human	System
Clarity	4.4	3.9
Detail	4.6	3.6
Effectiveness	4.2	3.6

It compares the decisions of a human assistant and the decision-theoretic system based on a DDN. The evaluation scale is from 1 (worst) to 5 (best)

and (iii) effectiveness of the system. Detail of the prompt refers to the level of specificity. Clarity considers the user's ease of understanding the message. Effectiveness evaluates the system's guidance for the successful completion of the task.

The results obtained are summarized in Table 10.1. The results indicate a small advantage when the best prompt is selected by a human, however, the difference between the system and human controller is not considerable. Two aspects show a small difference (0.6 or less), and one shows a more significant difference (detail of the prompt). This last aspect has to do with the verbal phrase recorded to be associated with each prompt, which could be easily improved; and not so much with the decisions of the system.

10.6 Additional Reading

The original reference on influence diagrams is the book by Howard and Matheson [3]. The book by Jensen includes decision networks [4]. Decision trees are described in [1]. The elimination algorithm for evaluating influence diagrams is presented in [8]; the alternative evaluation technique based on a transformation to a BN is described in [2]. Limited memory influence diagrams are introduced in [5] and extended to dynamic models in [10].

10.7 Exercises

1. Define a function for utility in terms of monetary value considering risk aversion, such that the optimal decision for the example of Sect. 10.2.1.1 is to keep the money (D1) when calculating the expected utility for both possible decisions.
2. Consider the decision tree in Fig. 10.2. The futures on the stock market have changed, and now the potential gain has increased to $3000. Also, the insurance price for gold has increased to $600. After applying these changes to the decision tree, reevaluate it. Have the decisions changed with respect to the original example?
3. Define the required CPTs for the influence diagram of Fig. 10.3. Consider two possible airport locations, Loc_A and Loc_B, and that all the random variables are

binary: $traffic = \{low, high\}$, $demand = \{no, yes\}$, $construction = \{simple, complex\}$, $accidents = \{very - low, low\}$, $noise = \{low, medium\}$, $costs = \{medium, high\}$. Define the parameters according to your intuition and following the axioms of probability.

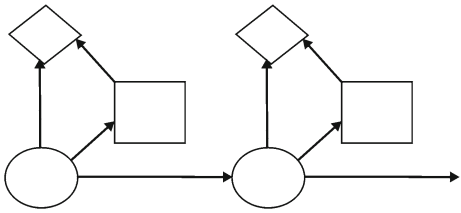
4. For the ID of Fig. 10.3 define a utility function in terms of the accidents, noise and costs, using the same values as in the previous exercise: (a) Define it as a mathematical function, $U = f(accidents, noise, costs)$. (b) Define it as a table.
5. Based on the parameters and utilities of the previous two exercises, evaluate the ID of Fig. 10.3 by calculating the utility function for each possible location, considering there is no evidence. Which is the best location according to the model?
6. Repeat the previous exercise for different scenarios, for instance: $traffic = low$, $demand = no$, $construction = simple$; and $traffic = high$, $demand = yes$, $construction = complex$. Does the optimal location change under the different scenarios?
7. Consider the problem of deciding whether to take an umbrella according to the weather forecast. There are two decisions for this problem: watch the weather forecast (no, yes) and take the umbrella (no, yes); and one random variable: $weather = \{sunny, light - rain, heavy - rain\}$. Model this problem as a decision tree, establishing costs/utilities for the different scenarios, i.e.: take the umbrella and sunny, take the umbrella and light rain, do not take the umbrella and heavy rain, etc.
8. Define an influence diagram for the umbrella decision problem of the previous exercise.
9. *** Develop a program that transforms an ID to a BN. Then use probabilistic inference to evaluate the BN. Apply it to solve the airport location model using the parameters and utilities defined in the previous exercises.
10. *** Investigate the procedure used to transform an ID to a decision tree. Apply it to solve the airport location model.

References

1. Cole, S., Rowley, J.: Revisiting decision trees. Manag. Decis. **33**(8), 46–50 (1995)
2. Cooper, G.: A method for using belief networks as influence diagrams. In: Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI), pp. 55–63 (1988)
3. Howard, R., Matheson, J.: Influence diagrams. In: Howard, R., Matheson, J. (eds.) Readings on the Principles and Applications of Decision Analysis. Strategic Decisions Group, Menlo Park (1984)
4. Jensen, F.V.: Bayesian Networks and Decision Graphs. Springer, New York (2001)
5. Lauritzen, S., Nilsson, D.: Representing and solving decision problems with limited information. Manag. Sci. **47**, 1235–1251 (2001)
6. Ley-Borrás, R.: Análisis de Incertidumbre y Riesgo para la Toma de Decisiones. Comunidad Morelos, Orizaba, Mexico (2001) (In Spanish)
7. Montero, A., Sucar, L.E.: Decision-theoretic assistants based on contextual gesture recognition. Ann. Inf. Syst. (to be published)

8. Shachter, R.D.: Evaluating influence diagrams. Oper. Res. **34**(6), 871–882 (1986)
9. Shachter, R.D., Peot, M.: Decision making using probabilistic inference methods. In: Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence (UAI), pp. 276–283 (1992)
10. van Gerven, M.A.J., Díez, F.J., Taal, B.G., Lucas, P.J.F.: Selecting treatment strategies with dynamic limited-memory influence diagrams. Artif. Intell. Med. **40**(3), 171–186 (2007)
11. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, Princeton (1944)

Chapter 11
Markov Decision Processes



11.1 Introduction

In this chapter, we will see how to solve sequential decision problems; that is, those that involve a series of decisions over time. It is assumed that the decision agent is rational, so the objective is to maximize the expected utility for the long term. Considering that there is uncertainty in the results of the agent's decisions, these type of problems can be modeled as Markov decision processes (MDPs). By solving the MDP model, we obtain what is known as a *policy*, which indicates to the agent which action to select at each time step based on its current state; the optimal policy is the one that selects the actions so that the expected value is maximized.

We will first formalize the MDP model, and then present two standard ways to solve it: value iteration and policy iteration. Although the complexity of solving an MDP is quadratic in terms of the number of state–actions, it could still become impractical in terms of memory and time when the number of state–actions is too large. Factored MDPs provide a representation based on graphical models to solve very large MDPs.

Finally, we will introduce partially observable MDPs (POMDPs), in which there is not only uncertainty in the results of the actions but also in the state.

11.2 Modeling

A Markov decision process (MDP) [13] models a sequential decision problem, in which a system evolves over time and is controlled by an agent. The system dynamics are governed by a probabilistic transition function Φ that maps states **S** and actions **A** to new states **S'**. At each time, an agent receives a reward R that depends on the current state s and the applied action a. By solving an MDP representation of the problem, we obtain a recommendation strategy or *policy* that maximizes the expected reward over time and that also deals with the uncertainty of the effects of an action.

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_11

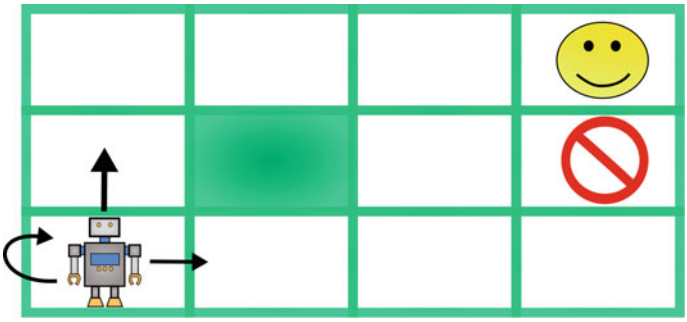


Fig. 11.1 A robot in the grid world. Each cell represents the possible states of the robot, with a smiling face for the goal and a forbidden sign for danger. The robot is shown in a cell with the *arrows* illustrating the probability for the next state given the action *up*

For example, consider an agent (a simulated robot) that *lives* in a grid world, so the state of the robot is determined by the cell where it is; see Fig. 11.1. The robot wants to go to the goal (cell with a smiling face) and avoid the obstacles and dangers (filled cells and a forbidden sign). The robot's possible actions are to move to the neighboring cells (up, down, left, right). We assume that the robot receives a certain immediate reward when it passes through each cell, for instance $+100$ when it arrives to the goal, -100 if it goes to the forbidden cell (this could represent a dangerous place), and -1 for all the other cells (this will motivate the robot to find the shortest route to the goal).

Consider that there is uncertainty in the result of each action taken by the robot. For example, if the selected action is *up* the robot goes to the upper cell with a probability of 0.8 and with probability of 0.2 to other cells (left and right). This is illustrated with the width of the arrows in Fig. 11.1. Having defined the states, actions, rewards, and transition function, we can model this problem as an MDP.

The objective of the robot is to go to the goal cell as fast as possible and avoid the dangers. This will be achieved by solving the MDP that represents this problem, and maximizing the expected reward.[1] The solution will provide the agent with a policy, that is, what is the best action to perform in each state, as illustrated graphically (by the arrows) for this small example in Fig. 11.2.

Formally, an MDP is a tuple $M =< S, A, \Phi, R >$, where S is a finite set of states $\{s_1, \ldots, s_n\}$. A is a finite set of actions $\{a_1, \ldots, s_m\}$. $\Phi : A \times S \times S \rightarrow [0, 1]$ is the state transition function specified as a probability distribution. The probability of reaching state s' by performing action a in state s is written as $\Phi(a, s, s')$. $R : S \times A \rightarrow \Re$ is the reward function. $R(s, a)$ is the reward that the agent receives if it takes action a in state s.

Depending on how much into the future (horizon) we consider there are two main types of MDPs: (i) finite horizon and (ii) infinite horizon. Finite horizon problems consider that there exists a fixed, predetermined number of time steps for which

[1] This assumes that the defined reward function correctly models the desired objective.

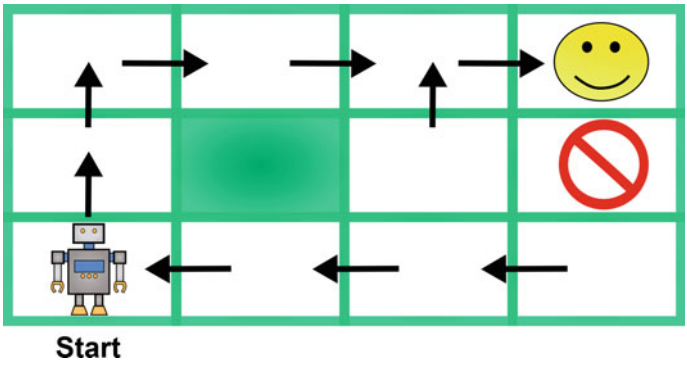


Fig. 11.2 The policy resulting from solving the MDP model is illustrated, with an *arrow* for each cell indicating the optimal action

we want to maximize the expected reward (or minimize the cost). For example, consider an investor that buys or sells actions each day (time step) and wants to maximize his profit for a year (horizon). Infinite horizon problems do not have a fixed, predetermined number of time steps, these could vary and in principle could be infinite. This is the case for the robot planning problem in general, as initially the number of movements (actions) that the robot will require to reach its goal or goals is unknown.

In the rest of this chapter, we will focus on infinite horizon problems, as these are more common in practice. They also have the advantage that under certain conditions the optimal policy is stationary; that is, it only depends on the state and not on the time step.

A *policy*, π, for an MDP is a function $\pi : S \rightarrow A$ that specifies for each state, s_i, the action to be executed, a_i. Given a certain policy, the expected accumulated reward for a certain state, s, is known as the *value* for that state according to the policy, $V^{\pi}(s)$; it can be calculated using the following recursive equation:

$$V^{\pi}(s) = R(s,a) + \sum_{s \in \mathbf{S}} \Phi(a,s,s') V^{\pi}(s') \tag{11.1}$$

where $R(s,a)$ represents the immediate reward given action a, and $\sum_{s \in \mathbf{S}} \Phi(a,s,s')$ $V^{\pi}(s')$ is the expected value of the next states according to the chosen policy.

For the infinite horizon case, a parameter known as the *discount factor*, $0 \leq \gamma < 1$, is included so that the sum converges. This parameter can be interpreted as giving more value to the rewards obtained at the present time than those obtained in the future.[2]

[2]This has an obvious value in the case of financial investments, related to the inflation or interest rates. For other applications, there usually is not a clear way to determine the discount factor, and in general, a value close to one, such as 0.9, is used.

Including the discount factor, the value function is written as:

$$V^{\pi}(s) = R(s,a) + \gamma \sum_{s \in \mathbf{S}} \Phi(a,s,s') V^{\pi}(s') \tag{11.2}$$

What is desired is to find the policy that maximizes the expected reward; that is, the policy that gives the highest value for all states. For the discounted infinite-horizon case with any given discount factor γ, there is a policy π^* that is optimal regardless of the starting state and that satisfies what is known as the *Bellman* equation [2]:

$$V^{\pi}(s) = max_a \left\{ R(s,a) + \gamma \sum_{s \in \mathbf{S}} \Phi(a,s,s') V^{\pi}(s') \right\} \tag{11.3}$$

The policy that maximizes the previous equation is then the optimal policy, π^*:

$$\pi^*(s) = argmax_a \left\{ R(s,a) + \gamma \sum_{s \in \mathbf{S}} \Phi(a,s,s') V^{\pi}(s') \right\} \tag{11.4}$$

The *Bellman* equation is a recursive equation that cannot be solved directly. However, there are several methods to solve it efficiently; these will be covered in the next section.

11.3 Evaluation

There are three basic methods for solving an MDP and finding an optimal policy: (a) value iteration, (b) policy iteration, and (c) linear programming [13]. The first two techniques solve the problem iteratively, improving an initial value function or policy, respectively. The third one transforms the problem to a linear program which can then be solved using standard optimization techniques such as the simplex method. We will cover the first two approaches; for more on the third one see the additional reading section.

11.3.1 Value Iteration

Value iteration starts by assigning an initial value to each state; usually this value is the immediate reward for that state. That is, at iteration 0, the $V_o(s) = R(a,s)$. Then these estimates of the values are improved in each iteration by maximizing using the *Bellman* equation. The process is terminated when the value for all states *converges*, this is when the difference between the values in the previous and current iterations is

less than a predefined threshold. The actions selected in the last iteration correspond to the optimal policy. The method is shown in Algorithm 11.1.

Algorithm 11.1 The Value Iteration Algorithm.

1: $\forall_s V_0(s) = R(s,a)$ {Initialization}
2: $t = 1$
3: **repeat**
4: $\quad \forall_s V_t(s) = max_a\{R(s,a) + \gamma \sum_{s \in \mathbf{S}} \Phi(a,s,s')V_{t-1}(s')\}$ {Iterative improvement}
5: **until** $\forall_s \mid V_t(s) - V_{t-1}(s) \mid < \varepsilon$
6: $\pi^*(s) = argmax_a\{R(s,a) + \gamma \sum_{s \in \mathbf{S}} \Phi(a,s,s')V_t(s')\}$ {Obtain optimal policy}

The time complexity of the algorithm is quadratic in terms of the number of state–actions.

Usually, the policy converges before the values converge; this means that there is no change in the policy even if the value has not yet converged. This gives rise to the second approach, policy iteration.

11.3.2 Policy Iteration

Policy iteration starts by selecting a random, initial policy (if we have certain domain knowledge, this can be used to seed the initial policy). Then the policy is iteratively improved by selecting the action for each state that increases the most the expected value. The algorithm terminates when the policy converges, that is, the policy does not change from the previous iteration. The method is shown is Algorithm 11.2.

Algorithm 11.2 The Policy Iteration Algorithm.

1: $\pi_0 : \forall_s a_0(s) = a_k$ {Initialize the policy}
2: $t = 1$
3: **repeat**
4: $\quad$ {Iterative improvement}
5: $\quad \forall_s V_t^{\pi_{t-1}}(s) = \{R(s,a) + \gamma \sum_{s \in \mathbf{S}} \Phi(a,s,s')V_{t-1}(s')\}$ {Calculate values for the current policy}
6: $\quad \forall_s \pi_t(s) = argmax_a\{R(s,a) + \gamma \sum_{s \in \mathbf{S}} \Phi(a,s,s')V_t(s')\}$ {Iterative improvement}
7: **until** $\pi_t = \pi_{t-1}$

Policy iteration tends to converge in fewer iterations than value iteration, however the computational cost of each iteration is higher, as the values have to be updated.

Solving *small* MDPs with the previous algorithms is very efficient; however it becomes difficult when the state–actions space is very large. Consider, for instance, a problem with 10,000 states and 10 actions, which is common in applications such as robot navigation. In this case, the space required to store the transition table will be $10{,}000 \times 10{,}000 \times 10 = 10^9$; and updating the value function will

require in the order of 10^8 operations per iteration. So solving very large MDPs could become problematic even with current computer technology. An alternative is to decompose the state space and take advantage of the independence relations to reduce the memory and computation requirements, using a graphical model-based representation of MDPs known as *Factored* MDPs.

11.4 Factored MDPs

In a factored MDP, the set of states is described via a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each X_i takes on values in some finite domain $Dom(X_i)$. A state $\mathbf{s}$ defines a value $x_i \in Dom(X_i)$ for each variable X_i. The transition model and reward function can become exponentially large if they are explicitly represented as matrices; however, the frameworks of dynamic Bayesian networks (see Chap. 7) and decision trees [14] give us the tools to describe the transition model and the reward function concisely.

Let X_i denote a variable at the current time and X_i' the variable at the next step. The transition function for each action, a, is represented as a two–stage dynamic Bayesian network, that is a two–layer directed acyclic graph G_T whose nodes are $\{X_1, \ldots, X_n, X_1', \ldots, X_n'\}$; see Fig. 11.3 (left). Each node X_i' is associated with a *conditional probability distribution* $P_\Phi(X_i' \mid Parents(X_i'))$, which is usually represented by a matrix (*conditional probability table*) or more compactly by a decision tree. The transition probability $\Phi(a, s_i, s_i')$ is then defined to be $\Pi_i P_\Phi(x_i' \mid \mathbf{u_i})$ where $\mathbf{u_i}$ represents the values of the variables in $Parents(X_i')$.

The reward associated with a state often depends only on the values of certain features of the state. The relationship between rewards and state variables can be represented with value nodes in an influence diagrams, as shown in Fig. 11.3 (center). The conditional reward table (CRT) for such a node is a table that associates a reward with every combination of values for its parents in the graph. This table is exponential in the number of relevant variables. Although in the worst case the CRT will take exponential space to store the reward function, in many cases the reward function exhibits structure allowing it to be represented compactly using decision trees or graphs, as shown in Fig. 11.3 (right).

In many cases, the conditional probability tables (CPTs) in the DBN exhibit a certain structure; in particular some probability values tend to be repeated many times (such as a zero probability). Taking advantage of these properties, the representation can be compacted even more by representing a CPT as a tree or a graph, such that repeated probability values appear only once in the leaves of these graphs. A particular presentation that is very efficient is an *algebraic decision diagram* or ADD. An example of a CPT represented as an ADD is depicted in Fig. 11.4.

The representation of the transition functions in MDPs as two-stage DBNs, and the reward function as a DT with the further reduction of these based on trees or ADDs, implies, in many cases, huge savings in memory for storing very large MDPs. An example of this will be shown in the applications section of this chapter.

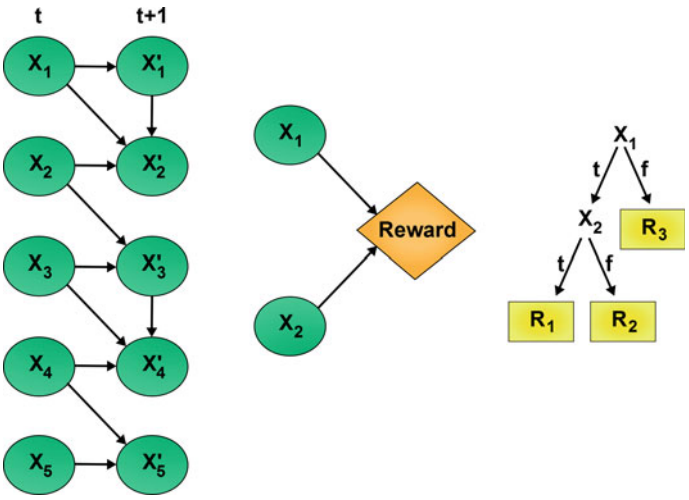


Fig. 11.3 *Left* A DBN with five state variables that represents the transition function for one action. *Center* Influence diagram denoting a reward function. *Right* Structured conditional reward (CR) represented as a binary decision tree

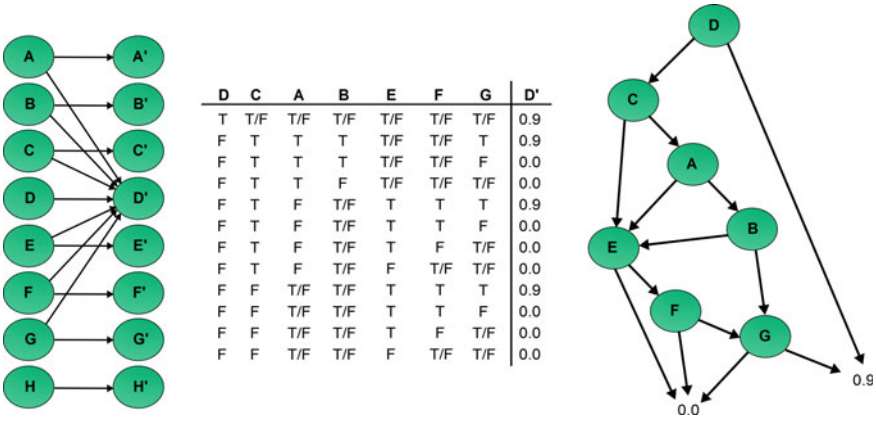


Fig. 11.4 An example of a CPT represented as an algebraic decision diagram. *Left* two-stage DBN representing the transition function. *Center* CPT for one of the variables. *Right* ADD representation for the CPT—for each node, the *right arrow* corresponds to the value T and the *left arrow* to the value F

Additionally, based on this compact representation, very efficient versions of the value and policy iteration algorithms have been developed that also reduce the computational time required to solve complex MDP models. An example of this is the SPUDD algorithm [8].

Further reduction in computational complexity can be achieved using other techniques, such as *abstraction* and *decomposition*, which are summarized below.

11.4.1 Abstraction

The idea of abstraction is to reduce the state space by creating an abstract model where states with similar features are grouped together [9].

Equivalent states are those that have the same transition and reward functions; these can be grouped together without altering the original model, so that the optimal policy will be the same for the reduced model. However, the state space reduction achieved by only joining equivalent states is in general not significant; further reductions can be achieved by grouping *similar* states; this results in approximate models and creates a tradeoff between the precision of the model (and the resulting policy) and its complexity.

Different strategies have been used to create reduced, approximate models. One is to partition the state space into a set of blocks such that each block is stable; that is, it preserves the same transition probabilities as the original model [5]. Another alternative is to partition the state space into *qualitative* states that have similar reward functions [15].

11.4.2 Decomposition

Decomposition consists in dividing the global problem into smaller subproblems that are solved independently and their solutions combined [4, 10]. There are two main types of decomposition: (i) serial or hierarchical, and (ii) parallel or concurrent.

Hierarchical MDPs provide a sequential decomposition, in which different subgoals are solved in sequence to reach the final goal. That is, at the execution phase, only one task is active at a given time. Hierarchical MDPs accelerate the solution of complex problems by defining different subtasks that correspond to intermediate goals, solving for each subgoal, and then combining these subprocesses to solve the overall problem; examples of hierarchical approaches are HAM [11] and MAXQ [6].

In concurrent or parallel MDPs, the subtasks are executed in parallel to solve the global task. In general, these approaches consider that the task can be divided in several relatively independent subtasks that can be solved independently and then the solutions combined to solve the global problem. When the subtasks are not completely independent some additional considerations are required. For instance, Loosely Coupled MDPs [10] consider several independent subprocesses which are coupled due to common resource constraints. To solve them they use an iterative procedure based on a heuristic allocation of resources to each task. An alternative approach is taken by [4], which initially solves each subtask independently, and when the solutions are combined, they take into account potential conflicts between the partial policies, and solve these conflicts to obtain a global, approximately optimal policy. An example of the application of this last technique in robotics is given below in Sect. 11.6.

11.5 Partially Observable Markov Decision Processes

In some domains, the state cannot be observed completely, there is only partial information about the state of the system; these type of problems are known as *partially observable Markov decision processes* (POMDPs). In this case, there are certain observations from which the state can be estimated probabilistically. For instance, consider the previous example of the robot in the grid world. It could be that the robot can not determine precisely the cell where it is (its state), but can estimate the probability of being in each cell by observing the surrounding environment. Such is the case in real mobile robot environments, where the robot cannot know with precision its localization in the environment, only probabilistically by using its sensors and certain *landmarks*.

Formally, a POMDP is a tuple $M =< S, A, \Phi, R, O, \Omega, \Pi >$. The first four elements are the same as in an MDP. O is a finite set of observations $\{o_1, \ldots, o_l\}$. $\Omega : S \times O \rightarrow [0, 1]$ is the observation function specified as a probability distribution, which gives the probability of an observation o given that the process is in state s, $P(o \mid s)$. Π is the initial state distribution that specifies the probability of being in state s at $t = 0$.

In a POMDP, the current state is not known with certainty, only the probability distribution of the state, which is known as the *belief state*. So solving a POMDP requires finding a mapping form the belief space to the action space, such that the optimal action is selected. This is equivalent to a continuous state space MDP. Thus, solving a POMDP is much more difficult than solving an MDP, as the belief space is in principle infinite.

Solving a POMDP exactly is computationally intractable except for very small problems. As a result, several approximate solution techniques have been developed. Some of the main types of approaches for finding approximate solutions to a POMDP are the following:

Policy tree and DDN techniques: assuming a finite horizon, a POMDP can be represented as a *policy tree* (similar to a decision tree) or as a dynamic decision network; then, algorithms for solving these types of models can be applied.

Sampling techniques: the value function is computed for a set of points in the belief space, and interpolation is used to determine the optimal action to take for other belief states which are not in the set of sampling points.

Value function approximation techniques: given that the value function is convex and piece-wise linear, it can be described as a set of vectors (called α vectors). Thus, it can be approximated by a set of vectors that *dominate* the others, and in this way find an approximate solution.

The additional reading section gives pointers to more information on POMDPs.

11.6 Applications

The application of MDPs is illustrated in two different domains. One is for assisting power plant operators in the operation of a power plant under difficult situations. The other is for coordinating a set of modules to solve a complex task for service robots.

11.6.1 Power Plant Operation

The steam generation system of a combined-cycle power plant provides superheated steam to a steam turbine. It is composed by a recovery steam generator, a recirculation pump, control valves, and interconnection pipes. A *heat recovery steam generator* (*HRSG*) is a process machinery capable of recovering residual energy from a gas turbine's exhaust gases to generate high-pressure (*Pd*) steam in a special tank (*steam drum*). The *recirculation pump* is a device that extracts residual water from the steam drum to keep a water supply in the HRSG (*Ffw*). The result of this process is a high-pressure steam flow (*Fms*) that keeps running a *steam turbine* to produce electric energy (*g*) in a *power generator*. The main control elements associated are the feedwater valve (*fwv*) and the main steam valve (*msv*).

During normal operation, a three-element feedwater control system commands the feedwater control valve (*fwv*) to regulate the level (*dl*) and pressure (*pd*) in the drum. However, this traditional controller does not consider the possibility of failures in the control loop (valves, instrumentation, or any other process devices). Furthermore, it ignores whether the outcomes of executing a decision will help, in the future, to increase the steam drum lifetime, security, and productivity. The problem is to obtain a function that maps plant states to recommendations for the power plant operator; this function should consider all these aspects.

This problem was modeled as an MDP, which served as the basis for developing a tool for training and assistance for power plant operators.

11.6.1.1 Power Plant Operator Assistant

AsistO [16] is an intelligent assistant that provides recommendations for training and online assistance in the power plant domain. The assistant is coupled to a power plant simulator capable of partially reproducing the operation of a combined-cycle power plant.

AsistO is based on a decision-theoretic model that represents the main elements of the steam generation system of a combined-cycle power plant. The main variables in the steam generation system represent the state in a factored form. Some of these variables are continuous, so they are discretized into a finite number of intervals. The actions correspond to the control of the main valves in this subsystem of the power plant, in particular those that have to do with controlling the level of the

drum (a critical element of the plant): feedwater valve (*fwv*) and main steam valve (*msv*). The reward function is defined in terms of a recommended operation curve for the relation between the drum pressure and steam flow (see Fig. 11.5). The idea is to maintain a balance between the efficiency and safety of the plant. As such, the control actions should try to maintain the plant within this recommended operation curve; if it deviates they should return it to a *safe* point; this is shown schematically with arrows in Fig. 11.5.

The last element to be defined to complete the MDP model is the transition function. In this application, it can be learned by using the power plant simulator and sampling the state and action spaces. Once the MDP is built, it can be solved to obtain the optimal policy, and from this, the system can give recommendations to the operator under different plant conditions.

11.6.1.2 Experimental Results

A relatively simple example was considered with five state variables: Fms, Ffw, Pd, g, d; and four actions: open/close the feedwater (*fwv*) and main steam (*msv*) valves a certain amount. The reward function was defined based on the recommended operation curve. To learn the transition function, samples of the system dynamics were gathered using simulation.

The memory requirements for a *flat* MDP representation and a factored representation were compared. The flat MDP required 589,824 parameters (probability values) while the factored MDP only 758. The optimal solution for the factored MDP was obtained in less than two minutes on a standard personal computer.

The recommended actions of the MDP controller and the traditional automatic control were compared using the power plant simulator. The actions taken by both are similar, however the MDP-based controller takes less time to return the plant to a safe operating point when a disturbance occurs.

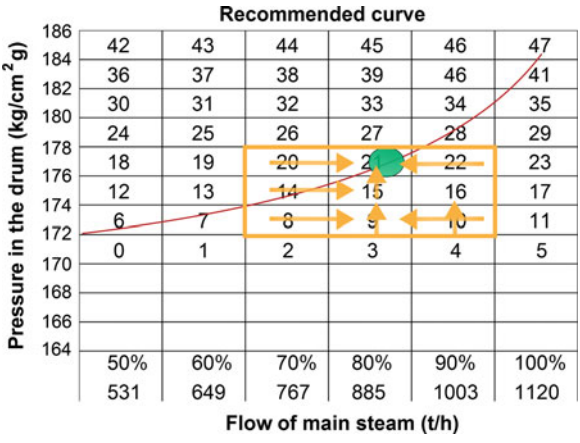


Fig. 11.5 Recommended operation curve for the drum of a combined-cycle power plant, defining the desirable relation between drum pressure and steam flow. The *arrows* illustrate schematically the idea of the recommender system—returning the operation to a safe relation (*circle*) when the plant deviates from the recommended operation curve

11.6.2 Robot Task Coordination

Service robots are, in general, mobile robots developed for assisting humans in different activities, such as helping a person to clean his home, assisting senior citizens in nursing homes or taking medicines to a patient in a hospital. To perform these types of tasks, a service robot should combine several capabilities, such as localization and navigation, obstacle avoidance, people detection and recognition, object recognition and manipulation, etc. To simplify the development of service robots and promote reutilization, these different capabilities can be implemented as independent software modules, which can then be *combined* for solving a particular task, such as delivering messages or objects between people in an office environment. In this case, it is necessary to coordinate the different modules to perform a task, ideally in an optimal way.

Markov decision processes provide an appropriate framework for task coordination for service robots [7]. The state space can be defined in terms of a number of variables that define the high-level situation of the tasks. The actions correspond to commands (calls) to the different software modules, for instance indicating to the navigator to move the robot to a certain position in the environment. The reward function can be defined based on the objectives of the task. For example, for a message delivery robot, a certain reward for when it receives a message from the sender, and another, higher reward, when it delivers it to the recipient. Once a task is modeled as an MDP, the MDP can be solved to obtain a policy to perform the task. This is in general better than a traditional plan, as it provides a *plan* for any initial state (a kind of general plan) and it is robust with respect to the uncertainty in the results of the different actions.

Under this framework, based on general software modules and an MDP-based coordinator, it is in principle relatively easy for a service robot to solve different tasks. We just need to modify the MDP reward function according to the new task objectives, and solve the modified MDP to obtain a policy for the other task.

Additionally, it is desirable for the robot to perform several actions *simultaneously*, such as navigation to a certain location, avoiding obstacles and looking for people; all at the same time. However, if we represent the robot task coordination problem as a single MDP, we have to consider all possible combinations of all the possible simultaneous actions. This implies an explosion in the action–state space and thus an important increase in the complexity for solving the MDP. It also becomes much more difficult to specify or learn the model.

An alternative is to model each subtask as an independent MDP, then solve each MDP to obtain its optimal policy, and then execute these policies *concurrently*. This approach is known as *concurrent MDPs* [4].

11.6.2.1 Concurrent Markov Decision Processes

Based on functional decomposition, a complex task is partitioned into several subtasks. Each subtask is represented as an MDP and solved independently, and the policies are executed in parallel assuming no conflicts. All the subtasks have a common goal and can share part of the state space, that is represented in a factored form. However, conflicts may arise between the subtasks.

There are two main types of conflicts: (i) *resource conflicts*, and (ii) *behavior conflicts*. Resource conflicts occur when two actions require the same physical resource (e.g., to control the wheels of a robot) and cannot be executed concurrently. This type of conflict is solved offline by a two-phase process [3]. In the first phase, we obtained an optimal policy for each subtask (MDP). An initial global policy is obtained by combining the local policies, such that if there is a conflict between the actions selected by each MDP for a certain state, the one with maximum value is considered, and the state is marked as a *conflict* state. This initial solution is improved in a second phase using policy iteration. By taking the previous policy as its initial policy and considering only the states marked as conflicts, the time complexity is drastically reduced and a near-optimal global policy is obtained.

Behavior conflicts arise in situations in which it is possible to execute two (or more) actions at the same time but it is not desirable given the application. For example, it is not desirable for a mobile robot to be navigating and handing an object to a person at the same time (this situation is also difficult for a person). Behavior conflicts are solved online based on a set of restrictions specified by the user. If there are no restrictions, all the actions are executed concurrently; otherwise, a constraint satisfaction module selects the set of actions with the highest expected utility.

11.6.2.2 Experiments

An experiment was done with *Markovito*, a service robot, which performed a delivery task; conflicts were considered. Markovito is a service robot based on an ActivMedia PeopleBot robot platform, which has laser, sonar, and infrared sensors; a camera, a gripper and two computers (see Fig. 11.6) [1].

In the task considered for Markovito, the goal is for the robot to receive and deliver a message, an object or both, under a user's request. The user gives an order to send a message/object and the robot asks for the name of the sender and the receiver. The robot either records a message or uses its gripper to hold an object, and navigates to the receiver's position for delivery. The task is decomposed into five subtasks, each represented as an MDP:

1. *navigation*, the robot navigates safely in different scenarios;
2. *vision*, for looking and recognizing people and objects;
3. *interaction*, for listening and talking with a user;
4. *manipulation*, to receive and deliver an object safely; and
5. *expression*, to show *emotions* using an animated face.

Fig. 11.6 Markovito, a service robot

Each subtask is represented as a factored MDP:

Navigation: States: 256 decomposed into 6 state variables (*located, has-place, has-path, direction, see-user, listen-user*). Actions: 4 (*go-to, stop, turn, move*).

Vision: States: 24 decomposed into 3 state variables (*user-position, user-known, user-in-db*). Actions: 3 (*look-for people, recognize-user, register-user*).

Interaction: States: 9216 decomposed into 10 state variables (*user-position, user-known, listen, offer-service, has-petition, has-message, has-object, has-user-id, has-receiver-id, greeting*). Actions: 8 (*hear, offer-service, get-message, deliver-message, ask-receiver-name, ask-user-name, hello, bye*).

Manipulation: States: 32 decomposed into 3 state variables (*has-petition, see-user, has-object*). Actions: 3 (*get-object, deliver-object, move-gripper*).

Expression: States: 192 decomposed into 5 state variables (*located, has-petition, see-user, see-receiver, greeting*). Actions: 4 (*normal, happy, sad, angry*).

Table 11.1 Restriction set for the messenger robot

Action(s)	Restriction	Action(s)
get message	**not_during**	turn OR advance
ask_user_name	**not_before**	recognize_user
recognize_user	**not_start**	avoid_obstacle
get_object		directed toward
OR	**not_during**	OR turn
deliver_object		OR moving

Note that several state variables are common to two or more MDPs. If we represent this task as a single, flat MDP, there are 1,179,648 states (considering all the non-duplicated state variables) and 1,536 action combinations, giving a total of nearly two billion state-actions. Thus, to solve this task as a single MDP will be difficult even for state of the art MDP solvers.

The MDP model for each subtask was defined using a structured representation [8]. The transition and reward functions were specified by the user based on task knowledge and intuition. Given that the MDP for each subtask is relatively simple, its manual specification is not too difficult. In this task, conflicts might arise between the different subtasks, so we need to include conflict resolution strategies. The conflicts considered are behavior conflicts, so these are solved based on a set of restrictions, which are summarized in Table 11.1.

For comparison, the delivery task was solved under two conditions: (i) without restrictions and (ii) with restrictions.

In the case without restrictions, all actions can be executed concurrently, but the robot performs some undesirable behaviors. For example, in a typical experiment, the robot is not able to identify the person who wants to send a message for a long time, after which it is able to complete the task. This is because the *vision MDP* cannot get a good image to analyze and recognize the user because the *navigation MDP* is moving trying to avoid the user. Also, because of this, the user has to follow the robot to provide an adequate input to the microphone.

In the case where restrictions were used, these allowed a more fluid and efficient solution. For example, in a similar experiment as without restrictions, the *vision MDP* with the restriction set is now able to detect and recognize the user much earlier. When the *interaction MDP* is activated, the *navigation MDP* actions are not executed, allowing an effective interaction and recognition.

On average, the version with restrictions takes about 50 % of the time steps required by the version without restrictions to complete the task. In summary, by considering conflicts via restrictions, not only does the robot perform the expected behavior, but it also has more robust performance, avoids conflicts, and displays a significant reduction in the time required to complete the task.

11.7 Additional Reading

Puterman [13] is an excellent reference for MDPs, including their formal definition and the different solution techniques. A recent overview of decision-theoretic models and their applications is given in [18]. Reference [12] reviews MDPs and POMDPs. The representation of MDPs using ADDs and SPUDD are described in [8]. A review of different approaches for solving POMDPs is presented in [17].

11.8 Exercises

1. For the grid world example of Fig. 11.1, consider that each cell is a state and that there are four possible actions: *up, down, left, right*. Complete the specification of the MDP, including the transition and reward functions.
2. Solve the MDP for the previous exercise by value iteration. Initialize the values for each state to the immediate reward, and show how the values change with each iteration.
3. Solve the grid world MDP using policy iteration. Initialize the policy to *up* for all states. Show how the policy changes with each iteration.
4. Define a factored representation for the grid world example, considering that a state is represented in terms of two variables, *row* and *column*. Specify the transition function as a two-stage DBN and the reward function as a decision tree.
5. Consider a grid world scenario in which the grid is divided in several rooms and a hallway that connects the rooms. Develop a hierarchical solution to the robot navigation for this scenario, considering one MDP to go from any cell in a room to the door that connects to the hallway, other MDP that navigates through the hallway from one door to another, and a third MDP to navigate from the door entering a room to the goal in the room. Specify each MDP and then a high-level controller that coordinates the low-level policies.
6. Develop a solution to the grid navigation problem based on two concurrent MDPs, one that navigates toward the goal (without considering obstacles) and another that avoids obstacles. Specify the model for each of these two subtasks and how to combine their resulting policies to navigate to the goal and avoid obstacles. Does the solution always reach the goal or can it get stuck in local maxima?
7. Prove that the solution to the Bellman equation using value iteration always converges.
8. *** Develop a general program that implements the value iteration algorithm.
9. *** Develop a program to solve the grid world robot navigation problem, including a graphical interface to define the size of the grid, the obstacles and the position of the goal. Considering a high positive reward for the goal cell, high negative rewards for the obstacles, and small negative rewards for the other cells,

represent the problem as an MDP and obtain the optimal policy using value iteration.

10. *** When the model for an MDP is not known, an alternative is to learn the optimal policy by trial and error with *reinforcement learning*. Investigate the basic algorithms for reinforcement learning, such as *Q-learning*, and implement a program to learn a policy to solve the robot navigation problem for the grid world example of Fig. 11.1 (use the same rewards as in the example). Is the policy obtained the same as the solution to the MDP model?

References

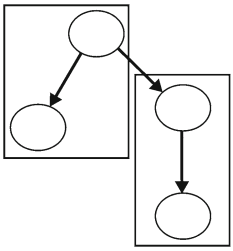
1. Avilés-Arriaga, H.H., Sucar, L.E., Morales, E.F., Vargas, B.A., Corona, E.: Markovito: a flexible and general service robot. In: Liu, D., Wang, L., Tan, K.C. (eds.) Computational Intelligence in Autonomous Robotic Systems, pp. 401–423. Springer, Berlin (2009)
2. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
3. Corona, E., Morales, E.F., Sucar, L.E.: Solving policy conflicts in concurrent Markov decision processes. In: ICAPS Workshop on Planning and Scheduling Under Uncertainty, Association for the Advancement of Artificial Intelligence (2010)
4. Corona, E., Sucar, L.E.: Task coordination for service robots based on multiple Markov decision processes. In: Sucar, L.E., Hoey, J., Morales, E. (eds.) Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions. IGI Global, Hershey (2011)
5. Dean, T., Givan, R.: Model minimization in Markov decision processes. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 106–111 (1997)
6. Dietterich, T.: Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. **13**, 227–303 (2000)
7. Elinas, P., Sucar, L., Reyes, A., Hoey, J.: A decision theoretic approach for task coordination in social robots. In: Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN), pp. 679–684 (2004)
8. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: SPUDD: stochastic planning using decision diagrams. In: Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI), pp. 279–288 (1999)
9. Li, L., Walsh, T.J., Littman, M.L.: Towards a unified theory of state abstraction for MDPs. In: Proceedings of the Nineth International Symposium on Artificial Intelligence and Mathematics, pp. 21–30 (2006)
10. Meuleau, N., Hauskrecht, M., Kim, K.E., Peshkin, L., Kaelbling, L.P., Dean, T., Boutilier, C.: Solving very large weakly coupled Markov decision processes. In: Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI), pp. 165–172 (1998)
11. Parr, R., Russell, S. J.: Reinforcement learning with hierarchies of machines. In: Proceeding of the Advances in Neural Information Processing Systems (NIPS) (1997)
12. Poupart, P.: An introduction to fully and partially observable Markov decision processes. In: Sucar, L.E., Hoey, J., Morales, E. (eds.) Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions. IGI Global, Hershey (2011)
13. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)
14. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
15. Reyes, A., Sucar, L.E., Morales, E.F., Ibargüngoytia, P.: Hybrid Markov Decision Processes. Lecture Notes in Computer Science, vol. 4293. Springer, Berlin (2006)

16. Reyes, A., Sucar, L.E., Morales, E.F.: AsistO: a qualitative MDP-based recommender system for power plant operation. Computacion y Sistemas **13**(1), 5–20 (2009)
17. Ross, S., Pineau, J., Paquet, S., Chaib-draa, B.: Online planning algorithms for POMDPs. J. Artif. Intell. Res. **32**, 663–704 (2008)
18. Sucar, L.E., Hoey, J., Morales, E.: Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions. IGI Global, Hershey (2011)

Part IV
Relational and Causal Models

The fourth and last part of the book describes two interesting extensions to probabilistic graphical models: relational probabilistic models and causal graphical models. Relational probabilistic models increase the representational power of *standard* PGMs, by combining the expressive power of first-order logic with the uncertain reasoning capabilities of probabilistic models. Causal graphical models go beyond representing probabilistic dependencies, to express cause and effect relations, based on the same framework of graphical models.

Chapter 12
Relational Probabilistic Graphical Models



12.1 Introduction

The *standard* probabilistic graphical models that have been covered until now, have to represent explicitly each object in the domain, so they are equivalent in terms of their logical expressive power to propositional logic. However, there are problems in which the number of objects (variables) could increase significantly, so a more expressive (compact) representation is desirable. Consider, for instance, that we want to model a student's knowledge of a certain topic (this is known as student modeling or, in general, user modeling), and that we want to include in the model all of the students in a college, where each student is enrolled in several topics. If we model this explicitly with a PGM such as a Bayesian network, it could become a huge model, difficult to acquire and store. Instead, it would be more efficient if in some way we could have a general model that represents the dependency relations for any student, S and any course, C, which could then be parameterized for particular cases. This can be done using predicate logic; however, standard logical representations do not consider uncertainty. Thus, a formalism is required that combines predicate logic and probabilistic models.

Relational probabilistic graphical models (RPGMs) combine the expressive power of predicate logic with the uncertain reasoning capabilities of probabilistic graphical models. Some of these models extend PGMs such as Bayesian networks or Markov networks by representing objects, their attributes, and their relations with other objects. Other approaches extend the logic-based representations, in order to incorporate uncertainty, by describing a probability distribution over logic formulas.

Different formalisms have been proposed to combine logic and PGMs. We propose a taxonomy for the classification of RPGMs (inspired and partially based on [5]):

1. Extensions of logic models
    a. Undirected graphical models
        i. Markov Logic Networks

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_12

   b. Directed graphical models
      i. Bayesian Logic Programs
      ii. Bayesian Logic Networks

2. Extensions of probabilistic models

   a. Undirected graphical models
      i. Relational Markov Networks
      ii. Relational Dependency Networks
   b. Directed graphical models
      i. Relational Bayesian Networks
      ii. Probabilistic Relational Models

3. Extensions of programming languages

   a. Stochastic Logic Programs
   b. Probabilistic Inductive Logic programming
   c. Bayesian Logic (BLOG)
   d. Probabilistic Modeling Language (IBAL)

In this chapter we will review two of them. One, *probabilistic relational models* [3], extends Bayesian networks to incorporate objects and relations, as in a relational database. The other are *Markov logic networks* [11], which add weights to logical formulas, and can be considered as an extension of Markov networks. First, a brief review of first-order logic is presented, then we describe the two relational probabilistic approaches, and finally we illustrate their application in two domains: student modeling and visual object recognition.

12.2 Logic

Logic is a very well-studied representation language with well-defined syntax and semantic components. Here we only include a concise introduction, for more extensive references see the additional reading section. We will start by defining propositional logic, and then go into first-order logic.

12.2.1 Propositional Logic

Propositional logic allows us to reason about expressions or *propositions* that are *True* (T) or *False* (F). For instance, *Joe is an engineering student*. Propositions are denoted with capital letters, such as *P, Q, …*, known as atomic propositions or *atoms*. Propositions are combined using logic connectives or operators, obtaining what are

known as *compound propositions*. The logic operators are:

- Negation: $\neg$
- Conjunction: $\wedge$
- Disjunction: $\vee$
- Implication: $\rightarrow$
- Double implication $\leftrightarrow$

For example, if $P =$ "Joe is an engineering student" and $Q =$ "Joe is young," then $P \wedge Q$ means "Joe is an engineering student AND Joe is young."

Atoms and connectives can be combined according to a set of syntactic rules; valid combinations are known as *well-formed formulas* (WFF). A well-formed formula in propositional logic is an expression obtained according to the following rules:

1. An atom is a WFF.
2. If P is a WFF, then $\neg P$ is a WFF.
3. If P and Q are WFFs, then $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, and $P \leftrightarrow Q$ are WFFs.
4. No other formula is a WFF.

For instance, $P \rightarrow (Q \wedge R)$ is a WFF; $\rightarrow P$ and $\vee Q$ are not WFFs.

The meaning (semantics) of a logical formula can be expressed by a function which gives a $True$ or $False$ value to the formula for each possible *interpretation* (truth values of the atoms in the formula). In the case of propositional logic, the interpretation can be represented as a *truth table*. Table 12.1 depicts the truth tables for the basic logic operators. An interpretation that assigns a truth value to a formula F is a $model$ of F.

A formula F is a *logical consequence* of a set of formulas $\mathbf{G} = \{G_1, G_2, \ldots, G_n\}$ if for every interpretation for which $\mathbf{G}$ is True, F is True. It is denoted as $G \models F$.

Propositional logic cannot express general properties, such as *all engineering students are young*. For this we need first-order logic.

12.2.2 First-Order Predicate Logic

Consider the two statements: "All students in a technical university are engineering majors" and "A particular person t of a particular university $\mathscr{I}$ is a computer science

Table 12.1 Truth tables for the logical connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

major." Whereas the first statement declares a property that applies to all persons in a technical university, the second applies only to a specific person in a particular university. First-order logic lets us deal with these differences.

Expressions or formulas in first-order predicate logic are constructed using four types of symbols: *constants, variables, functions*, and *predicates*. Constant symbols represent objects in the domain of interest (e.g., persons, courses, universities, etc.). Variable symbols range over the objects in the domain. Variables, which could be arguments for a predicate or function, are represented with lower case letters, i.e., *x, y, z*.

Predicates are expressions that can be True or False; they contain a number of arguments. Predicates can represent relations among objects in the domain (e.g., *Above*) or attributes of objects (e.g., *IsRed*). If the number of arguments is zero then it is an atom as in propositional logic. Predicates are represented with capital letters, for instance, $P(x)$.

Function symbols (e.g., *neighborOf*) represent mappings from tuples of objects to objects. They are represented with lower case letters and can also have a number of arguments. For example, $f(x)$ will represent a function with one argument, x. A function with no arguments is a constant.

Predicate logic includes the logical connectives of propositional logic, plus other operators known as *quantifiers*:

Universal quantifier: $\forall x$ (for all x).
Existential quantifier: $\exists x$ (exists an x).

A *term* is a constant, variable, or a function of terms. An *atomic formula* or *atom* is a predicate that has as arguments N terms.

Now we can rewrite the example at the beginning of the section in first-order logic. If the formula $M(t_x, \mathscr{I})$ represents the major of a person t_x in a technical university $\mathscr{I}$, the statement "All students in a technical university are engineering majors" can be rewritten as:

$$\forall t_x \forall \mathscr{I} : M(t_x, \mathscr{I}) = engineering. \tag{12.1}$$

Similarly to predicate logic, in first-order predicate logic there are a set of syntactic rules that define which expressions are well-formed formulas:

1. An atom is a WFF.
2. If P is a WFF, then $\neg P$ is a WFF.
3. If P and Q are WFFs, then $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, and $P \leftrightarrow Q$ are WFFs.
4. If P is a WFF and x is a free variable in P, then $\forall x\, P$ and $\exists x\, P$ are WFFs.
5. No other formula is a WFF.

Roughly speaking, a first-order knowledge base (*KB*) is a set of sentences or formulas in first-order logic [4].

An $\mathscr{L}$-interpretation specifies which objects, functions, and relations in the domain are represented by which symbols. Variables and constants may be typed, in which case variables range only over objects of the corresponding type, and constants can only represent objects of the corresponding type. For example, the variable x

might range over universities (e.g., public universities, private universities, etc.), and the constant might represent a university or a specific set of universities (e.g., u_2, $(u_6 \cup u_2)$, etc.). An *atomic formula* (or simply *atom*) is a predicate symbol applied to a tuple of terms: $Near(u_4, u_1)$.

In standard predicate logic all predicates are *True* or *False*, so it cannot deal with probabilistic uncertainty directly. For example, if we do not know if a person is in a certain university, we can only say that $Univ(p) = u_1$ OR $Univ(p) = u_2$; but we cannot specify if it is more probable that she is in u_1 than in u_2.

To have the expressive power of predicate logic and at the same time be able to represent and reason under uncertainty (in probabilistic terms) we need to combine predicate logic with probabilistic models under a single framework. Next we will describe two of these frameworks.

12.3 Probabilistic Relational Models

Probabilistic relational models (PRMs) [6] are an extension of Bayesian networks that provide a more expressive, object-oriented representation facilitating knowledge acquisition. They also make it easier to extend a model to other domains. For the case of a very large model, only part of it is considered at any time, so the inference complexity is reduced.

The basic entities in a PRM are objects or domain entities. Objects in the domain are partitioned into a set of disjoint classes $X_1, \ldots, X_n$. Each class is associated with a set of attributes $A(X_i)$. Each attribute $A_{ij} \in A(X_i)$ (that is, attribute j of class i) takes on values in some fixed domain of values $V(A_{ij})$. $X : A$ denotes the attribute A of an object in class X [6]. A set of relations, R_j, are defined between the classes. A binary relation $R(X_i, X_j)$ between classes X_i and X_j can be viewed as a *slot* of X_i. The classes and relations define the *schema* of the model. Then, a PRM defines a probability distribution over a set of instances of a schema; in particular, a distribution over the attributes of the objects in the model.

The dependency model is defined at the class level, allowing it to be used for any object in the class. PRMs explicitly use the relational structure of the model, so an attribute of an object will depend on some attributes of related objects. A PRM specifies the probability distribution using the same underlying principles used in Bayesian networks. Each of the random variables in a PRM, the attributes $x.a$ of the individual objects x, is directly influenced by other attributes, which are its parents. A PRM, therefore, defines for each attribute, a set of parents, which are the directed influences on it, and a local probabilistic model that specifies probabilistic parameters.

There are two basic differences between PRMs and Bayesian networks [6]: (i) In a PRM the dependency model is specified at the class level, allowing it to be used for any object in the class. (ii) A PRM explicitly uses the relational structure of the model, allowing an attribute of an object to depend on attributes of related objects.

An example of a PRM for the school domain, based on [6], is shown in Fig. 12.1. There are 4 classes, with 2 attributes each in this example:

Professor: teaching-ability, popularity
Student: intelligence, ranking
Course: rating, difficulty
Registration: satisfaction, grade

This representation allows for two types of attributes in each class: (i) information variables, (ii) random variables. The random variables are the ones that are linked in a kind of Bayesian network that is called a *skeleton*. From this skeleton, different Bayesian networks can be generated, according to other variables in the model. For example, in the student model described in Sect. 12.5, we define a general skeleton for an experiment, from which particular instances for each experiment are generated. This gives the model a greater flexibility and generality, facilitating knowledge acquisition. It also makes inference more efficient, because only part of the model is used in each particular case.

The probability distribution for the skeletons are specified as in Bayesian networks. A PRM, therefore, defines for each attribute $x.a$, a set of parents, which are the directed influences on it, and a local probabilistic model that specifies the conditional probability of the attribute given its parents. To guarantee that the local models define a coherent global probability distribution, the underlying dependency structure should be acyclic, as in a BN. Then, the joint distribution is equal to the product of the local distributions, similar to Bayesian networks.

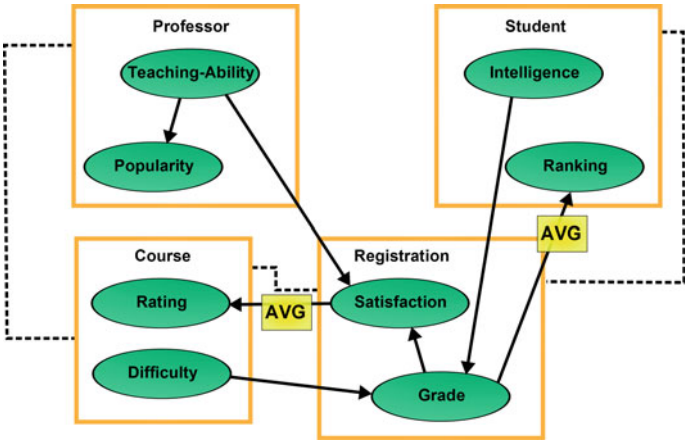


Fig. 12.1 An example of a PRM structure for the school domain. *Dashed edges* represent relations between classes, and arrows correspond to a probabilistic dependency. The *AVG* in a link indicates that the conditional probabilities depend on this variable

12.3.1 Inference

The inference mechanism for PRMs is the same as for Bayesian networks, once the model is instantiated to particular objects in the domain. However, PRMs can take advantage of two properties to make inference more efficient. These properties are made explicit by the PRM representation, in contrast to Bayesian networks where they are only implicit.

One property is the locality of influence, most attributes will tend to depend mostly on attributes of the same class, and there are few interclass dependencies. Probabilistic inference techniques can take advantage of this locality property by using a divide and conquer approach.

The other aspect which can make inference more efficient is reuse. In a PRM there are usually several objects of the same class, with similar structure and parameters. Then, once inference is performed for one object, this can be reused for the other similar objects.

12.3.2 Learning

Given that PRMs share the same underlying principles of BNs, the learning techniques developed for BNs can be extended for PRMs. The expectation maximization algorithm has been extended to learn the parameters of a PRM, and structure learning techniques have been developed to learn the dependency structure from a relational database [3].

12.4 Markov Logic Networks

In contrast to PRMs, Markov logic networks (MLN) start from a logic representation, adding *weights* to formulas to incorporate uncertainty.

In logic, a $\mathscr{L}$-interpretation which violates a formula given in a knowledge base (KB) has zero probability. It means that its occurrence is impossible, because all the *possible worlds* must be consistent with the KB. In Markov Logic Networks, this assumption is relaxed. If the interpretation violates the KB, it has *less* probability than others with no violations. Less probability means that it has a nonzero probability. In a MLN, a weight to each formula is added in order to reflect how strong the constraint is: higher weights entail higher changes in probability whether that interpretation satisfies that formula or not.

Before we formally define a MLN, we need to review Markov networks (see Chap. 6). A Markov network is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \mathscr{X}$. It is composed of an undirected graph G and a set of potential functions ϕ_k. The associated graph has a node for each variable, and the

model has a potential function for each clique in the graph. The joint distribution represented by a Markov network is given by

$$P(X = x) = \frac{1}{z} \prod_k \phi_k(x_{\{k\}}) \tag{12.2}$$

where $x_{\{k\}}$ is the state of the kth clique, and Z is the partition function.

Normally, Markov networks can also be represented using *log-linear* models, where each clique potential function is replaced by an exponential weighted sum:

$$P(X = x) = \frac{1}{z} \exp \sum_j w_j f_j(x) \tag{12.3}$$

where w_j is a weight (real value) and f_j is, for our purposes, a binary formula $f_j(x) \in \{0, 1\}$.

We now provide a formal definition for a MLN [1].

An MLN L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ Eqs. 12.2 and 12.3:

1. $M_{L,C}$ contains one binary node for each possible grounding of each formula appearing in the MLN L. The value of the node is 1 if the ground atom is true and 0 otherwise.
2. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

MLNs are a generalization of Markov networks, so they can be seen as templates for constructing Markov networks. Given a MLN and a set of different constants, different Markov networks can be produced; these are known as ground Markov networks. The joint probability distribution of a ground Markov network is defined in a similar way as a Markov network, using Eq. 12.3.

The graphical structure of a MLN is based on its definition; there is an edge between two nodes of the MLN if the corresponding ground atoms appear together in at least one grounding of one formula in the knowledge base, L. For example [1], consider the following MLN consisting of two logical formulas:

$$\forall x Smoking(x) \rightarrow Cancer(x)$$
$$\forall x \forall y Friends(x) \rightarrow (Smoking(x) \leftrightarrow Smoking(y))$$

If the variables x and y are instantiated to the constants A and B, we obtain the structure depicted in Fig. 12.2.

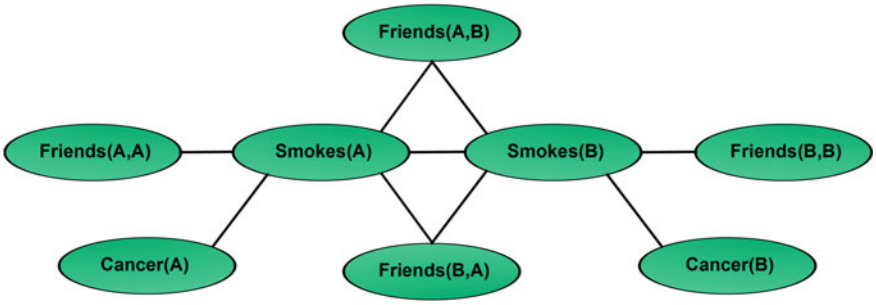


Fig. 12.2 Structure of the ground MLN obtained from the two logical formulas of the example. (Figure based on [1])

12.4.1 Inference

Inference in MLN consists on estimating the probability of a logical formula, F_1, given that another formula (or formulas), F_2, are true. That is, calculating $P(F_1 \mid F_2, L, C)$, where L is a MLN consisting of a set of weighted logical formulas, and C is a set of constants. To compute this probability, we can estimate the proportion of possible worlds in which F_1 and F_2 are true, over the possible worlds in which F_2 holds. For this calculation, the probability of each possible world is considered according to the weights of the formulas and the structure of the corresponding grounded Markov network.

Performing the previous calculations directly is, computationally, very costly; thus, it becomes prohibitive but for very small models. Several alternative probabilistic inference techniques can be used to make this computation more efficient. One alternative is using stochastic simulation; by sampling the possible worlds we can obtain an estimate of the desired probability; for instance, using the Markov chain Montecarlo techniques.

Another alternative is to make certain reasonable assumptions about the structure of the logical formulas that simplify the inference process. In [1], they develop an efficient inference algorithm for the case that F_1 and F_2 are conjunctions of ground literals.

12.4.2 Learning

Learning a MLN involves two aspects, just as for learning Bayesian networks. One aspect is learning the logical formulas (structure), and the other is learning the weights for each formula (parameters).

For learning the logical formulas, we can apply techniques from the area of inductive logic programming (ILP). In the area of ILP there are different approaches that

can induce logical relations from data, usually considering some background knowledge. For more information, see the additional reading section.

The weights of the logical formulas can be learned from a relational database. Basically, the weight of a formula is proportional to its number of true groundings in the data with respect to its expectation according to the model. Counting the number of true groundings of a formula could be too computationally expensive for large domains. An alternative is make an estimate based on sampling the groundings of a formula and verifying if they are true in the data.

An example of a MLN in the context of an application for visual object recognition is presented in Sect. 12.5.

12.5 Applications

We will illustrate the application of the two classes of RPGM presented in the previous sections in two different domains. First, we will see how we can build a kind of *general* student model for virtual laboratories based on PRMs. Then we will use MLNs for representing visual grammars for object recognition.

12.5.1 Student Modeling

A particularly challenging area for student modeling is virtual laboratories. A virtual lab provides a simulated model of some equipment, so that students can interact with it and learn by doing. A tutor serves as a virtual assistant in this lab, providing help and advice to the user, and setting the difficulty of the experiments, according to the student's level. In general, it is not desirable to trouble the student with questions and tests to update the student model. So the cognitive state should be obtained based solely on the student's interactions with the virtual lab and the results of the experiments. For this a student model is required. The model infers, from the student's interactions with the laboratory, the cognitive state; and based on this model, an intelligent tutor can give personalized advice to the student [13].

12.6 Probabilistic Relational Student Model

PRMs provide a compact and natural representation for student modeling. Probabilistic relational models allow each attending student to be represented in the same model. Each class represents the set of parameters of several students, like in databases, but the model also includes the probabilistic dependencies between classes for each student.

In order to apply PRMs to student modeling we have to define the main objects involved in the domain. A general student model oriented to virtual laboratories was designed, starting from a high-level structure at the class level, and ending with specific Bayesian networks for different experiments at the lower level. As shown in Fig. 12.3, the main classes, related with students and experiments, were defined. In this case there are 8 classes, with several attributes for each class, as listed below:

Student: student-id, student-name, major, quarter, category.
Knowledge Theme: student-id, knowledge-theme-id, knowledge-theme-known.
Knowledge Sub-theme: student-id, knowledge-sub-theme-id, knowledge-sub-theme-known.
Knowledge Items: student-id, knowledge-item-id, knowledge-item-known.
Academic background: previous-course, grade.
Student behavior: student-id, experiment-id, behavior-var1, behavior-var2, …
Experiment results: student-id, experiment-id, experiment-repetition, result-var1, result-var2, …
Experiments: experiment-id, experiment-description, exp-var1, exp-var2, …

The dependency model is defined at the class level, allowing it to be used for any object in the class. Some attributes in this model represent probabilistic values. This means than an attribute represents a random variable that is related to other attributes in the same class or in other classes.

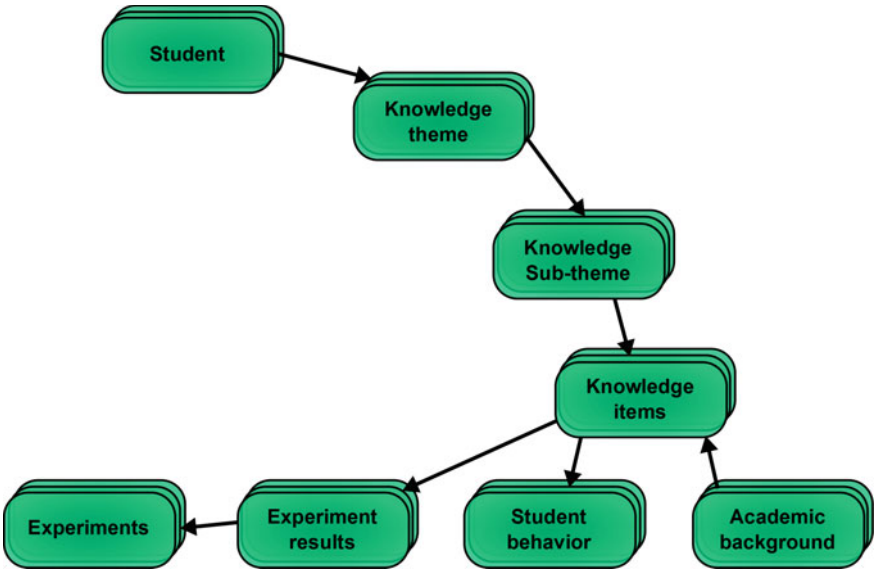


Fig. 12.3 A high-level view of the PRM structure for the student model, showing the main classes and their relations

From the PRM student model we can define a general Bayesian network, a skeleton, that can be instantiated for different scenarios, in this case experiments. In this model it is easy to organize the classes by levels to improve the understanding of the model. From the class model we obtain a hierarchical skeleton, as shown in Fig. 12.4. We partitioned the experiment class, according to our object of interest, creating two subclasses: experiment performance and experiment behavior, which constitute the lowest level in the hierarchy. The intermediate level represents the different knowledge items (concepts) associated to each experiment. These items are linked to the highest level which groups the items in sub-themes and themes, and finally into the students' general category. We defined three categories of students: *novice, intermediate, and advanced*. Each category has the same Bayesian net structure, obtained from the skeleton, but different CPTs are used for each one.

From the skeleton, it is possible to define different instances according to the values of specific variables in the model. For example, from the general skeleton for the experiments of Fig. 12.4, we can define particular instances for each experiment (for example, in the robotics domain, there could be experiments related to robot design, control, motion planning, etc.) and student level (novice, intermediate, advanced).

Once a specific Bayesian network is generated, it can be used to update the student model via standard probability propagation techniques. In this case, it is used to propagate evidence from the experiment evaluation to the knowledge items, and then to the knowledge sub-themes and to the knowledge themes. After each experiment, the knowledge level at each different level of granularity—items, sub-themes and

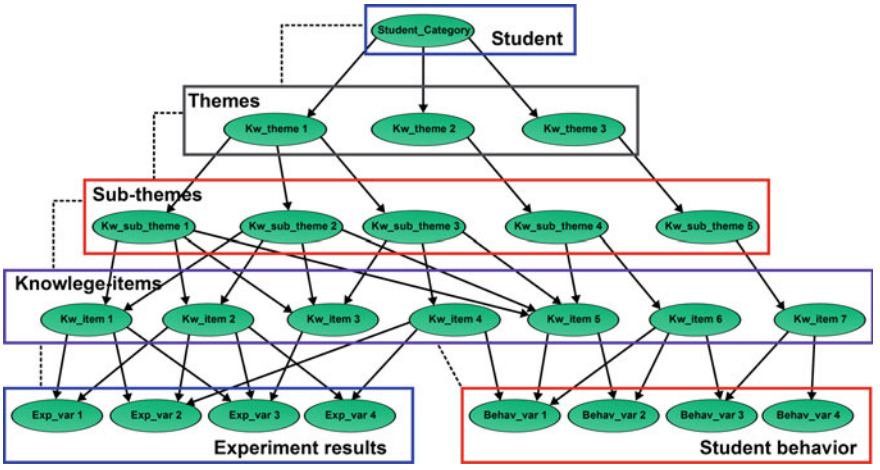


Fig. 12.4 A general skeleton for an experiment derived from the PRM student model for virtual laboratories. Basically, the network has a hierarchical structure, starting form a node that represents the student category at the *top*, and then with three layers of variables that represent the knowledge the student has of the domain at different levels of abstraction: themes, sub-themes, and items. At the bottom level there are two sets of variables that correspond to the results of the student's interaction with an experiment in the virtual lab, divided in experimental results and student behavior [13]

themes, is used by the tutor to decide if it should provide help to the student, and at what level of detail. For example, if in general the experiment was successful, but some aspect was not very good, a lesson on a specific concept (item) is given to the student. While if the experiment was unsuccessful, a lesson on a complete theme or sub-theme is recommended. Based on the student category, the tutor decides the difficulty of the next experiments to be presented to the student.

12.6.1 Visual Grammars

A visual grammar describes objects hierarchically. It can represent a diagram, a geometric drawing or an image. For example, the description of a flowchart is made by decomposition: complex elements are decomposed into simple elements (from the complete image to arrows or simple boxes).

For visual object recognition, we need a grammar that allows us to model the decomposition of a visual object into its parts and how they relate with each other [12]. One interesting kind of relational grammar are Symbol-Relation Grammars (*SR-grammars*) [2], because they can provide this type of description and also incorporate the possibility of adding rewritten rules to specify relationships between terminal and nonterminal symbols once a decomposition for all the nonterminal symbols has taken place.

12.6.1.1 Representing Object with SR Grammars

Classes of objects are represented based on symbol-relational grammars. This includes three basic parts: (i) the basic elements of the grammar or lexicon, (ii) the spatial relations, (iii) the transformation rules. Next, we briefly describe them.

The idea is to use simple and general features as basic elements so they can be applied to different classes of objects. These regions define a visual dictionary. The visual features considered are: *uniform color regions* (color is quantized in 32 levels) and edges at different orientations (obtained with Gabbor filters). These features form a set of training examples that are clustered and the centroids of these clusters constitute the *Visual Lexicon*.

The spatial relations include topological and order relationships. The relationships used are: *Inside_of* (A, B) (A region is *within* B region), *Contains*(A, B) (A region *covers* completely B region), *Left*(A, B) (A is *touched* by B and A is located *left* from B), *Above*(A, B) (A is *touched* by B and A is located *above* from B), *Invading*(A, B) (A is *partially covering* B more than *Above* and *Left* but less than *Contains*).

The next step is to generate the rules that make up the grammar. Using training images for the class of the object of interest, the most common relationships between clusters are obtained. Such relationships become candidate rules to build the grammar. This is an iterative process where the rules are subsumed and converted to new

nonterminal elements of the grammar. This process is repeated until a threshold (in terms of the minimum number of elements) is reached; the starting symbol of the grammar represents the class of objects to be recognized.

Visual object recognition involves uncertainty: noise in the image, occlusions, imperfect low-level processing, etc. SR grammars do not consider uncertainty, so to incorporate it, they can be extended using RPGMs, in particular MLNs.

12.6.1.2 Transforming a SR Grammar into a Markov Logic Network

The SR grammar for a class of objects is transformed *directly* to formulas in the MLN language. In this way the structural aspect of the MLN is obtained. The parameters—weights associated to each formula—are obtained from the training image set.

Consider a simple example, a SR grammar to recognize faces based on high-level features: eyes, mouth, nose, head. The productions of this simple SR grammar for faces are:

$$
\begin{aligned}
&1 : FACE^0 \rightarrow < \{eyes^2, mouth^2\}, \{above(eyes^2, mouth^2)\} > \\
&2 : FACE^0 \rightarrow < \{nose^2, mouth^2\}, \{above(nose^2, mouth^2)\} > \\
&3 : FACE^0 \rightarrow < \{eyes^2, head^2\}, \{inside\_of(eyes^2, head^2)\} > \\
&4 : FACE^0 \rightarrow < \{nose^2, head^2\}, \{inside\_of(nose^2, head^2)\} > \\
&5 : FACE^0 \rightarrow < \{mouth^2, head^2\}, \{inside\_of(mouth^2, head^2)\} >
\end{aligned}
$$

The transformation into a MLN is nearly straightforward. First, we need to declare the formulas:

```
aboveEM(eyes,mouth)
aboveNM(nose,mouth)
insideOfEH(eyes,head)
insideOfNH(nose,head)
insideOfMH(mouth,head)
isFaceENMH(eyes,nose,mouth,head)
```

Subsequently, we need to declare the domain:

```
eyes={E1,E2,E3,E4}
nose={N1,N2,N3,N4}
mouth={M1,M2,M3,M4}
head={H1,H2,H3,H4}
```

Finally, we need to write the weighted first-order formulas. We used a validation image dataset and translated the probabilities into weights:

```
1.58 isFaceENMH(e,n,m,h) => aboveEM(e,m)
1.67 isFaceENMH(e,n,m,h) => aboveNM(n,m)
1.16 isFaceENMH(e,n,m,h) => insideOfEH(e,h)
1.25 isFaceENMH(e,n,m,h) => insideOfNH(n,h)
1.34 isFaceENMH(e,n,m,h) => insideOfMH(m,h)
```

To recognize a *face* in an image, the relevant aspects of the image are transformed into a first-order *KB*. For this, the terminal elements are detected (in the example the eyes, mouth, nose and head) in the image, as well as the spatial relations between these elements. Then, the particular image *KB* is combined with the *general* model represented as a MLN; and from this combination a *grounded Markov network* is generated. Object (face) recognition is performed using standard probabilistic inference (see Chap. 6) over the Markov network.

12.7 Additional Reading

There are several introductory books on logic, such as [8, 9]. From an artificial intelligence perspective, [4] provides a good introduction to predicate logic and logic-based representations. Most of the current relational probabilistic models are described in [5], which includes a chapter on each approach. Probabilistic relational models are introduced in [6]; and how to learn PRMs from data is presented in [3]. A review of Markov logic networks is included in [1]. References [7, 10] are introductory books to inductive logic programming.

12.8 Exercises

1. If p and r are false, and q and s are true, determine the truth values of the following expressions:
   - $p \vee q$
   - $\neg p \wedge \neg(q \wedge r)$
   - $p \rightarrow q$
   - $(p \rightarrow q) \wedge (q \rightarrow r)$
   - $(s \rightarrow (p \wedge \neg r)) \wedge ((p \rightarrow (r \vee q)) \wedge s)$
2. Which of the following expressions are true?
   - $\forall x((x^2 - 1) > 0)$
   - $\forall x(x^2 > 0)$
   - $\exists x(1/(x^2 + 1) > 1)$
   - $\neg\exists x((x^2 - 1) \leq 0)$
3. Determine if the next expressions are WFFs:
   - $\forall x \neg(p(x) \rightarrow q(x))$
   - $\exists x \forall x(p(x) \leftrightarrow q(x))$
   - $\exists x \vee q(x) \wedge q(y)$
   - $\forall x \exists y p(x) \vee (p(x) \leftrightarrow q(y))$
4. Write the following sentences in first-order predicate logic: (i) all persons have a father and a mother, (ii) some persons have brothers or sisters or brothers and

sisters, (iii) if a person has a brother, then his father is also the father of his brother, (iv) no person has two natural fathers or two natural mothers.

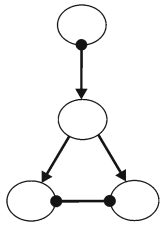
5. Given the PRM of Fig. 12.1, assume that all variables are binary (i.e., TeachingAbility = {Good, Average}, Popularity = {High, Low}, etc.). According to the structure of the model, specify the required conditional probability tables.
6. Based on the PRM of the previous exercise, assume there are two professors, three students, three courses, and five registrations (one student registers for two courses and the other to three courses). Expand the PRM according to the previous objects and generate the corresponding Bayesian network.
7. Consider the MLN example of Sect. 12.4 with two logical formulas and an additional third formula: $\forall x\, UnhealthyDiet(x) \rightarrow Cancer(x)$. Given that the variables x and y are instantiated to *Tim*, *Sue* and *Maria*, obtain the graphical dependency structure of the MLN.
8. Determine the weights for the two logical formulas of the MLN example of Sect. 12.4, assuming that you have extracted the following statistics from a database: (i) 19 persons smoked and have cancer, (ii) 11 persons smoked and do not have cancer, (iii) 10 persons did not smoke and do not have cancer, (iv) 5 persons did not smoke and have cancer, (v) 15 pairs of persons were friends and both smoked, (vi) 5 pairs of persons were friends, one smoked and the other did not.
9. *** Investigate alternative formalisms that combine logic and probability. Analyze their advantages and disadvantages in terms of expressive power and computational efficiency.
10. *** Develop a program for doing inference with PRMs. Given the PRM described at the class level (you can use an object-oriented database), and a set of objects, transform it to a BN. Then perform inference over the BN (use the algorithms developed previously).

References

1. Domingos, P., Richardson, M.: Markov Logic: A Unifying Framework for Statistical Relational Learning. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning, pp. 339–371. MIT Press, Cambridge (2007)
2. Ferrucci, F., Pacini, G., Satta, G., Sessa, M.I., Tortora, G., Tucci, M., Vitiello, G.: Symbol-Relation Grammars: A Formalism for Graphical Languages. Information and Computation **131**(1), 1–46 (1996)
3. Friedman, N., Getoor, L., Koller, D., Pfeffe, A.: Learning Probabilistic Relational Models. In: Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 1300–1309 (1999)
4. Genesereth, M.R., Nilsson, N.J.: Logical Foundations of Artificial Intelligence. Morgan Kaufmann (1988)
5. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
6. Koller D.: Probabilistic Relational Models. In: Proceedings of the 9th International Workshop on Inductive Logic Programming. Lecture Notes in Artificial Intelligence, vol. 1634, Springer, 3–13 (1999)

7. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York (1994)
8. Lemmon, E.J.: Beginning Logic. Hackett Publishing Company (1978)
9. Newton-Smith, W.H.: Logic: An Introductory Course. Routledge, Milton Park (1985)
10. Nienhuys-Cheng, S., de Wolf, R.: Foundations of Inductive Logic Programming. Springer-Verlag, Berlin (1991)
11. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning. **62**(1–2), 107–136 (2006)
12. Ruiz, E., Sucar, L.E.: An object recognition model based on visual grammars and Bayesian networks. In: Proceedings of the Pacific Rim Symposium on Image and Video Technology, LNCS 8333, pp. 349–359, Springer-Verlag (2014)
13. Sucar, L.E., Noguez, J.: Student Modeling. In: O. Pourret, P. Naim, B. Marcot (eds.) Bayesian Belief Networks: A Practical Guide to Applications, pp. 173–186. Wiley and Sons (2008)

Chapter 13
Graphical Causal Models



13.1 Introduction

Causality has to do with cause-effect relations; that is, identifying when there are two (or more) related phenomena, which is the *cause* and which is the *effect*. However, there could be a third explanation, namely, there is another phenomena which is the *common cause* of the original two phenomena of interest.

Probabilistic models do not necessarily represent causal relations. For instance, consider the following two directed Bayesian networks: BN1: $A \rightarrow B \rightarrow C$ and BN2: $A \leftarrow B \leftarrow C$. From a probabilistic perspective, both represent the same set of dependency and independency relations: direct dependencies between A and B, B and C; and $I(A, B, C)$, that is, A and C are independent given B. However, if we define that a directed link $A \rightarrow B$ means A causes B, both models represent very different causal relations.

Given that we can model and solve complex phenomena with probabilistic graphical models, and also that causality is for many a complex and controversial concept, we may ask ourselves, Why do we need causal models? There are several advantages to causal models, in particular, graphical causal models.

First, if we build a Bayesian network based on causal relations, the models tend to be easier to construct and understand (e.g., for communicating with domain experts), and are usually simpler. For example, in the medical domain it is common to have a disease that produces several symptoms, which usually are conditionally independent given the disease. If we represent this as a BN in the direction of causality, we obtain the structure of Fig. 13.1a. An alternative structure is shown in Fig. 13.1b, in which we require links between the symptom variables, as these are conditionally independent given the disease but not marginally independent. The model built based on causal information is simpler.

Secondly, with causal models we can perform other types of reasoning that are not possible, at least in a direct way, with PGMs such as Bayesian networks. These other inference situations are: (i) *interventions*, in which we want to find the effects

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3_13

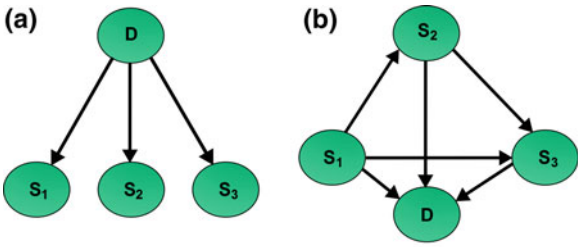


Fig. 13.1 A Bayesian network that represents a disease (D) and three symptoms (S_1, S_2, S_3) which are conditionally independent given D. **a** Structure based on *causal* relations. **b** Alternative structure

of setting a certain variable to a specific value by an external agent (note that this is different from observing a variable); and (ii) *counterfactuals*, where we want to reason about what would have happened if certain information had been different from what actually happened. We will discuss these two situations in more detail in the rest of the chapter.

Several causal modeling techniques have been developed; for example, functional equations, path diagrams, and structural equation models, among others. In this chapter we will focus on graphical models, in particular causal Bayesian networks.

13.2 Causal Bayesian Networks

A Causal Bayesian network (CBN) is a directed acyclic graph, G, in which each node represents a variable and the arcs in the graph represent *causal* relations; that is, the relation $A \rightarrow B$ represents some physical mechanism such that the value of B is directly affected by the value of A. This relation can be interpreted in terms of *interventions*—setting of the value of some variable or variables by an external agent. For example, assume that A stands for a water sprinkler (OFF/ON) and B represents if the grass is wet (FALSE/TRUE). If the grass (B) is originally not WET and the sprinkler is set to ON by an intervention, then B will change to TRUE.

As in BNs, if there is an arc from A to B (A is a direct cause of B), then A is a parent of B, and B is a child of A. Given any variable X in a CBN, $Pa(X)$ is the set of all parents of X. Also, similarly to BNs, when the direct or immediate causes—parents—of a variable are known, the more remote causes (or ancestors) are irrelevant. For example, once we know that the grass is WET, this makes it SLIPPERY, no matter how the grass became wet (we turned on the sprinkler or it rained).

Causal networks represent stronger assumptions than Bayesian networks, as all the relations implied by the network structure should correspond to causal relations. Thus, all the parent nodes, $Pa(X)$, of a certain variable, X, correspond to the direct causes of X. This means that if any of the parent variables of X, or any combination

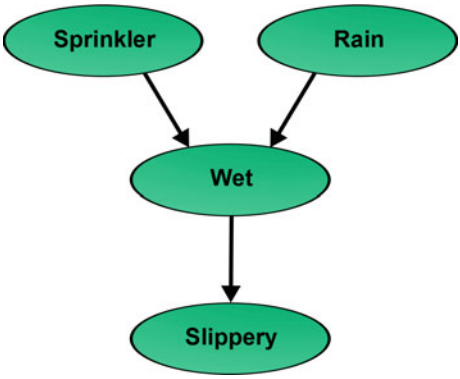


Fig. 13.2 A simple example of a CBN which represents the relations: *Sprinkler* causes *Wet*, *Rain* causes *Wet*, and *Wet* causes *Slippery*. (Example taken from J. Pearl [3].)

of them, is set to a certain value via an intervention, this will have an effect on X. In a CBN, a variable which is a root node (variable with no parents) is called *exogenous*, and all other variables are *endogenous*.

A simple example of a CBN is depicted in Fig. 13.2, which basically encodes the following causal relations: (i) *Sprinkler* causes *Wet*, (ii) *Rain* causes *Wet*, (iii) *Wet* causes *Slippery*. In this case, *Sprinkler* and *Rain* are exogenous variables, and *Wet* and *Slippery* are endogenous variables.

If a $P(\mathbf{X})$ is the joint probability distribution of the set of variables $\mathbf{X}$, then we define $P_y(\mathbf{X})$ as the distribution resulting from setting the value for a subset of variables, $\mathbf{Y}$, via an intervention. This can be represented as $do(\mathbf{Y} = \mathbf{y})$, where $\mathbf{y}$ is a set of constants. For instance, given the CBN of Fig. 13.2, if we set the sprinkler to ON—$do(Sprinkler{=}ON)$, then the resulting distribution will be denoted as $P_{Sprinkler=ON}(Sprinkler, Rain, Wet, Slippery)$.

Formally, a Causal Bayesian Network can be defined as follows [3]:

A CBN G is a directed acyclic graph over a set of variables $\mathbf{X}$ that is compatible with all the distributions resulting from interventions on $\mathbf{Y} \subseteq \mathbf{X}$, in which the following conditions are satisfied:

1. The probability distribution $P_y(\mathbf{X})$ resulting from an intervention is Markov compatible with the graph G; that is, it is equivalent to the product of the conditional probability of each variable $X \in G$ given its parents: $P_y(\mathbf{X}) = \prod_{X_i} P(X_i \mid Pa(X_i))$.
2. The probability of all the variables that are part of an intervention is equal to one for the value it is set to: $P_y(X_i) = 1$ if $X_i = x_i$ is consistent with $Y = y$, $\forall X_i \in \mathbf{Y}$.
3. The probability of each of the remaining variables that are not part of the intervention is equal to the probability of the variable given its parents and it is consistent with the intervention: $P_y(X_i \mid Pa(X_i)) = P(X_i \mid Pa(X_i)), \forall X_i \notin \mathbf{Y}$.

Given the previous definition, and in particular the fact that the probability of the variables that are set in the intervention is equal to one (condition 2), the joint probability distribution can be calculated as a *truncated* factorization:

$$P_y(\mathbf{X}) = \prod_{X_i \notin \mathbf{Y}} P(X_i \mid Pa(X_i)) \tag{13.1}$$

such that all X_i are consistent with the intervention $\mathbf{Y}$.

Another consequence of the previous definition, is that once all the parents of a variable X_i are set by an intervention, setting any other variable, $\mathbf{W}$, does not affect the probability of X_i:

$$P_{Pa(X_i),\mathbf{W}}(X_i) = P_{Pa(X_i)}(X_i) \tag{13.2}$$

such that $\mathbf{W} \cap (X_i, Pa(X_i)) = \emptyset$.

Considering again the example in Fig. 13.2, if we make the grass wet by any mean, $do(Wet = TRUE)$, then the probability of *Slippery* is not affected by *Rain* or *Sprinkler*.

13.3 Causal Reasoning

Causal reasoning has to do with answering causal queries from a causal model, and in our case in particular from graphical causal models. There are several types of causal queries we might consider, we will start by analyzing causal predictions, and then we will analyze counterfactuals.

13.3.1 Prediction

Consider a causal Bayesian network which includes a set of variables: $X_G = \{X_C, X_E, X_O\}$; where X_C is a subset of *causes* and X_E is a subset of *effects*; X_O are the remaining variables. We want to perform a causal query on the model: What will be the effect on X_E when setting $X_C = x_C$? That is, we want to obtain the probability distribution of X_E that results from the intervention $X_C = x_C$:

$$P_C(X_E \mid do(X_C = x_C)) \tag{13.3}$$

To perform causal prediction with a CBN, G, the following procedure is followed:

1. Eliminate all incoming arrows in the graph to all nodes in X_C, thus obtaining a modified CBN, G_r.
2. Fix the values of all variables in X_C, $X_C = x_C$.

3. Calculate the resulting distribution in the modified model G_r (via probability propagation as in Bayesian networks).

For example, consider the hypothetical CBN depicted in Fig. 13.3a that represents certain causal knowledge about a stroke. If we want to measure the effect of an "unhealthy diet" in the variable "stroke," then, according to the previous procedure, we eliminate the link from "health consciousness" to "diet," resulting in the model in Fig. 13.3b. Then we should set the value of "unhealthy diet" to *TRUE*, and by probability propagation obtain the distribution of "stroke." If the distribution of "stroke" changes depending on the value of "unhealthy diet," we can conclude according to this model that it does have an effect.

An interesting question is: When is the distribution resulting from an intervention equal to the distribution resulting from an observation? In mathematical terms, is $P(X_E \mid X_C) = P_C(X_E \mid do(X_C = x_C))$? Both are equal if X_C includes all the parents of X_E and none of its descendants; given that any variable in a BN is independent of its non-descendants given its parents. In other cases they are not necessarily equal, they will depend on other conditions.

13.3.2 Counterfactuals

Counterfactuals are a way of reasoning that we commonly perform in our life. For instance, consider the causal model of Fig. 13.3a. A typical counterfactual question

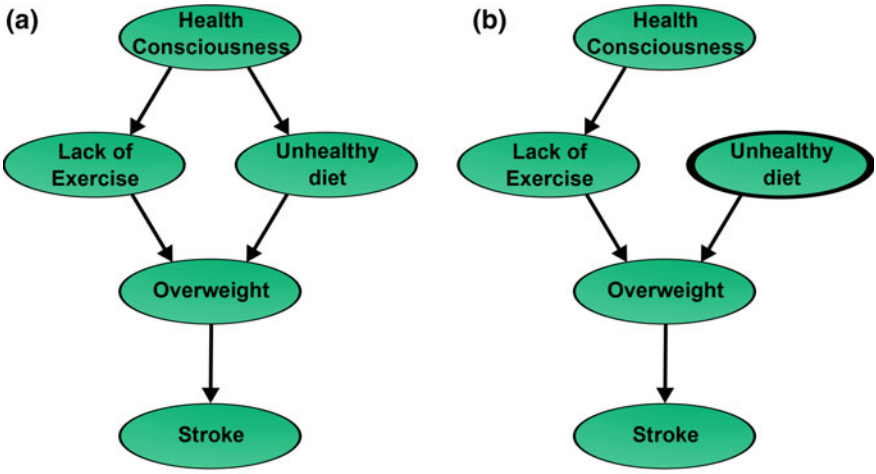


Fig. 13.3 An example of causal prediction. **a** A simple, hypothetical CBN that represents some causal relations related to a stroke. **b** The resulting graphical model obtained from (**a**) by the intervention $do(unhealthy\text{-}diet = TRUE)$

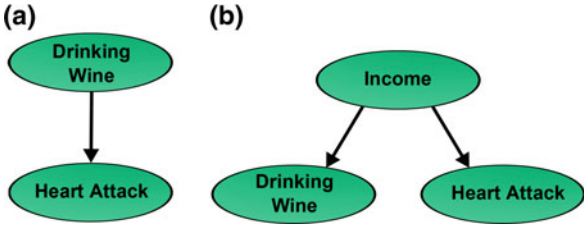


Fig. 13.4 An example of the difficulty of learning causal models. **a** The initial network that shows an apparent causal relation between "drinking wine" and "heart attack." **b** An alternative network with a common cause, "income," that explains the dependency between "drinking wine" and "heart attack"

would be: If some person suffered a stroke, would he still have suffered the stroke ("stroke" = TRUE) if he had exercised more ("lack of exercise" = FALSE)?

Counterfactual inference involves three main steps:

1. Abduction: modify the model in terms of the new evidence (in the example, modify the value of "stroke" to unknown).
2. Action: perform the minimal intervention in the model according to the hypothetical evidence (in the example, set "lack of exercise" to FALSE, removing the link from "health consciousness" to "lack of exercise").
3. Prediction: perform probabilistic inference on the modified model and obtain the probability of the variable(s) of interest (in the example, perform probability propagation to estimate $P_{lack\text{-}exercise}(stroke \mid do(lack\text{-}exercise = FALSE))$).

13.4 Learning Causal Models

Learning causal models from data without direct interventions poses many challenges. As we mentioned before, if we discover that there is a certain dependency between two variables, X and Y, we cannot determine, without additional information, if X *causes* Y or vice versa. Additionally, there could be some other factor that produces the dependency between these two variables.

For instance, consider that based on data we discover that people who drink wine tend to have less heart attacks. Then we might be inclined to conclude that drinking wine tends to decrease the probability of a heart attack (Fig. 13.4a). However, there might be another variable that produces this apparent causal relation, known as a *latent common cause*. It could be that both, wine drinking and heart attacks, have to do with the income level of the person, as persons with high income tend to drink more wine and at the same time have a lower probability of heart attack because of better medical services (see Fig. 13.4b). Thus, a difficulty of learning causal relations lies in how to include in the model all the relevant factors.

Several algorithms have been developed for learning causal networks. In general, the following assumptions are used when learning the structure of causal networks:

Causal Markov Condition: a variable is independent of its non-descendants given its direct causes (parents in the graph).

Faithfulness: there are no additional independencies between the variables in the model that are not implied by the causal Markov condition.

Causal Sufficiency: there are no common confounders of the observed variables in the model.

An algorithm for learning causal networks is the Bayesian Constraint-Based Causal Discovery (BCCD) [2]. This technique is an extension of the PC Bayesian network structure learning algorithm (see Chap. 8) that consists of two main phases:

1. Start with a completely connected graph and estimate the reliability of each causal link, $X - Y$, by measuring the conditional independence between X and Y. If a pair of variables are conditionally independent with a reliability above a certain threshold, then delete the edge between these variables.
2. The remaining causal relations (undirected edges in the graph) are ordered according to their reliability. Then the edges in the graph are oriented starting from the most reliable relations, based on the conditional independence test for variable triples.

To estimate the reliability of a causal relation, $R = X \rightarrow Y$, the algorithm uses a Bayesian score:

$$P(R \mid D) = \frac{P(D \mid M_R)P(M_R)}{P(D \mid M)p(M)} \tag{13.4}$$

where D is the data, M are all the possible structures, and M_R are all the structures that contain the relation R. Thus, $P(M)$ denote the prior probability of a structure M and $P(D \mid M)$ the probability of the data given the structure. Calculating (13.4) is very costly, so it is approximated by the marginal likelihood of the data given the structure, and usually restricted to a maximum number of variables in the network.

Depending on the reliability threshold, the resulting network can have undirected edges, $-$, which means that there is not enough information to obtain the direction of the link, and bidirected edges, $\leftrightarrow$, indicating that there is a common cofounder. This type of structure is called a *Maximal Ancestral Graph* (MAG). The equivalence class for MAGs is a *Partial Ancestral Graph* (PAG). In a PAG there are three types of edges: directed, $\rightarrow$, and undirected, $-$, when these are consistent for all the graphs in the equivalence class; and those which are not consistent are marked with a circle "o".

In the next section, we will illustrate the resulting PAG when learning a casual model for a real-world application.

13.5 Applications

There are many practical applications in which a causal model is useful. Just to mention a few:

- Predicting the effects of certain interventions.
- Learning causal graphical models from data.
- Diagnosis of physical systems.
- Generating explanations.
- Understanding causal expressions.

Next, we describe an application for learning causal graphical models from data.

13.5.1 Learning a Causal Model for ADHD

In [4], the authors extended the BCCD algorithm for a mixture of discrete and continuous variables, and apply it to a dataset that contains phenotypic information about children with Attention Deficit Hyperactivity Disorder (ADHD). They used a reduced dataset that contains data from 223 subjects, with the following nine variables per subject:

1. Gender (male/female)
2. Attention deficit score (continuous)
3. Hyperactivity/impulsivity score (continuous)
4. Verbal IQ (continuous)
5. Performance IQ (continuous)
6. Full IQ (continuous)
7. Aggressive behavior (yes/no)
8. Medication status (naïve/not naïve)
9. Handedness (right/left)

Given the small dataset they included some background information, in particular that no variable in the network can cause "gender." Using a reliability threshold of 50 %, the network depicted in Fig. 13.5 was obtained; which is represented as a parental ancestral graph.

Several interesting causal relations are suggested by the resulting network, some of which were already known from previous medical studies [4]:

- There is a strong effect from gender on the level of attention deficit.
- The level of attention deficit affects hyperactivity/impulsivity and aggressiveness.
- Handedness (left) is associated with aggressive behavior.
- The association between performance IQ, verbal IQ, and full IQ is explained by a latent common cause.
- Only the performance IQ has a direct causal link with attention deficit.

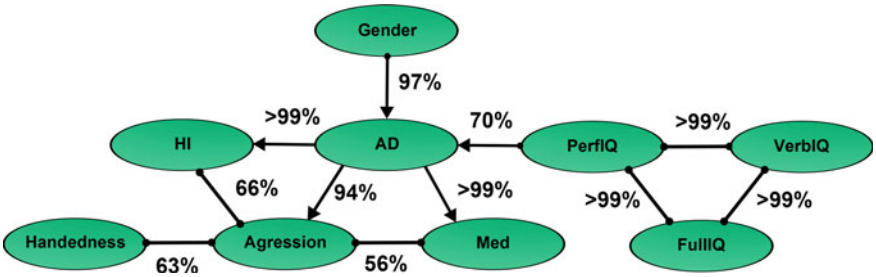


Fig. 13.5 The causal model obtained from the ADHD dataset. The graph represents the resulting PAG in which edges are marked as $\rightarrow$ or $-$ for invariant relations and as "o" for non-invariant relations. The reliability of each edge is indicated. (*AD* attention deficit score, *HI* hyperactivity/impulsivity score, *PerfIQ* performance IQ, *VerbIQ* verbal IQ, *Med* medication status.) Figure based on [4]

13.6 Additional Reading

Graphical causal modeling was originally introduced by [6] in genetics. Two comprehensive books on graphical causal models are [3, 5]. The BCCD algorithm for learning causal graphs is introduced in [2]. A review of alternative approaches for learning causal networks can be found in [1].

13.7 Exercises

1. What are the differences between a directed graphical model, such as a Bayesian network, and a causal model?
2. Consider the CBN in Fig. 13.2. Obtain some other alternative Bayesian network models for these four variables which are not necessarily *causal*. How do these networks compare in terms of simplicity and clarity to the original model?
3. Given the CBN in Fig. 13.3a: (a) assuming all variables are binary, define the conditional probability tables based on your intuition; (b) obtain the prior probability for all the variables via probabilistic inference; (c) given the intervention $do(\textit{Unhealthy-diet} = \textit{true})$, calculate the posterior probability of *Overweight* and *Stroke*.
4. According to the results of the previous exercise, does an unhealthy-diet have an effect on *Stroke*? Why?
5. Compute the posterior probability of *Overweight* and *Stroke* given the observation $\textit{Unhealthy-diet} = \textit{true}$ for the network in Fig. 13.3a (use the same parameters as for Exercise 3). Do the obtained probabilities coincide with the ones from the corresponding intervention? Why?

6. Given the CBN in Fig. 13.3a, consider the counterfactual question: Will the probability for a stroke decrease if the person has a healthy diet (*Unhealthy-diet* = *false*)? Perform the required operations on the model (using the same parameters as in Exercise 3) to answer this question.
7. Given the data on the golf example (in Chap. 8), apply the standard PC algorithm to learn the structure of a BN. Then apply the BCCD algorithm, using an estimate of the reliability of each causal link to order the causal relation, and setting a threshold to determine if a link has certain direction or remains undirected. Compare the structures obtained with PC and BCCD.
8. Given the PAG in Fig. 13.5, show all the MAGs included in this PAG.
9. *** Develop a program that implements the BCCD algorithm for discrete variables.
10. *** Obtain data from some real-world domain and use the program from the previous exercise to obtain a causal model. Vary the reliability threshold and compare the models obtained.

References

1. Aitken, J.S.: Learning Bayesian networks: approaches and issues. Knowl. Eng. Rev. **26**(2), 99–157 (2011)
2. Claassen, T., Heskes, T: A Bayesian approach to constraint based causal inference. In: Proceedings of Uncertainty in Artificial Intelligence (UAI), AUAI Press, pp. 207–216 (2012)
3. Pearl, J.: Causality: Models. Reasoning and Inference. Cambridge University Press, New York (2009)
4. Sokolova, E., Groot, P., Classen, T., Heskes, T.: Causal discovery from databases with discrete and continuous variables. In: Probabilistic graphical models (PGM). Springer, pp. 442–457 (2014)
5. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. MIT Press, Cambridge (2000)
6. Wright, S.: Correlation and causation. J. Agric. Res. **20**, 557–585 (1921)

Glossary

Bayesian Classifier A classifier that assigns probabilities to the different object labels based on Bayes rule.

Bayesian network A directed cycling graph that represents the joint distribution of a set of random variables such that each variable is conditionally independent of its non-descendants given its parents in the graph.

Causal Bayesian Network A directed acyclic graph in which the nodes represent random variables and the arcs causal relations.

Causal reasoning A procedure for answering causal queries form a causal model.

Classifier A method or algorithm that assigns labels to objects.

Clique A completely connected subset of nodes in a graph that is maximal.

Conditional independence Two variables are conditionally independent given a third variable if they become independent when the third variable is known.

Conditional Probability Probability of certain event given that another event has occurred.

Conditional Random Field A random field in which all the variables are globally conditioned on the observations.

Decision Tree A tree that represents a decision problem and has three types of nodes: decisions, uncertain events and results.

Directed Acyclic Graph A directed graph that has no directed circuits (a directed circuit is a circuit in which all edges in the sequence follow the directions of the arrows).

D-separation A graphical criteria for determining if two subsets of variables are conditionally independent given a third subset in a Bayesian network.

Dynamic Bayesian Network An extension of Bayesian networks to model dynamic processes; it consists of series of time slices, each time slice represents the state of all variables at certain time.

Expectation-Maximization An statistical technique used for parameter estimation when there are non-observable variables.

Graph A graphical representation of binary relations between a set of objects.

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3

Hidden Markov Model A Markov chain in which the states are not directly observable.

Independent variables Two random variables are independent if knowing the value of one of them does not affect the probability distribution of the other one.

Influence Diagram A graphical model for solving decision problems. It is an extension of Bayesian networks that incorporates decision and utility nodes.

Junction Tree A tree in which each node corresponds to a subset of variables of a probabilistic graphical model.

Limited Memory Influence Diagram An influence diagram in which the variables known when making a decision are not necessarily remembered for future decisions.

Markov Blanket A set of variables that make a variable independent of all other variables in a probabilistic graphical model.

Markov Chain A state machine in which the transition between states are nondeterministic and satisfy the Markov property.

Markov Decision Process A graphical model for sequential decision making composed of a finite set of states and actions, in which the states follow the Markov property.

Markov Network A random field represent as an undirected graph that satisfies the locality property—each variable in the field is independent of all other variables given its neighbors in the graph.

Markov Property The probability of the next (future) state is independent of the previous states (past) given the current (present) state.

Markov Random Field Markov network.

Multidimensional classifier A classifier that can assign more than one label to each object.

Naive Bayes Classifier A Bayesian classifier that assumes that all attributes are independent given the class variable.

Partially Observable Markov Decision Process A Markov decision process in which the states are not directly observable.

Policy A function that maps states to actions.

Probabilistic Graphical Model A compact representation of a joint probability distribution of a set of random variables composed by a graph and a set of local probability distributions.

Probabilistic Inference A procedure for calculating the posterior probability of the unknown variables in a probabilistic graphical model given certain evidence (a subset of known or instantiated variables).

Probability A function that assigns a real number to each event (subset of a sample space) and satisfies certain axioms known as the probability axioms.

Random Field A collection of random variables indexed by sites.

Random Variable A mapping form a sample space to real numbers.

Rational Agent An agent that selects its decisions to maximize its expected utility according to its preferences.

Relational Probabilistic Graphical Models An extension of probabilistic graphical models that are more expressive by incorporating some type of relational representation.

Sample space The set of possible outcomes of an experiment.

Temporal Event Network A Bayesian network for modeling dynamic processes in which each node represents the time of occurrence of an event or state change of certain variable.

Tree A connected graph that does not have simple circuits.

Index

L.E. Sucar, *Probabilistic Graphical Models*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-6699-3