9

Knowledge Engineering with Bayesian Networks

9.1 Introduction

Within the Bayesian network research community, the initial work in the 1980's and early 1990's focused on inference algorithms to make the technology computationally feasible. As it became clear that the "knowledge bottleneck" of the early expert systems was back — meaning the difficulties of finding human domain experts, extracting their knowledge and putting it into production systems — the research emphasis shifted to automated learning methods. That is necessary and inevitable. But what practitioners require then is a overarching methodology which combines these diverse techniques into a single "knowledge engineering" process, allowing for the construction Bayesian models under a variety of circumstances, which we call **Knowledge Engineering with Bayesian Networks**, or **KEBN**.

In this chapter we tie together the various techniques and algorithms we have previously introduced for building BNs and supplement them with additional methods, largely drawn from the software engineering discipline, in order to propose a general methodology for the development and deployment of BNs. We supplement this in the next chapter with methods for evaluating Bayesian networks, giving an outline of a comprehensive methodology for modeling with Bayesian networks. No one has fully tested such a methodology, so our KEBN model must remain somewhat speculative. In any case, we can illustrate some of its major features with a number of case studies from our experience, which we proceed to do in Chapter 11.

9.1.1 Bayesian network modeling tasks

When constructing a Bayesian network, the major modeling issues that arise are:

1. What are the variables? What are their values/states?
2. What is the graph structure?
3. What are the parameters (probabilities)?

When building decision nets, the additional questions are:

4. What are the available actions/decisions, and what impact do they have?
5. What are the utility nodes and their dependencies?
6. What are the preferences (utilities)?

Expert elicitation is a major method for all of these tasks. Methods involving automated learning from data, and adapting from data, can be used for tasks 1-3 (if suitable data are available). We have described the main techniques for tasks 2 and 3 in Chapters 6, 7 and 8. Task 1 has been automated as well, although we do not go into these methods in this text. Identifying variables is known in the machine learning literature as unsupervised classification; see, for example, Chris Wallace's work on Snob [290, 291]. There are many techniques for automated discretization, for example [193]. On the other hand, very little has been done to automate methods for tasks associated with building decision networks and we do not cover them in this text. In the remainder of this chapter we shall first focus on how to perform all the tasks using expert elicitation, then we shall consider methods for **adaptation**, combining elicitation with machine learning, in §9.4.

9.2 The KEBN process

9.2.1 KEBN lifecycle model

A simple view of the software engineering process construes it as having a lifecycle: the software is born (design), matures (coding), has a lengthy middle age (maintenance) and dies of old age (obsolescence). Our best effort at construing KEBN in such a **lifecycle model** (also called a "waterfall" model) is shown in Figure 9.1. Although we prefer a different view of KEBN (presented just below), the lifecycle is a convenient way of introducing many aspects of the problem, partly because it is so widely known and understood.

Building the Bayesian network is where the vast majority of research effort in KEBN has gone to date. In the construction phase, the major network components of structure, parameters and, if a decision network, utilities (preferences) must be determined through elicitation from experts, or learned with data mining methods, or some combination of the two.

Evaluation aims to establish that the network is right for the job, answering such questions as: Is the predictive accuracy for a query node satisfactory? Does it respect any known temporal order of the variables? Does it incorporate known causal structure? **Sensitivity analysis** looks at how sensitive the network is to changes in input and parameter values, which can be useful both for validating that the network is correct and for understanding how best to use the network in the field.

Field testing first puts the BN into actual use, allowing its usability and performance to be gauged. **Alpha testing** refers to an intermediate test of the system by inhouse people who were not directly involved in developing it; for example, by other inhouse BN experts. **Beta testing** is testing in an actual application by a "friendly" end-user, who is prepared to accept hitting bugs in early release software. For software that is not being widely marketed, such as most BNs, this idea may be inapplicable — although domain experts may take on this role. Acceptance testing is surely required:

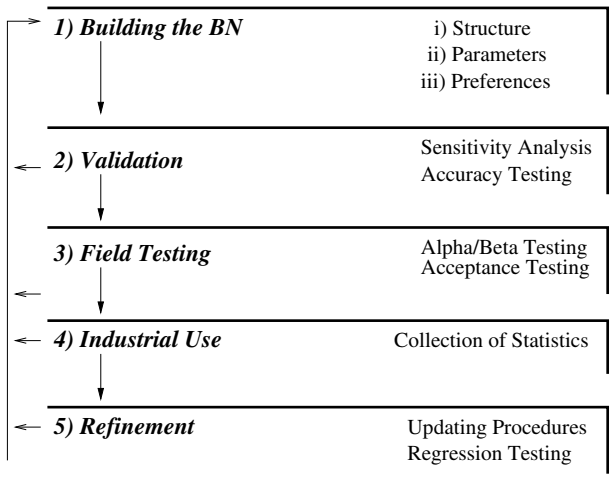

FIGURE 9.1
A KEBN lifecycle model.

it means getting the end users to accept that the BN software meets their criteria for use.
Industrial use sees the BN in regular use in the field and requires that procedures be put in place for this continued use. This may require the establishment of a new regime for collecting statistics on the performance of the BN and statistics monitoring the application domain, in order to further validate and refine the network.
Refinement requires some kind of change management regime to deal with requests for enhancement or fixing bugs. **Regression testing** verifies that any changes do not cause a degradation (regression) in prior performance.

In this chapter we will describe detailed procedures for implementing many of these steps for Bayesian network modeling. Those which we do not address specifically, such as how to do regression testing and acceptance testing, do not seem to have features specific to Bayesian network modeling which are not already addressed here. We refer you to other works on software engineering which treat those matters in the notes at the end of the chapter (§9.6).

9.2.2 Prototyping and spiral KEBN

We prefer the idea of prototyping for the KEBN process to the lifecycle model. Prototyping interprets the analogy of life somewhat differently: as an "organism," the software should grow by stages from childhood to adulthood, but at any given stage it is a self-sufficient, if limited, organism. **Prototypes** are functional implementations of software: they accept real input, such as the final system can be expected to deal with, and produce output of the type end-users will expect to find in the final system. What distinguishes early form prototypes from the final software is that the func-

tions they implement are limited, so that they are fairly easily developed, whereas later ones approximate the full functionality envisioned. Since each prototype in the entire sequence supports a limited form of the targeted functionality, end-users can experiment with them in just the way they are intended to use the final product, and so they can provide feedback and advice from the early stages of development. Much of the testing, then, is done in a setting as close to the target usage environment as possible.

The **initial prototypes** should be used for planning the KEBN process. The subproblem addressed should be self-contained but reasonably representative of the global problem. It should be scoped to minimize development risk, with the prototype employing available capabilities as much as possible and using simplified variables and structure. As a result you avoid one of the main risks in the BN development process, overselling the capabilities of the final system [166]. Since initial prototypes are both functional and representative, they provide a working product for provisional assessment and planning.

The **incremental prototypes** require relatively simple extensions to the preceding prototype. They should attack a high priority, but small, subset of the remaining difficult issues. The size of the subset of issues tackled, and their difficulty, is used to control the continuing development risk. The incremental development refines both the domain expert's and the knowledge engineer's understanding of the requirements and approach.

Why prototype? We believe that it just is the best software development process overall, agreeing with Fred Brooks [33] and Barry Boehm [22]. Prototyping allows the organic growth of software, tracking the problem specifications and growing in manageable spurts. The use of prototypes attacks the trade-off between comprehensiveness and intelligibility from the right starting point, namely the small end. Those are general attributes favoring prototyping for software engineering of all kinds. However, because Bayesian network development usually involves building graphical models using visual aids from early on, it is even easier than normal to provide end-users with the kind of graphical user interface the end product is likely to have, easing the prototyping approach.

In order to highlight the differences between prototyping and the usual approach to software development, as typified in lifecycle models, Barry Boehm introduced the **Spiral Model** of software development, which we illustrate in Figure 9.2. The construction of a sequence of prototypes can be viewed as a repeating cycle of analyzing requirements, design, implementation, operation and evaluation. In evaluation the behavior of each prototype model on sample problems is explored, in conjunction with end-users, and the next stage is planned as problems in the current prototype are uncovered.

9.2.3 Are BNs suitable for the domain problem?

As we have already seen, Bayesian networks can handle a wide variety of domains and types of application. They explicitly model uncertainty, allow for the representation of complex interactions between variables, and they can be extended with

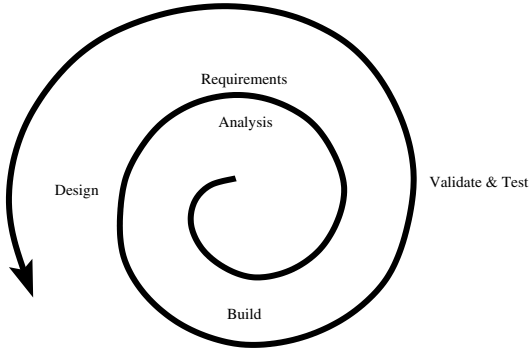

FIGURE 9.2
A spiral model for KEBN.

utilities and decision nodes for planning and decision making. Furthermore, the representations developed are not just black boxes, but have a clear semantics, available for inspection. Nevertheless, BNs are not suitable to any and every application, so before launching any large-scale KEBN process, it is important to make sure that BN technology is suitable for the particular problem. It is easiest to list features that would suggest BNs are *not* appropriate.

- If the problem is a "one-off," for which there is no data available and any model built won't be used again, then the overhead of the KE process may not be worth it. Bayesian networks might still be used in a one-off modeling process, of course, without going through all of the KEBN knowledge engineering overhead.
- There are no domain experts, nor useful data.
- If the problem is very complex or not obviously decomposable, it may not be worth attempting to analyze into a Bayesian network.
- If the problem is essentially one of learning a function from available data, and a "black box" model is all that is required, an artificial neural network or other standard machine learning technique may be applied.

9.2.4 Process management

It is important that the knowledge engineering process be properly managed. Foremost is the management of human relations. It is unrealistic to expect that putting the domain expert and the knowledge engineer in a room together will result in the smooth production of a BN model for the domain. It is far more likely that they will talk past each other! The knowledge engineer must learn about the problem domain and the domain expert must learn what BN models are and what they can do. This learning aspect of the process can be time-consuming and frustrating, but both sides must expect and tolerate it.

One productive way of training the participants is to start by constructing very simple models of simplified, throw-away domain problems. These are "pre-prototypes:" whereas prototypes are intended to feed into the next stage, at least conceptually, the throw-away models are intended to build mutual understanding. As knowledge engineering proceeds, both the domain expert and the knowledge engineer's understanding of the problem domain deepens. Throughout this process building communication between them is vital!

It is particularly important to get commitment from the expert to the project and the knowledge engineering process. It is going to take much time and effort from the expert, so the expert has to be convinced early on that it will be worth it. Certain characteristics are desirable in an expert: their expertise is acknowledged by their peers, they are articulate and they have the interest and the ability to *reason about* the reasoning process. Support, or at least tolerance, from the expert's management is, of course, necessary.

The rationale for modeling decisions should be recorded in a "style guide," which can be used to ensure that there is consistency across different parts of the model, even if they are developed at different times and by different people. Such a style guide should include naming conventions, definitions and any other modeling conventions. Most important is documenting the history of significant design decisions, so that subsequent decision making can be informed by that process. This is an aspect of **change management**. Another aspect is archiving the sequence of models developed. An automated tool, such as Unix's CVS, can be used to generate such archives, as well as avoid colliding changes when many people are working on the same project.

9.3 Modeling and elicitation

In this section we introduce methods for eliciting Bayesian network structure, parameters and preferences (utilities).

9.3.1 Variables and values

9.3.1.1 Types of node

When attempting to model large, complex domains it is important to limit the number of variables in the model, at least in the beginning, in order to keep the KE task tractable. The key is to determine which are the most important variables/nodes.

- One class of variables consists of those whose values the end-user wants to know about, the "output" nodes of the network, and are often referred to as the **target** or **query** nodes.
- The **evidence** or **observation** nodes play the role of "inputs" and can be identified by considering what sources of information about the domain are available, in particular, what evidence could be observed that would be useful in inferring the state of another variable.

- **Context** variables can be determined by considering sensing conditions and background causal conditions.
- **Controllable** variables are those whose values can be set by intervention in the domain environment (as opposed to simply observing their value).

It is important to note that the roles of nodes may change, depending how the BN is to be used. It is often useful to work backwards by identifying the query variables and "spreading out" to the related variables.

Let us return to the cancer diagnosis example used throughout Chapter 2 and look at it in terms of variable identification. The main interest of medical diagnosis is identifying the disease from which the patient is suffering; in this example, there are three candidate diagnoses. An initial modeling choice might be to have the query node *Disease*, with the observation nodes being *Dyspnoea* (shortness of breath), which is a possible symptom, and *X-ray*, which will provide another source of information. The context variables in this case are the background information about the patient, such as whether or not he is a *Smoker*, and what sort of exposure to *Pollution* he has had. In this simple diagnosis example, nothing has been described thus far that plainly falls into the category of a controllable variable. However, the doctor may well prefer to treat *Smoker* as a controllable variable, instead of as context, by attempting to get the patient to quit smoking. That may well turn into an example of a not-fully-effective intervention, as many doctors have discovered!

9.3.1.2 Types of values

When considering the variables, we must also decide what states, or values, the variable can take. Some common types of discrete nodes were introduced in §2.2.1: Boolean nodes, integer valued or multinomial categories. For its simplicity, and because people tend to think in terms of propositions (which are true or false), Boolean variables are very commonly employed. Equivalently, two-valued (binary) variables may be used, depending upon what seems most natural to the users. For example, when modeling the weather, the main weather node could be called *Weather*, and take the values {*fine, wet*}, or the node could be made a Boolean called *FineWeather* and take the values {*T, F*}. Other discrete node types will likely be chosen when potential observations are more fine-grained.

9.3.1.3 Common modeling errors

Discrete variable values must be **exhaustive** and **exclusive**, which means that the variable must take on exactly one of these values at a time. Modeling mistakes relating to each of these factors are common. For example, suppose that a preliminary choice for the *Disease* query node is to give it the values {*lungCancer, bronchitis, tuberculosis*}. This modeling choice isn't exhaustive, as it doesn't allow for the possibility of another disease being the cause of the symptoms; adding a fourth alternative $other$ alleviates the problem. However, this doesn't solve the second problem, since taking these as exclusive would imply that the patient can only suffer from one of these diseases. In reality it is possible (though of course uncommon) for a patient

to suffer from more than one of these, for example, *both* lung cancer and bronchitis. The best modeling solution here is to have distinct Boolean variables for each disease of interest, say the nodes *LungCancer*, *Bronchitis* and *Tuberculosis*. This model does not explicitly represent the situation where the patient suffers from another disease, but doesn't exclude it either.

Another common problem made by naive BN modelers is the creation of separate variables for different states of the same variable. For example, they might create both a *FineWeather* variable and a *WetWeather* variable (both Boolean). These states ought to be mutually exclusive, but once they are created as separate variables, the error is often "solved" by the addition of an additional arc between the nodes and a deterministic CPT that enforces the mutual exclusion. This is not very satisfactory, however, as the resultant structure is more complex than necessary. This is also an example of how the choices made when modeling nodes and states affects structure modeling.

9.3.1.4 Discretization

While it is possible to build BNs with continuous variables without discretization, the simplest approach is to **discretize** them, meaning that they are converted into multinomial variables where each value identifies a different subrange of the original range of continuous values. Indeed, many of the current BN software tools available (including Netica) require this. Netica provides a choice between its doing the discretization for you crudely, into even-sized chunks, or allowing the knowledge engineer more control over the process. We recommend exercising this control and discretize manually or else using a more sophisticated algorithm, such as [193].

TABLE 9.1
Alternative discretizations of an *Age* node with 4 values

Whole Population	Pension Population	Students
0-18	50-60	4-12
19-40	61-67	13-17
41-60	68-72	18-22
61-110	73-110	23-90

Consider the situation where you must discretize an *Age* node. One possible discretization might reflect the age distribution in the population. However, the optimal discretization may vary depending on the situation, from the whole population, to people who receive government pensions, or people engaged in full time study — see Table 9.1. Here, each range covers 25% of the target population.

This discretization is still based on the idea of even-sized chunks. It may be better to base the discretization on differences in effect on related variables. Table 9.2 shows a possible discretization when modeling the connection between *Age* and number of children, represented by the node *NumChildren*.

TABLE 9.2
Discretization of an *Age* node based on differences in number of children

Age	P(NumChildren|Age)				
	0	1	2	3	>4
0-11	1	0	0	0	0
12-16	0.95	0.04	0.01	0	0
17-21	0.90	0.07	0.02	0.01	0
22-25	0.80	0.12	0.05	0.02	0.01
26-30	0.40	0.25	0.18	0.10	0.07
31-34	0.30	0.25	0.25	0.14	0.06
35-42	0.25	0.20	0.30	0.20	0.05
43-110	0.22	0.23	0.25	0.22	0.08

9.3.2 Graphical structure

There are several competing goals when building the graphical structure of a network. First, we would like to minimize the number of parameters, both in order to make the probability elicitation task easier and to simplify belief updating. These goals suggest fewer nodes, fewer arcs and smaller state spaces. On the other hand, we would obviously like to maximize the fidelity of the model, which sometimes requires more nodes, arcs and states (although excess detail can also decrease accuracy). A tradeoff must be made between building a more accurate model and the cost of additional modeling.

When deciding on the structure of the network, the key is to focus on the **relationships** between variables. There are many types of qualitative understanding that can help determine the appropriate structure for a domain.

9.3.2.1 Causal relationships

The first, and most important, are the **causal** relationships. As we discussed earlier (see §2.4), while orienting the arcs in a causal direction is not required, doing so maximizes the representation of conditional independence, leading to a more compact, simpler model. To establish the causal relationships, the knowledge engineer must identify the variables that could cause a variable to take a particular state, or prevent it from taking a particular state. Once identified, arcs should be added from those causal variables, to the affected variable. In all the following examples, we will see a natural combination of variable elicitation with the identification of relationships between variables. Sometimes it is appropriate to ask direct questions about causes. For example:

Q: *"What can cause lung cancer?"*
A: *"Smoking and pollution."*
Modeling: suggests arcs from those nodes to the *LungCancer* node.

Q: *"Is there anything which prevents TB?"*
A: *"There is a TB immunization available."*
Modeling: suggests an arc from *Immunization* to *TB*.

Alternatively, the same cause-to-effect structure may be identified by asking about effects. For example:
Q: *"What are the effects of lung cancer?"*
A: *"Shortness of breath and a spot on the lungs that may show up on the X-ray."*
Modeling: suggests the arcs from *LungCancer* to *X-ray* and *Dyspnoea*.

Another kind of causal relationship is **prevention**, when an effect will occur *unless* a preventative action is taken first.
Q: *"Is there anything that can prevent HIV causing AIDS?"*
A: *"Anti-viral drugs such as AZT can prevent the development of AIDS."*
Modeling: suggests arcs from both *HIV* and *AZT* to *AIDS*.

Another way of looking at this is to consider the possibility of **interference** to the causal relationship.
Q: *"Is there any factor that might interfere with cholesterol-lowering medication treating heart disease?"*
A: *"Yes, if the patient doesn't modify her diet, the medication won't be effective."*
Modeling: suggests arcs from both *Medication* and *Diet* to *HeartDisease*.

We have already seen other ways of describing this sort of interference relationship in §5.4.4 when we looked at problems with a sensor that affect its measuring capacity, with terms such as **moderates** and **invalidates** being used to describe an **unless** condition.

Enabling relationships exist where the enabling variable may under certain conditions permit, enhance or inhibit the operation of a cause.
Q: *"Is anything else required to enable the cholesterol-lowering medication to be affective against heart disease?"*
A: *"Yes, the patient must also modify her diet."*
Modeling: again, suggests arcs from both *Medication* and *Diet* to *HeartDisease*.

As we discussed earlier in §2.3.1, a v-structure in Bayesian networks, with two parents sharing a common effect, gives rise to a form of reasoning called "explaining away." To investigate whether this substructure exists, the knowledge engineer should ask a series of questions around **explanations** such as:
Q: *"Are both X and Y possible explanations for Z?"*
Q: *"Would finding out that Z is true increase your belief that both X and Y may be true?"*
Q: *"Would then finding out that X is true undermine your previously increased belief in Y?"*
We look further at causal interactions in §9.3.4.

9.3.2.2 Dependence and independence relationships

As we know, Bayesian networks encode conditional dependency and independency relationships. Very generally, a dependency can be identified by asking the domain expert:

Q: *"Does knowing something about the value of one variable influence your beliefs as to the value of the other variable, and vice versa?"*

Once we obtain a positive answer to this, we must determine what kind of dependency exists. If two variables are dependent regardless of the values of all other variables, then they should be directly connected.

Alternatively, some pairs of variables will be dependent only through other variables, that is, they are only conditionally dependent. d-separation tests can be used to check that the encoded relationships agree with the domain expert's intuitions. Unfortunately, the concept of d-separation can be difficult to interpret and takes time for domain experts to understand. We have been involved in the development of Matilda [24], a software tool that can help domain experts explore these dependencies. It supports a visual exploration of d-separation in a network, supplemented by a non-technical explanation of possible interactions between variables.

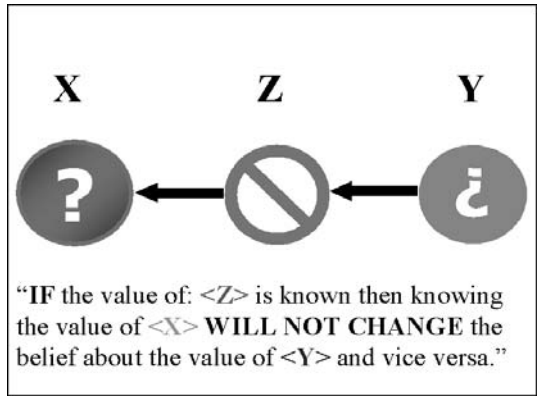

FIGURE 9.3
Matilda's visualization and explanation of "X is d-separated from Y given Z."

Matilda uses terms like 'direct causes,' 'causes' and 'paths' to explain graph relations. The term 'd-separation' is replaced by the more intuitive notion of 'blocking.' Extending this, a d-separating set found among parent nodes is called a 'simple blocking set.' This is a d-separating set but may contain proper subsets that can also d-separate. A minimal d-separating set found among parent nodes is called an 'essential blocking set.' This is a d-separating set from which no node can be removed without impairing the d-separation. A minimal d-separating set found anywhere in the network is called 'minimal blocking set.' Matilda's focus on the parent nodes

reflects the importance of these relationships in the modeling process.

In order to describe the general relation "X is d-separated from Y by Z" (i.e., $X \perp\!\!\!\perp Y \,|\, \mathbf{Z}$) Matilda gives the terms X, Y and Z a specific temporary role in the model. The nodes X become the **query** nodes, the nodes we want to reason about. The nodes Y become the **observation** nodes, the nodes we may observe. The nodes Z are **known** nodes, the values of which we already have. The notion of **influence**, in the context of "a change in the value of node X does not influence the value of node Y," is described by the term **change of belief** (*"knowing the value of node X does not change our belief in the value of node Y"*). To be sure, by giving the sets of nodes these roles the symmetry of the relation could be lost. To overcome this, the phrase *"vice versa"* needs to be added. Using these terms, Matilda describes the relation $X \perp\!\!\!\perp Y \,|\, \mathbf{Z}$ as:

> If the known nodes are observed, then knowing the value of the observation nodes will not change the belief about the value of the query nodes and vice versa.

Matilda's visualization of the relation "X is d-separated from Y given Z," with the corresponding verbal explanation is shown in Figure 9.3. The blocking relationship is shown using a common symbol for 'stop' or 'not allowed' (in red), for the *Z* node, with query nodes X indicated by "?" (purple), and observation nodes *Y* by an upside down "?" (blue). The color scheme ties the verbal and visual explanations together*.

Matilda allows the user to ask various types of questions about the relationships between variables. A BN for a fire alarm domain (extending one used in [219]), shown in Figure 9.4, will be used to illustrate this process.

Fire alarm example

We receive a phone call saying that everyone is leaving the building. The call can come from three different sources: a security center gets a report from a special sensor in the building. If the sensor reports 'Leaving' the security center calls us. We also get calls from kids who are playing practical jokes (mainly during the holidays) as well as from seriously concerned people who notice smoke coming out of the building (mainly after work-hours). The sensor is noisy. It sometimes does not report when everyone is leaving and sometimes reports leaving for no reason. If the fire alarm goes off, that causes people in the building to leave. The fire alarm can go off either because there is a fire or because someone tampers with it. The fire also causes smoke to rise from the building. In winter, fireplaces in the building can also cause the presence of smoke.

Matilda's Type 1 question. What is the relationship between two nodes?

This option allows the user to ask "What information d-separates two selected nodes?" by selecting two nodes *X* and *Y* and choosing between the three types of d-separating

*Note that we cannot reproduce the color scheme in this text.

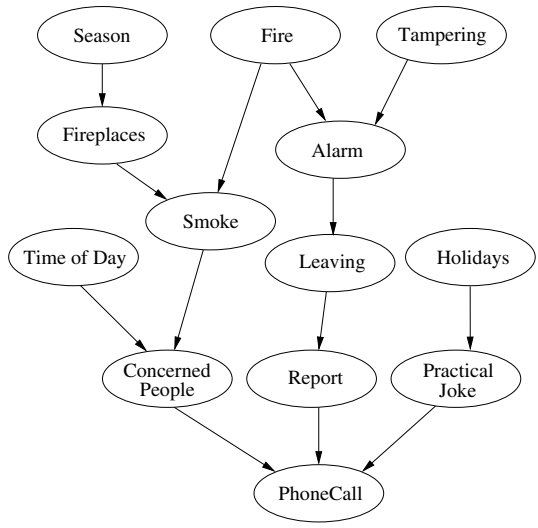

FIGURE 9.4
A BN solution for the fire alarm example.

sets: simple, essential and minimal blocking sets (as described above). Matilda then computes the selected type of d-separating set and highlights it using the blocking symbol.

Example: selected *X=Alarm*, *Y=Phone Call* (Figure 9.5).

A simple blocking set is the set of parents of *Phone Call*: {*Concerned People*, *Report*, *Practical Joke*}. The verbal explanation is: "IF the value of: <Practical Joke>, <Report>, <Concerned People> is known then knowing the value of <Alarm> WILL NOT CHANGE the belief about the value of <Phone Call> and vice versa."

An essential blocking set is the set {*Concerned People*, *Report*}. The verbal explanation is: "IF the value of: <Report>, <Concerned People> is known then knowing the value of <Alarm> WILL NOT CHANGE the belief about the value of <Phone Call> and vice versa."

The possible minimal blocking sets are: {*Fire*, *Report*}, {*Fire*, *Leaving*}, {*Report*, *Concerned People*}, {*Report*, *Smoke*}, {*Concerned People*, *Leaving*}, {*Leaving*, *Smoke*}. The verbal explanation of the first one is: "IF the value of: <Fire>, <Report> is known then knowing the value of <Alarm> WILL NOT CHANGE the belief about the value of <Phone Call> and vice versa."

Matilda's Type 2 question. When does a node become irrelevant?

Here, Matilda visualizes the relationships between one node and the rest of the network. This option allows the user to select a single node *X* and ask for a set of nodes that d-separates ("blocks" in Matilda terminology) this node from the rest of the structure. The number of such sets is potentially exponential, so Matilda highlights

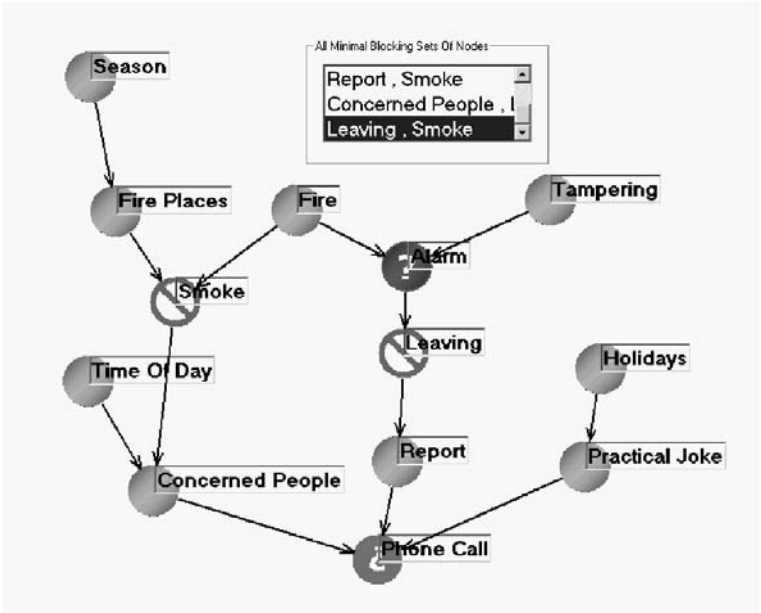

FIGURE 9.5
Matilda's Type 1 visualization of d-separation in the fire alarm example.

only one set, namely the Markov blanket (see §2.2.2).
Example: selected *X*=*Report* (Figure 9.6).

The Markov blanket for this node is the set {*Leaving* (parent), *Phone Call* (child), *Practical Joke*, *Concerned People* (child's parents)}. The verbal explanation is: "IF the value of: <Leaving>, <Phone Call>, <Practical Joke>, <Concerned People> is known then knowing the value of <Report> WILL NOT CHANGE the belief about the value of any other node and vice versa."

Matilda's Type 3 question. Given some information, what happens to the relationships between nodes?

Here, Matilda visualizes the relationships between *sets* of nodes. This option allows the user to select a set of nodes *X* (the query nodes) and a set of nodes *Z* (the prior information) and request the set of all *Y* nodes that are d-separated (blocked) from *X*. Matilda highlights all the nodes *Y* in response.
Example: selected *X*={*Phone Call*}, with prior information for nodes *Z*={*Smoke*, *Fire*} (Figure 9.7).

The nodes that are d-separated from *Phone Call* by *Smoke* and *Fire* are *Fire Places* and *Seasons*. The verbal explanation is "IF the value of: <Fire>, <Smoke> is known then knowing the value of <Phone Call> WILL NOT CHANGE the belief about the value of <Season>, <Fire Places> and vice versa."

In short, for various types of query, Matilda visualizes the relation "X is d-separated

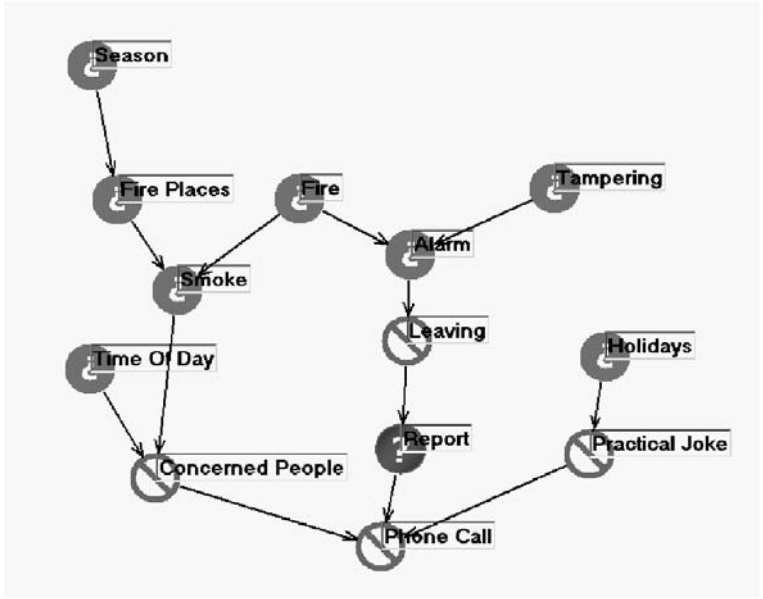

FIGURE 9.6
Matilda's Type 2 visualization of a Markov blanket for *Report*.

from Y given Z." In the first type, the user chooses *X* and *Y*; in response the tool highlights *Z*. In the second question type, the user chooses *X*; in response the tool highlights *Z*. In the third question type, the user chooses *X* and *Z*; in response the tool highlights *Y*. Note that in the first two types of queries the tool identifies the d-separating sets of nodes, whereas in the latter type of query the user is asking the question with regard to a specific d-separating set of nodes.

Case-studies [24] suggest that Matilda is useful in understanding networks as they are being built and in understanding the consequences of different possible design choices. It can be used not just by BN experts, but also by domain experts to validate the network and identify potential problems with the structure. It can be used to investigate an existing network at any stage during development. It is especially helpful in investigating networks built by automated methods, when prior intuitive understanding may be weak.

9.3.2.3 Other relationships

There are other less explicit indications of the correct network structure.

Association relationships occur when knowing a value of one variable provides information about another variable. By Reichenbach's Principle of the Common Cause, some causal nexus must explain the association; although any active (unblocked) path will do this, just the information that there is *some* active path may serve as a beginning in building the causal structure. The absence of an uncondi-

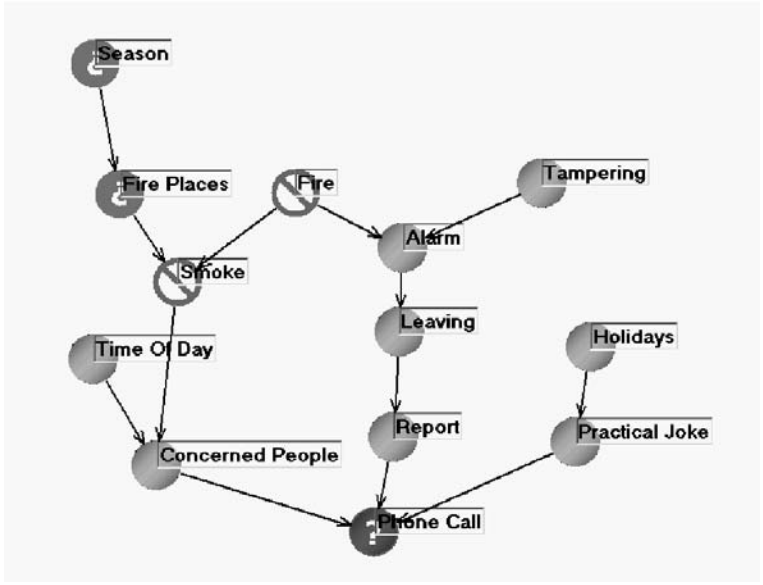

FIGURE 9.7
Matilda's Type 3 visualization.

tional association is also useful to know. Thus, in the fire alarm example of Figure 9.4, there is no marginal dependence between *Time of Day* and *Fire*, which is an implication worth checking. On the other hand, there is an implied association between *Report* and *Smoke* (through their common ancestor *Fire*).

In many domains, there may be a known **temporal ordering** of variables, where one event or value change occurs *before* another. The known ordering may be either total or partial. In either case, the temporal information will restrict the orientation of some of the arcs, assuming you are building a causal network.

9.3.2.4 Combining discrete and continuous variables

The initial work on BNs with continuous variables [217, 251] only allowed variables with linear Gaussian distributions. A method for combining discrete and continuous variables was proposed [168] and implemented in cHugin [210]; this approach did not allow for discrete children of continuous parents. The usual solution is to discretize all continuous variables at some stage. As suggested above, this is best done manually at the moment. In the future, we expect the best discretization methods will be incorporated in Bayesian network tools, so that the process can be automated, given sufficient statistical data.

There has also been some research on extending the inference algorithms to cope with some special cases of continuous variables and inference, and these no doubt will eventually be applied in application tools (e.g., [170]).

9.3.3 Probabilities

The parameters for a BN are a set of conditional probability distributions of child values given values of parents. There is one distribution for each possible instantiation of parent variables; so the bad news is that the task of probability assessment is exponential in the number of parent variables. If there is local structure (see §7.4), of course, the number of parameters to be estimated is reduced.

9.3.3.1 Parameter sources

There are three possible parameter sources.

1. Data

We have previously described some specific methods for learning parameters from domain data (see Chapter 7). General problems with data include: noise, missing values and small samples. These can be overcome to a certain extent by using robust data mining techniques. It is also useful to instrument the domain for collecting data to use in the future, by adaptation of parameters.

2. Domain Experts

The most basic problem is finding suitable experts who have the time and interest to assist with the modeling process. Another difficulty is that humans, including expert humans, almost always display various kinds of bias in estimating probabilities. These include:

- **Overconfidence**, the tendency to attribute higher than justifiable probabilities to events that have a probability sufficiently greater than 0.5. Thus, an event which objectively has a probability of 0.9 will usually be attributed a probability that is somewhat higher (see Figure 9.8). To be sure, expertise itself has been found to have a moderating effect on this kind of miscalibration [239].
- **Anchoring**, the tendency for subsequent estimates to be "weighed down" by an initial estimate. For example, if someone is asked to estimate the average age of coworkers and begins that process by estimating the average age of those in a meeting, a high (or low) age at the meeting will very probably bias the estimate upwards (or downwards) [137].
- **Availability**, that is, assessing an event as more probable than is justifiable, because it is easily remembered or more salient [278].

There is a large, and problematic, literature on assessing these biases and proposals to debias human probability estimates. The latter have met with highly limited success. The best advice we can give, short of an in-depth exploration of the literature (and we do recommend some such exploration, as described in §9.6 below), is to be aware of the existence of such biases, discuss them with the experts who are being asked to make judgments and to take advantage of whatever statistics are available, or can be made available, to test human judgments against a more objective standard.

Two attributes of a good elicitation process are (adapted from Morgan and Henrion [194, pp. 158-159]):

1. The expert should be apprised of what is known about the process, especially the nearly universal tendency to overconfidence and other forms of bias. In

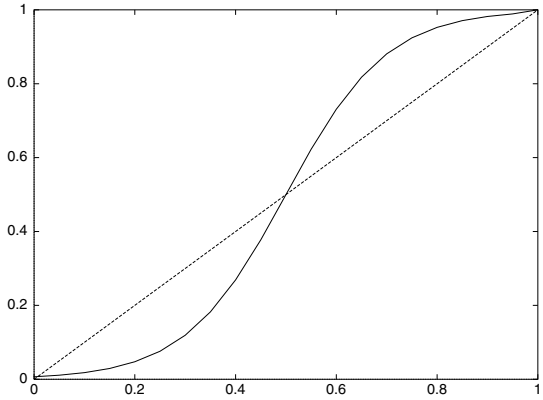

FIGURE 9.8
Overconfidence curve: subjective probability (vertical axis) vs. objective probability (horizontal axis).

order to avoid some of the problems, values should be elicited in random order and the expert *not* given feedback on how the different values fit together until a complete set has been elicited.

2. The elicitation process is not simply one of requesting and recording numbers, but also one of refining the definitions of variables and terms to be used in the model. What values are elicited depends directly upon the interpretation of terms and these should be made as explicit as possible and recorded during the elicitation. This is a part of the process management described earlier in §9.2.4.

3. The Literature

There may be a published body of knowledge about the application domain. One common problem with published statistics is sparseness. For example, in a medical diagnosis domain, the probability of observing a symptom given a disease, $P(Symptom|Disease = T)$, may be available in the medical textbooks ("80% of patients with TB will present with a cough"), but not $P(Symptom|Disease = F)$, the information about the frequency of the symptom occurring when the disease is not present. There is also a bias in what information is available in the literature: the fact of its publication reflects interest. These problems can be moderated by using expert opinion to review the proposed parameterization.

There is a risk involved with combining parameters from different sources, with different biases. It is not necessarily the case that combining sources with multiple biases smoothly averages out to no bias, or to a readily measurable and manageable bias. For an informal example, one author shot baskets at the Exploratorium in San Francisco with some biased goggles. At first, this resulted in missing to the left, but fairly soon the brain accommodated the misdirection and the basketball started hitting the hoop again. As soon as the goggles were removed, however, the shots

missed to the right! The unbiased new input was being combined with an older bias, leading to error.

9.3.3.2 Probability elicitation for discrete variables

For discrete variables, one approach is direct elicitation, where an expert provides a number such as "the probability is 0.7." However, given the problems people have with such estimation, noted above, other elicitation techniques might be considered. People are often better at providing frequencies rather than probabilities, such as "1 in 4" or "1 in 10,000" [92], especially for situations where the probabilities involved are very large or very small. Assessing extreme probabilities directly is difficult, and orders of magnitude assessments might be tried.

Assessing by odds is often useful. For example, given that $Y = y$, a domain expert may report:

"It is three times more likely that variable X *has the value* x_1 *than* x_2*."*

This information gives the equation $P(X = x_1|Y = y) = 3P(X = x_2|Y = y)$, and only one probability has to be elicited.

An alternative approach is to use qualitative assessment, where the domain expert describes the probability in common language, such as "very high" or "unlikely." Such probability elicitation using a scale with numerical and verbal anchors is described in [280]. The verbal cues in that scale were `certain`, `probable`, `expected`, `fifty-fifty`, `uncertain`, `improbable` and `impossible`. Having been translated from Dutch some of these cues are inappropriate, for example "`uncertain`" is ambiguous and could be replaced with "`unlikely`." Because these qualitative phrases can be ambiguous (in fact, this is the problem of **linguistic uncertainty**), they can cause miscommunication, especially where more than one domain expert is involved, unless they are themselves calibrated. It is advisable to do the mapping of verbal levels to actual probabilities (called the **verbal map**) separately from the probability elicitation exercise. The verbal map should be customized to suit the individual expert. We have developed a software tool called VE (for Verbal Elicitor) in conjunction with the Netica BN software (see §B.4.8), which supports the qualitative elicitation of probabilities and verbal maps [109].

An example of VE's main window for eliciting qualitative probabilities is shown in Figure 9.9, together with the window editing the verbal map for this tool. The probabilities associated with each verbal cue are set using a slider bar, and other verbal cues can be added if desired.

A common problem when eliciting probabilities is that an expert may specify probabilities that are **incoherent**, failing to satisfy the probability axioms. For example, suppose the verbal mapping shown in Figure 9.9 is being used and that the expert provides the qualitative assessment of the probabilities for X-ray results given in Table 9.3.

These probabilities are incoherent, since they do not sum to one for each conditioning case. VE provides a function to correct incoherent probabilities automatically; it uses an iterative, linear optimization to find verbal mappings that are coherent and as close as possible to the original map.

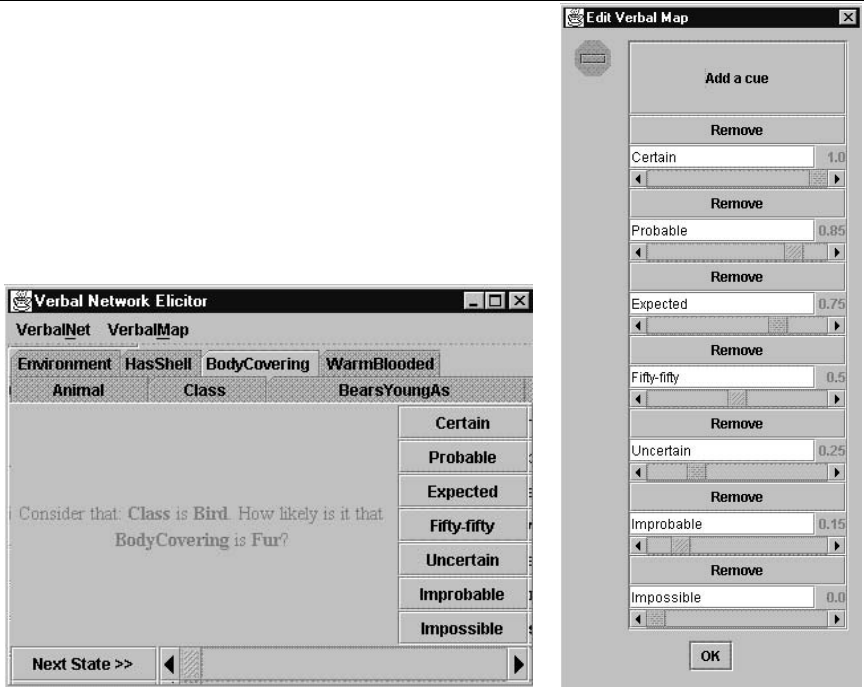

FIGURE 9.9
VE: main elicitation window (left) and the verbal map editing window (right).

9.3.3.3 Probability elicitation for continuous variables

The easiest continuous variables to parameterize are those which are distributed according to a **parametric model**, that is, those which are fully characterized by a limited number of parameters. The most common example is the Normal distribution (also known as the Gaussian or "bell curve"), $N(\mu, \sigma^2)$, which is characterized by just two parameters, the mean μ and the variance σ^2. Normal distributions are used to model: noisy measurement processes (e.g., velocities, the positions of stars); the central tendencies (average values) of almost any process (e.g., average age or income of samples from a population), which is justified by the central limit theorem of probability theory; and, by a kind of metaphorical extension, summative measures of

TABLE 9.3
Incoherent qualitative assessments for the *X-ray* CPT

$P(X = Pos \mid Cancer = T)$ = Probable
$P(X = Neg \mid Cancer = T)$ = Improbable
$P(X = Pos \mid Cancer = F)$ = Uncertain
$P(X = Neg \mid Cancer = F)$ = Certain

a complex of processes when the individual processes are not well understood (e.g., IQ). Another popular parametric distribution is the exponential, which is often used to model life- and death-processes, such as the life span of an electrical part. Still other parametric continuous distributions are the Gamma, Chi-square and F distributions. These, and the discrete parametric distributions (e.g., binomial and Poisson), form much of the material of probability and statistics classes. They are worth learning about since in every case one need only obtain a good estimate of only a few parameters from an expert in order to obtain a good estimate of the entire probability distribution — assuming, of course, that the variable in question is properly modeled by the chosen family of distributions!

If the problem is not as simple as estimating a couple of parameters, like the mean and variance of a normal distribution, then most common is to elicit estimates of key values for the probability density function. Recall from §1.3.2 that if the continuous density function is $f(x)$, then the cumulative distribution function $F(x)$ is

$$F(x) = P(X \leq x) = \int_{x' \leq x} f(x')dx' \tag{9.1}$$

A likely route to estimating the density function is by bi-sectioning it. First, elicit the median, the value at which X is equally likely to be found either above or below. Then elicit the 25th percentile by "bisecting" the region below the median into two further equally likely regions, and then the 75th percentile analogously. This process can be continued until the density has been sufficiently well refined for the problem, or until the expert can no longer make meaningful distinctions. This kind of estimation may be usefully accompanied by the expert simply sketching her or his impression of the shape of the density function; otherwise one might overlook something simple and important, such as whether the density is intended to be unimodal or bimodal, skewed or symmetric, etc.

Having estimated a distribution in this fashion, it may be best and simplest to find a parametric model (with parameter values) which reproduces the estimated distribution reasonably well and use that for your Bayesian network model.

9.3.3.4 Support for probability elicitation

Visual aids are known to be helpful and should be used for probability elicitation (see Figure 9.10). With a **pie chart** the expert aims to size a slice of the "pie" so that a spinner will land in that region with the probability desired. A **histogram** may help the expert to order discrete events by probability. As we mentioned, simple freehand drawings of probability distributions can also be informative.

Lotteries can be used to force estimates of either probabilities or utilities, in techniques going back to Ramsey [231]. Given clear utility values, say dollars, you can elicit someone's estimated probability of an uncertain event E by finding at what point the person is indifferent between two gambles: the first one paying, say, \$100 if E comes true; the second one paying, say, \$1 million if a (free) lottery ticket *Wins*. Since the two gambles are considered equivalued, we have (where N is the number

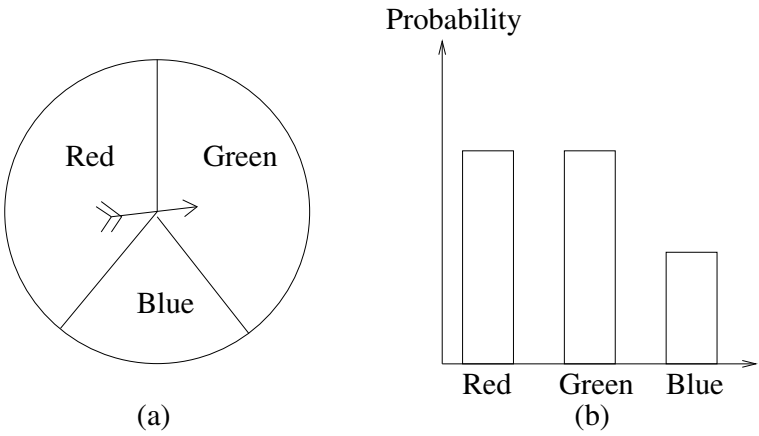

FIGURE 9.10
Examples of visual aids for probability elicitation: (a) pie chart; (b) histogram.

of lottery tickets required to reach indifference):

$$EU(Gamble) = \$100P(E) = EU(Lottery) = \frac{\$1000000}{N} \tag{9.2}$$

Hence, $P(E) = 10000/N$. Lotteries can be used analogously to elicit unclear utilities for an outcome state by manipulating the probability of reaching that state until the expert is indifferent between the proposed gamble and some lottery ticket with known value and probability of winning.

Our VE software tool provides a number of useful automated functions that facilitate probability elicitation. One function normalizes CPTs, allowing users to specify ratios in place of probabilities[†]. For example, if the expert thinks that someone has three times the chance of getting cancer as not getting cancer (say if they are a smoker), they can specify

$$P(Cancer = T|Smoker = T) : P(cancer = F|Smoker = F)$$

as 3:1, which the tool translates into the probabilities 0.75 and 0.25, respectively.

VE can also perform a maximum entropy fill of CPTs, where the remaining probabilities are filled in uniformly with the probability remaining after subtracting supplied probabilities from 1. This means the expert need only provide probabilities for combinations about which s/he is confident. For example, if the variable A has states a_1, a_2 and a_3, and the probability for $A = a_1$ given some combination of values for its parent variables, $P(A = a_1|B = T, C = F)$ is set to 0.95, then $P(A = a_2|B = T, C = F)$ and $P(A = a_3|B = T, C = F)$ will both be set to 0.025 automatically.

[†]Note that some other software packages have this feature — see §B.4.

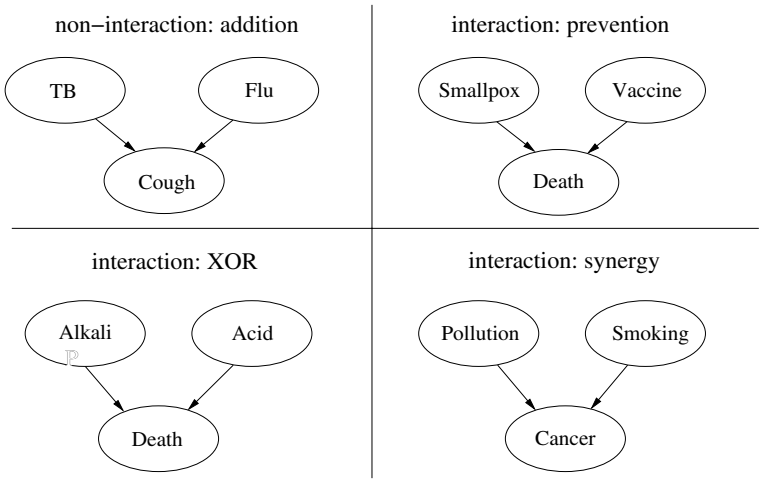

FIGURE 9.11
Different qualitative causal relationships.

9.3.4 Local structure

When parameterizing the relation between parents and a child node, the possibility of there being "local structure" was discussed in Chapter 7 in the context of automated parameter learning. It is also of interest in the elicitation process, of course. There are various *kinds* of causal interaction, such as those displayed in Figure 9.11. A classic example of interaction between causes is **XOR**, where each cause cancels the other out. The alkali/acid case (§7.4.1) is an example of this: *one might ingest alkali, and die; one might instead ingest acid, and die; but if one ingests both alkali and acid together, then one may well not die.*

Other causal interactions include **prevention**, where one causal factor intervenes to stop another, such as a *Vaccine* preventing *Smallpox* leading to *Death*. And again there is the possibility of **synergy**, where the effects are reinforced by the occurrence of both causes beyond the mere addition of the effects independently. All of these relationships can be looked for explicitly during an elicitation process. For example,
Q: *"Given that* Acid *and* Alkali *are independently causes of* Death*, when taken jointly what happens to the risk?"*
A: *"It is decreased."*
Modeling: in this case, the causal interaction is clearly an XOR type.

These kinds of interaction imply that the probabilities associated with one or more of the possible instantiations of the parents are *independent* of the probabilities associated with the other parent instantiations. For example, knowing what happens when you ingest Acid but not Alkali tells you little or nothing about what happens when you ingest both.

Local structure is the opposite situation: there is some structure across the different parent instantiations that allows you to infer some probabilities from the others. We

have already discussed some different models of non-interaction (local structure) in Chapter 7, namely noisy-or, logit models and classification tree models, all of which allow a more compact specification of the CPT under non-interaction. In our original noisy-or example of *Flu*, *TB* and *SevereCough* (in Figures 7.4 and 9.11), this relationship would be identified by negative answers to the questions:
Q: *"Does having* TB *change the way that* Flu *causes a* Severe Cough*?"*
A: *"No."*
Q: *"Similarly, does* Flu *change the way that* TB *causes a* Severe Cough*?"*
A: *"No."*

Assuming that *Flu* and *TB* have been identified as causes of *Severe Cough*, these answers imply that the probability of not having the symptom is just the product of the independent probabilities that every cause present will fail to induce the symptom. (This is illustrated in Table 7.2 and explained in the surrounding text.) Given such a noisy-or model, we only need to elicit three probabilities: namely, the probability that *Flu* will fail to show the symptom of *Severe Cough*, the probability that *TB* will fail to show the symptom and the background probability of not having the symptom.

Local structure, clearly, can be used to advantage in either the elicitation task or the automated learning of parameters (or both).

One method for eliciting local structure is "elicitation by partition" [102, 90, 181]. This involves dividing the joint states of parents into subsets such that each subset shares the conditional probability distribution for the child states. In other words, we **partition** the CPT, with each subset being a **partition element**. The task is then to elicit one probability distribution per partition element. Suppose, for example, that the probability of a high fever is the same in children, but not adults, for both the flu and measles. Then the partition for the parent variables *Flu*, *Measles* and *Age* and effect variable *Fever* would produce two partition elements for adults and one for children.

Note that this elicitation method directly corresponds to the use of classification trees and graphs in automated parameter learning (see §7.4.3). So, one way of doing partitioning is by building that corresponding classification tree by hand. Figure 9.12 illustrates this possibility for a simple network.

9.3.4.1 Divorcing

Another way to reduce the number of parameters is to alter the graph structure; **Divorcing multiple parents** is a useful technique of this type. It is typically applied when a node has many parents (and so a large CPT), and when there are likely groupings of the parents in terms of their effect on the child. Divorcing means introducing an intermediate node that summarizes the effect of a subset of parents on a child.

An example of divorcing is shown in Figure 9.13; the introduction of variable Z divorces parent nodes A and B from the other parents C and D. The method of divorcing parents was used in Munin [9]. Divorcing involves a trade-off between new structural complexity (the introduction of additional nodes and arcs) and parameter simplification. We note that divorcing may also be used for purposes other than reducing the number of parameters, such as dealing with overconfidence [209].

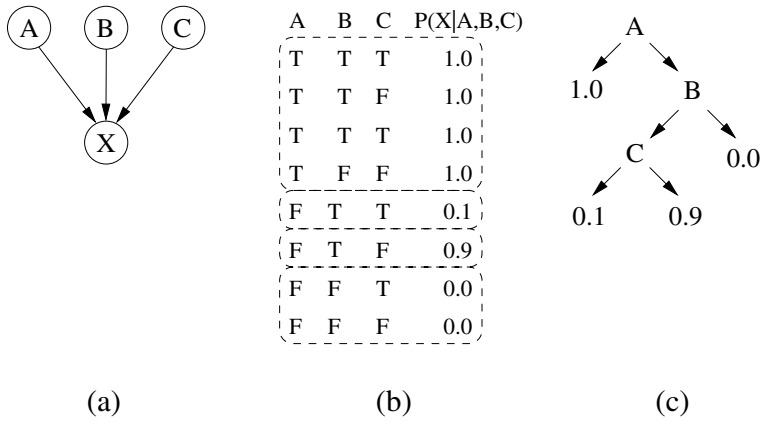

FIGURE 9.12
Local CPT structure: (a) the Bayesian network; (b) the partitioned CPT; (c) classification tree representation of the CPT.

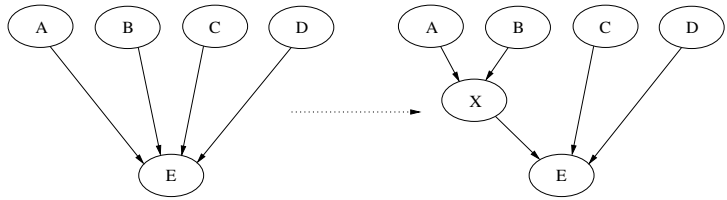

FIGURE 9.13
Divorcing example.

Divorcing example: search and rescue

Part of the land search and rescue problem is to predict what the health of a missing person will be when found. Factors that the domain expert initially considered were external factors, such as the temperature and weather, and features of the missing person, such as age, physical health and mental health.

This example arises from an actual case of Bayesian network modeling [24]. A BN constructed by the domain expert early in the modeling process for a part of the search and rescue problem is shown in Figure 9.14(a). The number of parameters required for the node *HealthWhenFound* is large. However, there is a natural grouping of the parent nodes of this variable into those relating to weather and those relating to the missing person, which suggests the introduction of intermediate nodes. In fact, it turns out that what really matters about the weather is a combination of temperature and weather conditions producing a dangerous wind chill, which in turn may lead to hypothermia. The extent to which the person's health deteriorates due to hypothermia depends on *Age* and the remaining personal factors, which may be summarized by a *ConditionWhenLost* node, hence, the introduction of three mediating variables,

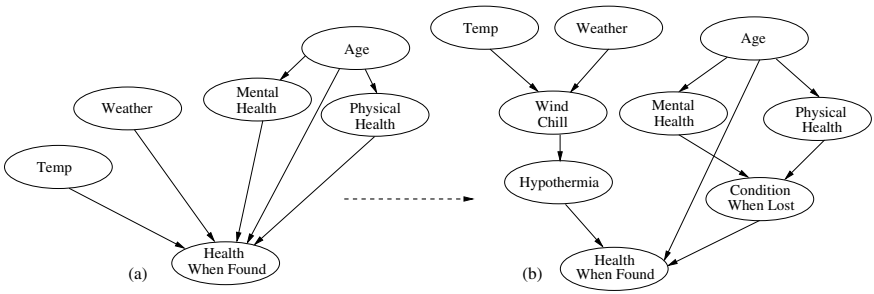

FIGURE 9.14
Search and rescue problem: (a) the original BN fragment; (b) the divorced structure.

WindChill, *Hypothermia* and *ConditionWhenLost*, as shown in Figure 9.14(b). The result is a structure that clearly reduces the number of parameters, making the elicitation process significantly easier[‡].

9.3.5 Variants of Bayesian networks

9.3.5.1 Qualitative probabilistic networks (QPN)

Qualitative probabilistic networks (QPN) [296] provide a qualitative abstraction of Bayesian networks, using the notion of positive and negative influences between variables. Wellman shows that QPNs are often able to make optimal decisions, without requiring the specification of the quantitative part of the BN, the probability tables.

9.3.5.2 Object-oriented BNs (OOBNs)

Object-oriented BNs are a generalization of BNs proposed by Koller and Pfeffer [151]. They facilitate network construction with respect to both structure and probabilities by allowing the representation of commonalities across variables. Most important for probability elicitation, they allow inheritance of priors and CPTs. However, OOBNs are not supported by the main BN software packages (other than Hugin Expert) and hence are not as yet widely used[§].

[‡] This example comes from a case study undertaken when evaluating Matilda [24]. Examination of the initial structure with Matilda led to the expert identifying two missing variables, *Wind Chill* and *Hypothermia*. (Note that these two could be combined in the network shown without any loss of information; this would not work, however, in the larger network.) Subsequent analysis led to the divorce solution.

[§] Other packages do support subnetwork structures, see §B.4.

9.3.6 Modeling example: missing car

We will now run through the modeling process for an example we have used in our BN courses, the missing car problem (Figure 9.15)[¶]. This is a small problem, but illustrates some interesting modeling issues.

The first step is to highlight parts of the problem statement which will assist in identifying the Bayesian network structure, which we have already done in Figure 9.15. For this initial analysis, we have used the annotations:

- possible situations (→ nodes and values)
- **words connecting situations** (→ graphical structure)
- indication of evidence (→ observation nodes)
- `focus of reasoning` (→ query nodes)

The underlined sections suggest five Boolean nodes, shown in the table in Figure 9.15, consisting of two `query` nodes and three observation nodes.

Marked-Up Problem Statement

John and Mary Nguyen arrive home after a night out to find that their second car is not in the garage. Two **explanations** *occur to them:* **either** *the car has been stolen* **or** *their daughter Sam has borrowed the car without permission. The Nguyens know that* **if** *the car was stolen,* **then** *the garage will probably show signs of forced entry. The Nguyens also know that Sam has a busy social life, so* **even if** *she didn't borrow the car, she* **may be** *out socializing.* `Should they be worried about their second car being stolen and notify the police, or has Sam just borrowed the car again?`

Preliminary Choice of Nodes and Values

Type	Description	Name	Values
Query	Has Nguyen's car been stolen?	*CarStolen*	*T,F*
	Has Sam borrowed the car?	*SamBorrowed*	*T,F*
Observation	See 2nd car is missing	*MissingCar*	*T,F*
	See signs of forced entry to garage?	*ForcedEntry*	*T,F*
	Check whether or not Sam is out?	*SamOut*	*T,F*

FIGURE 9.15

Missing car problem: preliminary analysis.

There are two possible **explanations**, or **causes**, for the car being missing, suggesting causal arcs from *CarStolen* and *SamBorrowed* to *MissingCar*. Signs of forced entry are a likely **effect** of the car being stolen, which suggests an arc from *CarStolen* to *ForcedEntry*. That leaves only *SamOut* to connect to the network. In the problem as stated, there is clearly an association between Sam being out and Sam borrowing the car; which way should the arc go? Some students add the arc

[¶] The networks developed by our students are available from the book Web site.

$SamOut \longrightarrow SamBorrowed$ because that is the direction of the inference or reasoning. A better way to think about it is to say "Sam borrowing the car *leads to* Sam being out," which is both a causal and a temporal relationship, to be modeled by $SamBorrowed \longrightarrow SamOut$.

The BN constructed with these modeling choices is shown in Figure 9.16(a). The next step of the KEBN process is to determine the parameters for this model; here we underline text about the "numbers."

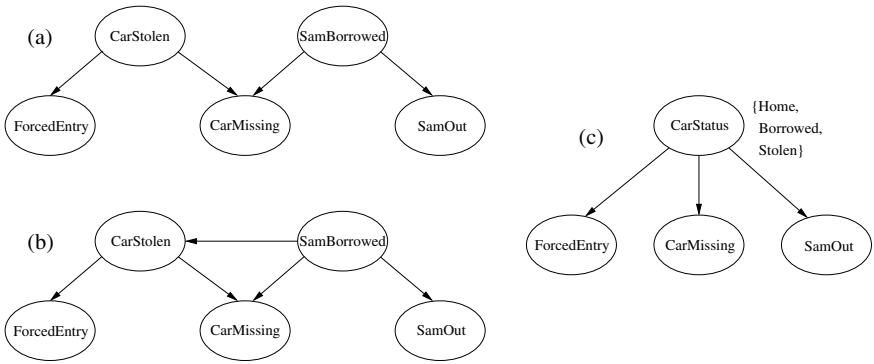

FIGURE 9.16
Alternative BNs for the missing car problem.

Additional problem information

The Nguyens know that the rate of car theft in their area is about 1 in 2000 each day, and that if the car was stolen, there is a 95% chance that the garage will show signs of forced entry. There is nothing else worth stealing in the garage, so it is reasonable to assume that if the car isn't stolen, the garage won't show signs of forced entry. The Nguyens also know that Sam borrows the car without asking about once a week, and that even if she didn't borrow the car, there is a 50% chance that she is out.

There are two root nodes in the model under consideration (Figure 9.16(a)). The prior probabilities for these nodes are given in the additional information:

- $P(CarStolen)$ is the rate of car theft in their area of "1 in 2000," or 0.0005
- $P(SamBorrowed)$ is "once a week," or $\approx$0.143

The conditional probabilities for the*ForcedEntry* node are also straightforward:

- $P(ForcedEntry = T|CarStolen = T) = 0.95$ (from the "95% chance")
- $P(ForcedEntry = T|CarStolen = F) = 0$

In practice, it might be better to leave open the possibility of there being signs of forced entry even though the car hasn't been stolen (something else stolen, or thieves interrupted), by having a very high probability (say 0.995) rather than the 1;

this removes the possibility that the implemented system will grind to a halt when confronted with a combination of impossible evidence (say, $CarStolen = F$ and $ForcedEntry = T$). There is a stronger case for

- $P(SamOut = T|SamBorrowed = T) = 1$

The new information also gives us

- $P(SamOut = T|SamBorrowed = F) = 0.5$

So the only CPT parameters still to fill in are those for the *MissingCar* node. But the following probabilities are clear:

- $P(MissingCar = T|CarStolen = T, SamBorrowed = F) \approx 1$
- $P(MissingCar = T|CarStolen = F, SamBorrowed = T) \approx 1$

If we want to allow the possibility of another explanation for the car being missing (not modeled explicitly with a variable in the network), we can adopt

- $P(MissingCar = T|CarStolen = F, SamBorrowed = F) = \alpha > 0$

Finally, only one probability remains, namely

- $P(MissingCar = T|CarStolen = T, SamBorrowed = T)$

Alert! Can you see the problem? How is it possible for Sam to have borrowed the car *and* for the car to have been stolen? These are mutually exclusive events!

The first modeling solution is to add an arc $SamBorrowed \longrightarrow CarStolen$ (or vice versa)(see Figure 9.16(b)), and make this mutual exclusion explicit with

- $P(CarStolen = T|SamBorrowed = T) = 0$

In which case, it doesn't matter what numbers are put in the CPT for $P(Missing Car = T|CarStolen = T, SamBorrowed = T)$. While this "add-an-arc" modeling solution "works," it isn't very elegant. Although Sam's borrowing the car will prevent it from being stolen, it is equally true that someone's stealing the car will prevent Sam's borrowing it!

A better solution is to go further back in the modeling process and re-visit the choice of nodes. The Nguyens are actually interested in the state of their car, whether it has been borrowed by their daughter, stolen, or is safe at home. So instead of two Boolean query nodes, we should have a single query node *CarStatus* with possible values $\{home, borrowed, stolen\}$. This simplifies the structure to that shown in Figure 9.16(c). A disadvantage of this simplified structure is that it requires additional parameters about other relationships, such as between signs of forced entry and Sam borrowing the car, and the car being stolen and Sam being out.

9.3.7 Decision networks

Since the 1970s there have been many good software packages for decision analysis. Such tools support the elicitation of decisions/actions, utilities and probabilities, building decision trees and performing sensitivity analysis. These areas as well covered in the decision analysis literature (see for example, Raiffa's *Decision Analysis* [229], an excellent book!).

The main differences between using these decision analysis tools and knowledge engineering with decision networks are: (1) the scale — decision analysis systems tend to require tens of parameters, compared to anything up to thousands in KEBN; and (2) the structure, as decision trees reflect straight-forward state-action combinations, without the causal structure, prediction and intervention aspects modeled in decision networks.

We have seen that the KE tasks for ordinary BN modeling are deciding on the variables and their values, determining the network structure, and adding the probabilities. There are several additional KE tasks when modeling with decision networks, encompassing decision/action nodes, utility (or value) nodes and how these are connected to the BN.

First, we must model what decisions can be made, through the addition of one or more decision nodes. If the decision task is to choose only a single decision at any one time from a set of possible actions, only one decision node is required. A good deal can be done with only a single decision node. Thus, a single *Treatment* decision node with options $\{$*medication*, *surgery*, *placebo*, *no-treatment*$\}$ precludes consideration of a combination of surgery and medication. However, combinations of actions can be modeled within the one node, for example, by explicitly adding a *surgery-medication* action. This modeling solution avoids the complexity of multiple decision nodes, but has the disadvantage that the overlap between different actions (e.g., medication and surgery-medication) is not modeled explicitly.

An alternative is to have separate decision nodes for actions that are not mutually exclusive. This can lead to new modeling problems, such as ensuring that a "no-action" option is possible. In the treatment example, the multiple decision node solution would entail 4 decision nodes, each of which represented the positive and the negative action choices, e.g., $\{$*surgery*, *no-surgery*$\}$. The decision problem becomes much more complex, as the number of combinations of actions is $2^4 = 16$. Another practical difficulty is that many of the current BN software tools (including Netica) only support decision networks containing either a single one-off decision node or multiple nodes for sequential decision making. That is, they do not compute optimal *combinations* of decisions to be taken at the same time.

The next KE task for decision making is to model the utility of outcomes. The first stage is to decide what the unit of measure ("utile") will mean. This is clearly domain specific and in some cases fairly subjective. Modeling a monetary cost/benefit is usually fairly straightforward. Simply adopting the transformation \$1 = 1 utile provides a linear numeric scale for the utility. Even here, however, there are pitfalls. One is that the utility of money is not, in fact, linear (as discussed in §4.2): the next dollar of income undoubtedly means more to a typical student than to a millionaire.

When the utile describes subjective preferences, things are more difficult. Requesting experts to provide preference orderings for different outcomes is one way to start. Numeric values can be used to fine tune the result. As noted previously, hypothetical lotteries can also be used to elicit utilities. It is also worth noting that the domain expert may well be the wrong person for utility elicitation, either in general or for particular utility nodes. It may be that the value or disvalue of an outcome arises largely from its impact on company management, customers or citizens at large. In such cases, utilities should be elicited from those people, rather than the domain experts.

It is worth observing that it may be possible to get more objective assessments of value from social, business or governmental practice. Consider the question: What is the value of a human life? Most people, asked this question, will reply that it is impossible to measure the value of a human life. On the other hand, most of *those* people, under the right circumstances, will go right ahead and measure the unmeasurable. Thus, there are implicit valuations of human life in governmental expenditures on health, automobile traffic and air safety, etc. More explicitly, the courts frequently hand down judgments valuing the loss of human life or the loss of quality of life. These kinds of measurement should not be used naively — clearly, there are many factors influencing such judgments — but they certainly can be used to bound regions of reasonable valuations. Two common measures used in these kinds of domains are the **micromort** (a one in a million chance of death) and the **QALY**, a quality-adjusted life year (equivalent to a year in good health with no infirmities).

The knowledge engineer must also determine whether the overall utility function consists of different value attributes which combine in an additive way.

Q: *"Are there different attributes that contribute to an overall utility?"*
Modeling: add one utility node for each attribute.

Finally, the decision and utility nodes must be linked into the graph structure. This involves considering the *causal* effects of decisions/actions, and the *temporal* aspects represented by information links and precedence links. The following questions probe these aspects.

Q: *"Which variables can decision/actions affect?"*
Modeling: add an arc from the action node to the chance node for that variable.

Q: *"Does the action/decision itself affect the utility?"*
Modeling: add an arc from the action node to the utility node.

Q: *"What are the outcome variables that there are preferences about?"*
Modeling: add arcs from those outcome nodes to the utility node.

Q: *"What information must be available before a particular decision can be made?"*
OR **Q:** *"Will the decision be contingent on particular information?"*
Modeling: add information arcs from those observation nodes to the decision node.

Q: *"(When there are multiple decisions) must decision D1 be taken before decision D2?"*
Modeling: add precedence arcs from decision node D1 to decision node D2.

Missing car example

Let's work through this for the missing car example from the previous section. Recall that the original description concluded with *"Should they [the Nguyens] be worried about their second car being stolen and notify the police?"* They have to *decide* whether to notify the police, so *NotifyPolice* becomes the decision node. In this problem, acting on this decision doesn't affect any of the state variables; it might lead to the Nyugens obtaining more information about their car, but it will not change whether it has been stolen or borrowed. So there is no arc from the decision node to any of the chance nodes. What about their preferences, which must be reflected in the connections to the utility node?

Additional information about Nguyen's preferences

If the Nguyen's car is stolen, they want to notify the police as soon as possible to increase the chance that it is found undamaged. However, being civic-minded citizens, they don't want to waste the time of the local police if Sam has borrowed it.

This preference information tells us that utility depends on both the decision node and the *CarStatus* node, shown in Figure 9.17 (using the version in Figure 9.16(c)). This means there are four combinations of situations for which preferences must be elicited. The problem description suggests the utility ordering, if not the exact numbers, shown in Table 9.4[‖]. The decision computed by this network as evidence is added sequentially, is shown in Table 9.5. We can see that finding the car missing isn't enough to make the Nguyen's call the police. Finding signs of forced entry changes their decision; however more information — finding out that Sam is out — once again changes their decision. With all possible information now available, they decide not to inform the police.

TABLE 9.4
Utility table for the missing car problem

CarStatus	*Inform Police*	Outcome utility	
		Qualitative	Quantitative
$borrowed$	no	neutral	0
$home$	no	neutral	0
$borrowed$	yes	poor	-5
$home$	yes	poor	-5
$stolen$	yes	bad	-40
$stolen$	no	terrible	-80

[‖] Note that the utilities incorporate unmodeled probabilistic reasoning about getting the car back.

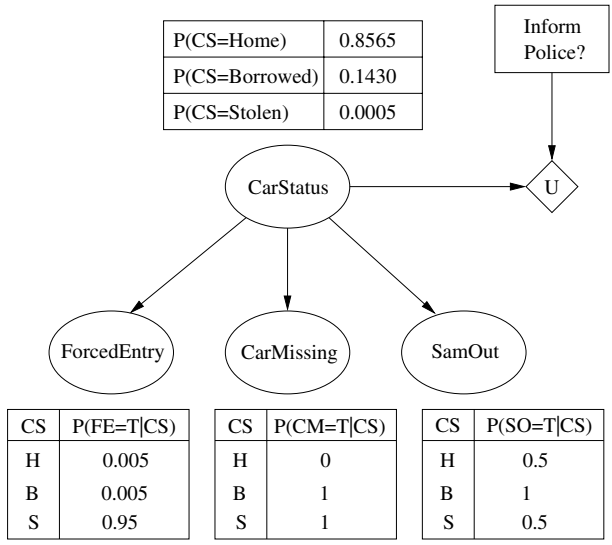

FIGURE 9.17
A decision network for the missing car problem.

TABLE 9.5
Expected utilities of *InformPolice* decision node for the missing car problem as evidence is added sequentially

Evidence	EU(*InformPolice*=Y)	EU(*InformPolice*=N)	Decision
None	-20.01	**-0.04**	No
CarMissing=T	-20.07	**-0.28**	No
ForcedEntry=T	**-27.98**	-31.93	Yes
SamOut=F	-24.99	**-19.95**	No

9.4 Adaptation

Adaptation in Bayesian networks means using some machine learning procedure to modify either the network's structure or its parameters over time, as new data arrives. It can be applied either to networks which have been elicited from human domain experts or to networks learned from some initial data. The motivation for applying adaptation is, of course, that there is some uncertainty about whether or not the Bayesian network is correct. The source of that uncertainty may be, for example, the expert's lack of confidence, or perhaps a lack of confidence in the modeled process itself being stable over time. In any case, part of the discipline of KEBN is the ongoing collection of statistics; so the opportunity of improving deployed networks by adaptation ought to be available.

We have already seen elements of adaptation. In Chapter 7 we saw how the Multi-

nomial Parameterization Algorithm 7.1 requires the specification of priors for the parameters, including an equivalent sample size which implicitly records a degree of confidence in those parameters. Also, §8.6.3 shows how prior probabilities for arc structure can be fed into the learning process. And, indeed, this section relies upon an understanding of those sections. The difference is one of intent and degree: in those chapters we were concerned with specifying some constraints on the learning of a causal model; here we are concerned with modifying a model already learned (or elicited).

9.4.1 Adapting parameters

We describe how to modify multinomial parameters given new complete joint observations. This can be done using the Spiegelhalter-Lauritzen Algorithm 7.1 under the same global and local assumptions about parameter independence. Recall that in that process the prior probabilities for each parameter are set with a particular equivalent sample size. If the parameters have actually been learned with some initial data set, then subsequent data can simply be applied using the same algorithm, starting with the new equivalent sample size and Dirichlet parameters set at whatever the initial training has left. If the parameters have been elicited, then you must somehow estimate your, or the expert's, degree of confidence in them. This is then expressed through the equivalent sample size, the larger the sample size the greater the confidence in the parameter estimates and the slower the change through adapting them with new data.

Suppose for a particular Dirichlet parameter α_i the expert is willing to say she gives a 90% chance to the corresponding probability $p_i = 0.2$ lying within the interval $[.1, .3]$. The expected value (i.e., the mean) of the Dirichlet distribution over state i corresponds to the estimated parameter value:

$$\mu_i = \frac{\alpha_i}{s} = 0.2$$

where $s = \sum_j \alpha_j$ is the equivalent sample size. The variance over state i of the Dirichlet distribution is:

$$\sigma_i^2 = \frac{\alpha_i(s - \alpha_i)}{s^2(s + 1)} = \frac{\mu_i(1 - \mu_i)}{s + 1}$$

The 90% confidence interval of this distribution lies within 1.645 standard deviations of the mean and is

$$\mu_i \pm \sigma_i = \mu_i \pm 1.645\sqrt{\frac{\mu_i(1 - \mu_i)}{s + 1}}$$

We can solve this algebraically for values of α_i and s that yield the mean and 90% confidence interval. A solution is an equivalent sample size $s = 15$ and a parameter value α_i of 3.

Of course, when this procedure is applied independently to the different Dirichlet parameters for a single parent instantiation the results may not be fully consistent. If

the expert reports the above interval for the first state of the binary child variable and $[0.4, 0.6]$ for the second, then the latter will lead to an equivalent sample size of $s = 24$ and $\alpha_2 = 12$. Since the equivalent sample size applies to all of the values of the child variable for a single instantiation, it cannot be both 15 and 24! The sample size must express a common degree of confidence across all of the parameter estimates for a single parent instantiation. So, the plausible approach is to compromise, for example, taking an average of the equivalent sample sizes, and then finding numbers as close to the estimated means for each state as possible. Suppose in this case, for example, we decide to compromise with an equivalent sample size of 20. Then the original probabilities for the two states, 0.2 and 0.5, yield $\alpha = (4, 10)$, which does not work. Normalizing (with round off) would yield instead $\alpha = (6, 14)$.

When parameters with confidence intervals are estimated in this fashion, and are not initially consistent, it is of course best to review the results with the expert(s) concerned.

Fractional updating is what Spiegelhalter and Lauritzen[263] call their technique for adapting parameters when the sample case is missing values, i.e., for incomplete data. The idea is simply to use the Bayesian network as it exists, applying the values observed in the sample case and performing Bayesian propagation to get posterior distributions over the unobserved cases. The observed values are used to update the Dirichlet distributions for those nodes; that is, a 1 is added to the relevant state parameter for the observed variable. The posteriors are used to proportionally update those variables which were unobserved; that is, $p' < 1$ is added to the state parameter corresponding to a value which takes the posterior p'. The procedure is complicated by the fact that a unique parent instantiation may not have been observed, when the proportional updating should be applied across all the possible parent instantiations, weighted by their posterior probabilities. This procedure unfortunately has the drawback of overweighting the equivalent sample size, resulting in an artificially high confidence in the probability estimates relative to new data.

Fading refers to using a time decay factor to underweight older data exponentially compared to more recent data. If we fade the contribution of the initial sample to determining parameters, then after sufficient time the parameters will reflect only what has been seen recently, allowing the adaptation process to track a changing process. A straightforward method for doing this involves a minor adjustment to the update process of Algorithm 7.1 [128, pp. 89-90]: when state i is observed, instead of simply adding 1 to the count for that state, moving from α_i to $\alpha_i + 1$, you first discount all of the counts by a multiplicative decay factor $d \in [0, 1]$. In other words the new Dirichlet distribution becomes $D[d\alpha_1, \ldots, d\alpha_i + 1, \ldots, d\alpha_\tau]$. In the limit, the Dirichlet parameters sum to $\frac{1}{1-d}$, which is called the **effective sample size**.

9.4.2 Structural adaptation

Conceivably, rather than just modifying parameters for an existing structure, as new information comes to light we might want to add, delete or reverse arcs as well. Jensen reports that "no handy method for incremental adaptation of structure has been constructed" [128]. He suggests the crude, but workable, approach of accumu-

lating cases and rerunning structure learning algorithms in batch mode periodically.

In addition to that, the Bayesian approach allows for structural adaptation, at least in principle. If we let what has previously been learned be reflected in our prior probabilities over structures, then new data will update our probability distributions over causal structure. That simply is Bayesian theory. To be sure, it is also Bayesian updating without model selection, since we need to maintain a probability distribution over all of the causal structures that we care to continue to entertain. A straightforward implementation of such inference will almost immediately be overwhelmed by computational intractability.

However, there is a simpler technique available with CaMML for structural adaptation. Since CaMML reports its estimate of the probability of each directed arc at the end of its sampling phase, and since CaMML allows the specification of a prior probability for such arcs before initiating causal discovery, those probabilities can be reused as the prior arc probabilities when performing learning with new data. This is crude in that there is no concept of equivalent sample size operating here; so there is no way to give greater weight to the initial data set over the latter, or vice versa. Nevertheless, it can be an effective way of adapting previously acquired structure.

9.5 Summary

There is an interplay between elements of the KEBN process: variable choice, graph structure and parameters. Both prototyping and the spiral process model support this interplay. Various BN structures are available to compactly and accurately represent certain types of domain features. While no standard knowledge engineering process exists as yet, we have sketched a framework and described in more detail methods to support at least some of the KE tasks. The integration of expert elicitation and automated methods, in particular, is still in early stages. There are few existing tools for supporting the KEBN process, although some are being developed in the research community, including our own development of VE and Matilda.

9.6 Bibliographic notes

An excellent start on making sense of software engineering is Brooks' *Mythical Man-Month* [33]. Sommerville's text is also worth a read [261]. You may also wish to consult a reference on software quality assurance in particular, which includes detailed advice on different kinds of testing [93].

Laskey and Mahoney also adapt the ideas of prototyping and the spiral model to the application of Bayesian networks [166]. Bruce Marcot [182] has described a

KEBN process for the application area of species-environment relations.

One introduction to parametric distributions is that of Balakrishnan and Nevzorov [14]. Mosteller et al.'s *Statistics by Example* [196] is an interesting and readily accessible review of common applications of different probability distributions. An excellent review of issues in elicitation can be found in Morgan and Henrion's *Uncertainty* [194], including the psychology of probability judgments.

9.7 Problems

Problem 1

The elicitation method of partitioning to deal with local structure is said in the text to directly correspond to the use of classification trees and graphs in Chapter 7. Illustrate this for the *Flu*, *Measles*, *Age*, *Fever* example by filling in the partitioned CPT with hypothetical numbers. Then build the corresponding classification tree. What do partition elements correspond to in the classification tree?

Problem 2

Consider the BN that you constructed for your own domain in Problem 5, Chapter 2. (Or if you haven't completed this problem, use an example network from elsewhere in this text, or one that comes with a BN software package.)

1. Identify the type of each node: target/query, evidence/observation, context, controllable.
2. Re-assess your choice of nodes and values, particularly if you discretized a continuous variable, in the light of the discussion in §9.3.1.
3. Label each arc with the relationship that is being modeled, using the relationships described in §9.3.2 or others you might think of.
4. Use Matilda to investigate the dependence and independence relationships in your network.
5. Can you map the probabilities in the CPTs into a qualitative scale, as in §9.3.3.2?

Problem 3

Using some of methods from this chapter, reengineer one of the BN applications you developed for one of the problems from Chapter 5. That is, redo your application, but employ techniques from this chapter. Write up what you have done, with reference to specific techniques used. Also, describe the difference use of these techniques has made in the final BN.