chapter two

Computer science tools and techniques

Martha E. Pollack
University of Michigan, Ann Arbor, Michigan
Bart Peintner
Artificial Intelligence Center, SRI International, Menlo Park, California

Contents

2.1 Introduction 22
2.2 Pervasive computing 23
2.2.1 Principles of pervasive computing 24
2.2.1.1 Decentralization 24
2.2.1.2 Diversification 24
2.2.1.3 Connectivity 24
2.2.1.4 Simplicity 24
2.2.2 Elements of pervasive computing 25
2.2.2.1 Devices 25
2.2.2.2 Standards and protocols 26
2.2.2.3 Application services 28
2.2.3 Applications to healthcare 28
2.2.3.1 Reduced cost of current tasks 28
2.2.3.2 Increased quality of care 28
2.2.3.3 Peace of mind for caregivers 28
2.2.3.4 Assistive technology 29
2.2.3.5 Reduced risk for common activities 29
2.2.3.6 Unimagined applications 29
2.3 Intelligent applications 29
2.3.1 Representing knowledge 29
2.3.2 Probabilistic reasoning 30

© 2007 by Taylor & Francis Group, LLC

2.3.2.1 Bayesian networks 31
2.3.2.2 Dynamic models 33
2.3.2.3 Inference in dynamic models 34
2.3.3 Machine learning 34
2.3.3.1 Supervised learning 35
2.3.3.2 Unsupervised learning 35
2.3.3.3 Reinforcement learning 36
2.3.4 Automated planning 37
2.4 Privacy and security 39
2.4.1 Privacy 40
2.4.2 Authentication 41
2.4.3 Authorization 41
2.4.4 Integrity 42
2.4.5 Caveat user 42
2.5 Summary 42
2.6 To learn more 43
References 43

2.1 Introduction

This chapter provides an overview of *pervasive computing* techniques that are being used to develop the advanced healthcare systems described in the remainder of the book. Pervasive computing—sometimes also known as ubiquitous computing—can be seen as the third wave of computing paradigms. In the first wave, which started in the 1960s and held sway through the 1980s, computers were large mainframes shared by multiple users. The second wave centered on personal computers with individual users. The third wave of computing, still emerging now at the start of the twenty-first century, is one in which each user has continual access to a large number of networked computers. However, these are not the traditional desktop-style computers as we have come to understand them but instead may include devices such as mobile phones, personal digital assistants (PDAs), and various kinds of sensors. Users do not interact with these computers using a keyboard, mouse, and operating-systems commands; instead interaction is much simpler, and sometimes even automatic, with the pervasive computing system sensing what is needed and acting accordingly. Of course, traditional computers can also play a role in a pervasive computing environment, when they are connected with devices that allow them to support "everywhere, anytime" access to data.

Thus, in the vision of ubiquitous computing, computer-based systems anticipate and respond to the needs of their users to control everyday environments such as homes, workplaces, and automobiles. For example, in a home environment, a pervasive computing system could be used to support a variety of tasks, including turning lights on and off as people move from one room to another, automatically generating shopping lists as items are

© 2007 by Taylor & Francis Group, LLC

removed from the refrigerator, and learning a person's favorite television programs and recording them automatically. Within the area of healthcare, pervasive computing techniques could lead to systems in which devices like thermometers and sphygnometers immediately transmit information to a patient's electronic medical records. They could provide a physician in the hospital with instant access on a secure PDA to the up-to-date records of a current patient. Pervasive computing could make it possible for a cognitively impaired person to have his activities monitored and to receive a reminder if he forgot to take his medicine or needed directions to get back home should he get lost walking around his neighborhood. Pervasive computing techniques could also make it possible for a physically disabled person to readily control her environment, with doors opening automatically as her wheelchair approached and the oven responding to commands spoken into a small, wearable microphone.

This chapter will discuss the kinds of computational techniques that make systems like the ones described above possible. We have divided the chapter into three main sections:

- **Pervasive Computing:** We begin by describing the overall pervasive computing enterprise in more detail, discussing the kinds of devices and protocols used.
- **Intelligent Applications:** Next we describe techniques, primarily from the field of artificial intelligence (AI), that are used to make pervasive healthcare applications behave "intelligently."
- **Privacy and Security:** Because of the nature of the data they are using and the services they are providing, it is essential that pervasive computing systems ensure privacy and security. In this section, we describe how privacy and security are ensured.

It is impossible to provide a detailed tutorial on each of these topics. Instead, the goal of this chapter is to ensure that the remainder of this book is accessible to a wide range of readers, who may have different amounts of background in computer science. We highlight key computational challenges in the design of pervasive computing systems and we sketch the types of approaches used to meet them, providing references for readers who want to follow up on particular topics more extensively.

2.2 *Pervasive computing*

As suggested by the examples in the introduction, pervasive computing is a broad effort to make information-centric tasks simple, mobile, and secure. The effort aims both to uncover new applications and to improve the efficiency of current computing applications by making them available in more places, more often, and with more convenience for users.

© 2007 by Taylor & Francis Group, LLC

One of the most well-known embodiments of pervasive computing is the mobile phone. The mobile phone in its simplest form allows two people to communicate at a distance, the same basic service that the standard phone has provided for decades. The mobile phone has made this service available in more places and—because we now rarely leave home without it—more often. The mobile phone is also an example of a new application made possible by pervasive computing. Current state-of-the-art mobile phones contain digital cameras, games, GPS services, and the ability to communicate in other ways, such as e-mail and text messaging. We do not think of our mobile phone as a computer, but rather just as a tool we use in our daily lives. In fact, this is one of the goals of pervasive computing: to seamlessly integrate information technology into our daily lives.

2.2.1 Principles of pervasive computing

The pervasive computing slogan "everywhere and anytime" summarizes its four basic principles: decentralization, diversification, connectivity, and simplicity.

2.2.1.1 Decentralization

In contrast to the earlier waves of computation where centralized computers performed a wide variety of tasks, the pervasive computing approach has many devices that perform more specialized tasks. Applications and services are provided by multiple networked devices that coordinate their resources.

2.2.1.2 Diversification

The pervasive computing paradigm employs a wide array of devices, all of which can interoperate and share information. Therefore, services are designed in such a way that current devices will be able to communicate with future devices. This focus aims to ensure that infrastructures created today do not limit future possibilities.

2.2.1.3 Connectivity

Interoperability requires communication protocols that must be agreed on by all parties that develop pervasive computing devices and services. For example, the communication protocols used by mobile phones are rigidly standardized to enable phone manufacturers to create phones that communicate and share services with the phones of other manufacturers. Computer languages and protocols that work on different types of hardware are needed to ensure that basic services do not have to be redeveloped for each new device.

2.2.1.4 Simplicity

Pervasive computing has a heavy focus on issues of human–computer interaction. The goal is to design systems for which there is almost no "learning

© 2007 by Taylor & Francis Group, LLC

curve" (i.e., systems that are so intuitive that people can use them as easily as they use their telephones and toasters). In part, the motivation for simplicity is the fact that small, mobile devices cannot support the same kinds of interactions as do computers with large screens, keyboards, and mice; however, simplicity also recognizes that mobile devices and applications must be easy to learn and use if they are to be more useful than the current way of doing things. Being simple is not the same as being primitive—pervasive computing systems can and do have complex functionality. Rather, simplicity focuses on creating effective, easy, and enjoyable methods of interacting with computer-based systems.

2.2.2 Elements of pervasive computing

The tangible results of pervasive computing can be categorized in three main areas: the devices that run pervasive computing applications, the protocols that allow devices and applications to communicate, and the applications and services pervasive computing provides.

2.2.2.1 Devices

We commonly think of pervasive computing devices as small portable devices, such as PDAs, mobile phones, pagers, and mobile digital music players. However, other, more stationary devices also play a part in seamlessly integrating technology into our lives. For example, home entertainment systems and game consoles are now able to connect to the Internet, providing access to digital content and a mechanism for playing games against remote players. Other devices provide behind-the-scenes support for pervasive applications, including network routers and modems, mobile phone towers, and wireless access points.

Many advanced applications, including a number of healthcare applications, rely heavily on sensors that can monitor aspects of an environment and on actuators that can take actions to physically change an environment. For example, patient monitoring is currently an active research topic. The goal is to recognize how well patients have slept, whether they have taken their medicine, whether they are eating and drinking, and so on. Various devices are being explored for these purposes. One approach makes use of a wide variety of environmental sensors, such as motion detectors, which sense a person's movement through the home; contact switches, which can indicate whether a cabinet, refrigerator door, or closet door has been opened; flush sensors on toilets, flow sensors on faucets, load sensors on beds, and so on. Networks of these sensors can be installed in a person's home or in an assisted living facility and can wirelessly transmit information that is useful in recognizing a person's activities.[1,2] An alternative approach is to use radio frequency identification (RFID) technology: here one attaches tiny RF sensors to household objects, and the user wears an RF reader (e.g., installed in a glove or on a piece of jewelry). When the person nears one of the tagged objects, the RF reader is triggered and wirelessly transmits information about

© 2007 by Taylor & Francis Group, LLC

the object.[3] When activities can be identified, then assistance can be provided (e.g., in the form of reminders about activities that need to be performed).[4,5] In an outdoor environment, global positioning systems (GPS) can be used to sense information about a person's location and to provide him with directions should he become disoriented and lost.[6] In addition, pervasive healthcare systems may use biosensors, which can range from a stationary device, such as a sphygnometer in the home that transmits a blood pressure reading to a doctor's office, to devices that are worn by a user to measure her temperature, heart rate, and so on, once again transmitting that information over a wireless network.[7] The information sensed in a pervasive computing environment can also trigger actuators that change the environment, such as stoves that automatically turn off if left on for too long without a person nearby, faucets that turn off when the water level is too high, or alarms that are activated when a person forgets to perform an important activity.

2.2.2.2 Standards and protocols

Standards and protocols are essential in allowing diverse devices to share information, and are required on many levels. At the hardware level, they are needed to specify which devices can "talk" and also to ensure that arbitrary devices "understand" each other. One level higher are network protocols, which define exactly how to address messages, package information, and ensure an entire message arrives at its destination. At the application level, interfaces must be defined to allow diverse applications to access the same functionality. Standards also define how certain aspects of security are handled, such as how to ensure a user or a device is properly identified, how to ensure a user or a device has permission to access a resource, and how to prevent others from "listening" to private messages. Security is discussed in more detail in Section 2.4 of this chapter.

Important current hardware-level standards include UPnP™ (universal plug and play) and the Tivoli® device management system. Using these protocols, device manufacturers can describe the kinds of information that their hardware produces and accepts—essentially, their I/O (input/output) patterns. By clearly defining how their devices communicate externally, designers enable their devices to be readily integrated into heterogeneous networks. Most computer users are familiar with the convenience of installing plug-and-play peripheral devices: after a new piece of hardware, such as a memory stick or an iPod, is plugged into their computer's USB port, the computer automatically identifies the device and downloads and installs drivers (programs that manage communication between the devices) for it. With plug-and-play, all the user has to do is simply plug in the new device and it is ready to use. This feature is possible because both the peripheral device and the computer make use of the same device management protocols and hence the peripheral is able to "tell" the computer about what it is, as well as about the name and location of the drivers that it needs. This type of essentially automatic network configuration is at the heart of the pervasive

© 2007 by Taylor & Francis Group, LLC

computing paradigm, where the goal is to make it possible for arbitrary mobile devices to be *seamlessly integrated* into existing networks whenever and wherever they are needed.

Network protocols have existed for quite a while. Possibly the best-known one is the TCP/IP (Transmission Control Protocol/Internet Protocol), which was developed in the mid- to late 1970s—practically ancient by computer-science standards! TCP/IP is built in to many operating systems and is also the protocol used by the Internet. TCP/IP is one way of ensuring that messages are correctly sent from one location to another in a network. It provides a way of specifying information about the network address to which the message is going, about the size of the message, and so on. It also includes a set of standard techniques for ensuring that the entire message is received at its intended destination and for message retransmission in case of error. Other protocols have been built "on top of" TCP/IP. An important example is Hypertext Transfer Protocol (HTTP), a communication protocol for transferring files of data on the World Wide Web. HTTP is a server–client-based protocol where one system called the client (e.g., a Web browser) makes a request to another system called the server (e.g., a Web server) for particular files, expressing the request according to HTTP protocol. The server replies, also using HTTP protocol to ensure the client can understand the response. By using the same protocol, systems that have completely different applications are able to successfully communicate with each other because they have rules regarding how to structure and interpret communications.

A number of other protocols have been developed in recent years to allow communication among cellular and other wireless devices. These include protocols for cellular telephone communication (there are a large variety of these, such as GSM, TDMA, CDMA, GPRS, and SMS); Bluetooth, a standard that supports communication devices that use a particular kind of small, inexpensive radio chip; Wireless Application Protocol (WAP), which can be thought of as a kind of HTTP for cellular devices; and X10, a protocol that allows home automation devices to communicate using 110V wiring.

Above we mentioned the importance of standardization at the programming and applications level. One of the main motivations for the development of the Java programming language in the early 1990s was to enable programs that were written and compiled on one machine to be run, with identical behavior, on any other machine. To achieve this goal, Java programs are not compiled directly into the machine language for any specific computer, but are rather compiled into a machine language called "Java bytecode." Java bytecode can be run on any computer that has a Java bytecode interpreter, and it will produce the same results, regardless of the machine the code is run on. Thus, although a Java bytecode interpreter must be constructed for each type of computer, once a computer has such an interpreter it can run any Java program that was compiled into Java bytecode on any machine. In a similar fashion, Hypertext Markup Language (HTML) was designed as a standard language for specifying Web pages in a way

© 2007 by Taylor & Francis Group, LLC

that (in principle) allows them to look the same on different machines and in different browsers. eXtensible Markup Language (XML) is a standard that outlines how to describe the data that is included on Web pages, enabling different devices to interpret it.

2.2.2.3 *Application services*

The purpose of developing devices and standards is to provide the necessary infrastructure for application services. Many basic services fall into the general category of communication including e-mail, telephones, text messaging, and video conferencing. Other services deal with managing information, such as remote database access and file transfer. Many services make secure transactions, such as streaming music, trading stock, or placing merchandise orders. A fundamental belief in pervasive computing is that new, previously undreamed-of services and applications will arise when devices, standards, and current services mature. In Section 2.2.3 we briefly mention some of the types of applications that are emerging from the use of pervasive computing in healthcare, and, of course, the remaining chapters of this book provide many more examples of such applications.

2.2.3 *Applications to healthcare*

How do advances in pervasive computing affect healthcare? Such advances are predicted to lead to profound changes in healthcare, including the following significant effects.

2.2.3.1 *Reduced cost of current tasks*

Reducing the heavy load of paperwork carried by medical professionals would free up time for these highly trained individuals to concentrate on their primary, caregiving tasks. Mobile devices, if secure and easy to use, could be used as methods for entering much patient data only once, automatically propagating relevant data to the many forms that must be completed. In addition to saving clinicians' time, digital information could be automatically processed, significantly reducing labor costs.

2.2.3.2 *Increased quality of care*

Pervasive devices could give doctors real-time access to medical records and available research. This could help doctors make more informed decisions about care. Automated reasoning about medical histories and proposed treatments could possibly reduce clinician errors.

2.2.3.3 *Peace of mind for caregivers*

Sensors and other mobile devices in a home can provide information about elderly or disabled individuals to their caregivers—for example, reducing a caregiver's worries about whether her loved one has fallen or has failed to carry out important activities.

© 2007 by Taylor & Francis Group, LLC

2.2.3.4 Assistive technology

The same sensors and devices used to give caregivers peace of mind can also be used to provide information to people recovering from an injury or living with a disability. For example, for someone with mild memory impairment, sensors could provide enough information to allow an application to issue reminders to users about important items, such as activities that they need to perform.

2.2.3.5 Reduced risk for common activities

Smart sensors and actuators can reduce the risk of injury for those in assisted living environments. Stoves or faucets that reason about when to shut off would reduce the risk of fire or other damage.

2.2.3.6 Unimagined applications

Of course, with new technologies come new ideas for applications. Just as we did not imagine all of the uses for mobile phones before they became part of our everyday lives, we cannot foresee all of the possibilities for healthcare applications and services.

2.3 Intelligent applications

The devices and protocols for pervasive computing make it possible to collect and transmit data to healthcare applications such as those outlined at the end of the previous section. The applications themselves incorporate a wide range of computational techniques, including many that come from research on artificial intelligence (AI). This section surveys those techniques, providing a "gentle" introduction to the interested reader. However, complete understanding of these concepts is not required for reading the chapters in the remainder of this book.

2.3.1 Representing knowledge

Pervasive computing applications tend to be knowledge intensive and make use of a great deal of information about the domains in which they operate. For example, a system that tracks the activities of a person would need to "know" a lot about those activities. It would have to know that making lunch typically occurs around noon, that it involves being in the kitchen and preparing certain kinds of foods, and that preparing food may involve things like opening cabinets and the refrigerator door. A system that helped a user get back home if he became lost would have to know about the normal travel habits of its user as well as understanding maps and bus routes. A system that automated much of the paperwork in a doctor's office would have to know about diagnosis codes and insurance-approval processes and billing procedures.

The field of AI has long been concerned with knowledge-intensive applications and has developed many techniques for representing and reasoning

© 2007 by Taylor & Francis Group, LLC

about knowledge. Researchers in the field speak about the design of *ontologies*. Ontologies are common vocabularies used for representing knowledge about some domain that also organizes the vocabularies in some specified way. For example, the early medical expert system MYCIN,[8] which could diagnose blood diseases, adopted a rule-based ontology where it modeled its knowledge in terms of rules that related concepts to other concepts, specifically symptoms to hypotheses, and hypotheses to other hypotheses or to diagnoses of disease. To reason with this type of ontology, MYCIN started with observed symptoms and applied "backward-chaining": first finding rules that matched the symptoms, applying those rules to derive intermediate hypotheses, then finding rules that matched the intermediate hypotheses and applying those, and so on. This style of backward-chaining is commonly used in many AI systems. An alternative style of ontology can be seen in INTERNIST,[9] an early medical AI system that performed diagnoses for problems that would typically be encountered in a general internal-medicine practice. INTERNIST used a frame-based ontology where each disease was associated with a "frame" that included all the information about prototypical presentations of that disease. The frame was structured into slots (e.g., "symptoms") and fillers (e.g., "rash"). Reasoning then occurred by having the system employ a pattern-matching algorithm to find the frame that most closely matched the observed symptoms.

Modern intelligent systems, including the kinds of pervasive healthcare systems that are the focus of this book, tend not to use either pure rule-based or frame-based ontologies. As we describe in more detail in the next section, this is because most of these systems require the ability to reason extensively about uncertain information, something that the rule-based and frame-based ontologies handled only in somewhat ad hoc ways. However, rich ontologies still play an important role in these modern systems and are integrated with probabilistic reasoning techniques. For example, the designers of the ILSA[1] system have developed an extensive ontology for use in systems that support the caregiving process (e.g., systems that provide reminders to cognitively impaired people or that issue alerts to the human caregivers of cognitively or physically impaired patients). This ontology consists of a hierarchically structured network of more than 1,000 terms that can describe many caregiving activities.

2.3.2 *Probabilistic reasoning*

Many interesting pervasive computing applications require a method to make decisions or recommendations based on incomplete or incorrect information. For example, a doctor may want to enter a list of symptoms into a portable device and have it search databases for rare conditions that match or partially match those symptoms. Or perhaps an application that monitors the location of a patient may need to locate her using sensor information that is insufficient or simply inaccurate, especially because most current sensor systems are "noisy." Applications like these require tools for reasoning about

© 2007 by Taylor & Francis Group, LLC

uncertainty—in other words, a way of converting incomplete or inaccurate information into "beliefs" about the situation being sensed or reasoned about. The field of probabilistic reasoning focuses on this problem and has provided tools for compactly representing and reasoning about uncertain knowledge and relationships.

At the core of probabilistic reasoning is the *probability distribution*. Given a set of possible situations (also called outcomes or states), a probability distribution defines the probability that each possible situation is the true situation. Consider a case in which a patient's symptoms indicate three possible conditions: a cold (denoted "C"), influenza ("I"), and food poisoning ("F"). One probability distribution could be {C = 0.1, I = 0.7, F = 0.2}. Given such a distribution, a doctor would strongly suspect that the patient has influenza, but further tests might be used to rule out food poisoning.

With only three outcomes, a probability distribution is easy to manage. However, distributions with millions of elements are very common and require a more structured representation. Imagine a distribution that defines a probability for every combination of symptoms and diseases. With only fifty conditions and twenty symptoms, the distribution would have more than fifty million elements.

2.3.2.1 *Bayesian networks*

The Bayesian network (BN) provides a compact, graphical way to represent large probability distributions. Its strength comes from its ability to represent *conditional independence*. For example, we might imagine that the probability that a person has a cough does not depend in any way on the probability that he has high blood pressure, and vice versa. If this is true, then the two conditions (cough and high blood pressure) are said to be conditionally independent of one another. The BN representation allows one to encode this conditional independence and thus reduce the number of elements that need to be included in the probability distribution: for example, we would not need to include a separate value for the probability of cough given high blood pressure, because that is the same as the probability of a cough without high blood pressure.

Figure 2.1(a) shows a BN that illustrates the problem mentioned above, in which there are fifty conditions and twenty systems. The BN factors the probability distribution over symptoms and conditions into twenty-one nodes: one node, labeled *Conditions*, represents all fifty conditions, while each of the other nodes each represent whether a particular symptom exists. In other words, the node labeled *Conditions* can be assigned any value that represent(s) the condition(s) a person might have. For example, it could be set to "cold" or "food poisoning" or even "cold and food poisoning." Each of the nodes labeled *Symptom* represents a particular symptom, so the node labeled *Symptom 2* might represent the fact that the person has a cough.

In this BN, the arrows represent cause and effect and thus point from condition to symptom, because the condition causes the symptoms. We can use the BN in Figure 2.1(a) to take a set of known symptoms in a given

© 2007 by Taylor & Francis Group, LLC

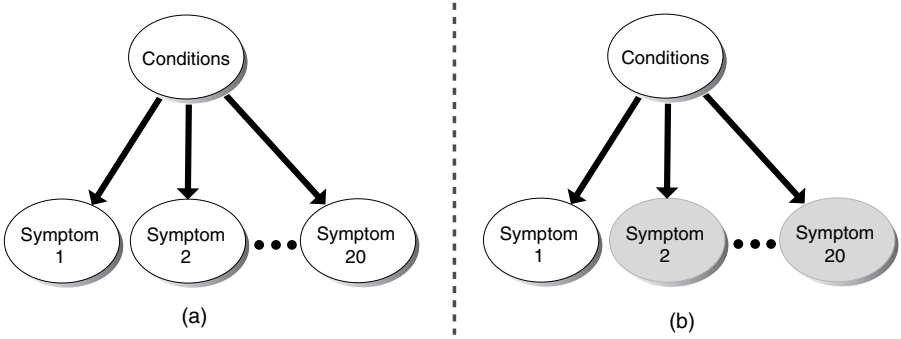

Figure 2.1 (a) A Bayesian network representing a probability distribution over conditions and symptoms. (b) A subset of the nodes set in evidence before inference.

patient and set them into evidence. Thus, suppose that a patient exhibits symptoms 2 and 20, as shown in Figure 2.1(b), where they are indicated by filled-in circles: if symptom 2 is "cough" and symptom 20 is "fever" then setting these in evidence represents the case where the patient has a cough and a fever and no other symptoms. Using the evidence that the patient has only symptoms 2 and 20, a probability distribution over all conditions can be calculated. It would tell us, in this example, that the probability that a person has a cold is relatively high, that the probability that he has a cold and influenza is relatively low, and that the probability that he has food poisoning is very low.

Calculating a probability distribution using evidence is known as *probabilistic inference*. Many different types of inference methods for Bayesian networks have been developed in recent decades. Some of these aim to calculate exact distributions, while others attempt to quickly calculate a distribution that approximates the true one.

While the example in Figure 2.1 illustrates the use of BNs for medical diagnosis problems, BNs are also frequently used for sensor interpretation. For example, consider the problem of activity recognition, which was mentioned in the previous section. A sensor network may detect that a person has entered her kitchen, opened a particular cabinet, used the electric can opener, turned on the faucet, and turned on the stove. What activity is she engaged in? If it is around noon, it may be fairly likely that she has opened a can of soup for lunch, but it is also possible that she is instead getting food and water for her cat. Just as medical conditions probabilistically give rise to symptoms, the performance of certain activities probabilistically gives rise to sensor firings. The connection is probabilistic, not only because people do not always do things the same way every time but also because the sensors themselves may be noisy, sometimes firing (or failing to fire) erroneously.

© 2007 by Taylor & Francis Group, LLC

2.3.2.2 Dynamic models

BNs only represent static situations: they assume the relationships and situations they model do not change over time. However, often the real situations we want to model do change over time. Imagine we want to know which room or hallway an elderly resident of an assisted living facility is in by using hallway sensors that detect when the patient is nearby. The hallway sensors alone may be insufficient to tell us which room the resident is in, but the sequence of hallway sensor readings could give us a much better idea.

To model a changing world such as this one, the hidden Markov model (HMM) maintains a probability distribution over all possible defined variables (in the example just given, this would be the rooms, hallways, and sensor readings) *for each point in time*. The resulting distribution is referred to as the *state*. At each point in time, an HMM uses the current state to estimate the likelihood of an *observation* occurring, such as the occupant's presence in a particular room, causing the observation of a particular sensor firing. Figure 2.2 shows a graphical representation of an HMM. The figure shows a series of identical *time slices* with two nodes each. The top node represents the state at a specific point in time and the bottom node represents the corresponding observation. The arrows model causality in the same manner as they did in the BN represented in Figure 2.1. For our resident location example, each time slice would represent the point in time when a sensor fired. It is the resident's being in a certain room (state) that causes the sensors to fire (observation) with some specified probability.

HMMs can incorporate and exploit information about impossible circumstances. For example, the current location of a person restricts the possibilities for her next location, as a person cannot instantly jump from one part of the building to another. If a person is in a room with a single door, the next location must either be within the same room or in the hallway outside of that door. Therefore, HMMs enable the construction of a much more accurate model than could be achieved by a simple BN, which, in the case of the tracked resident, would only be able to describe a static relationship between hallway sensors and location.

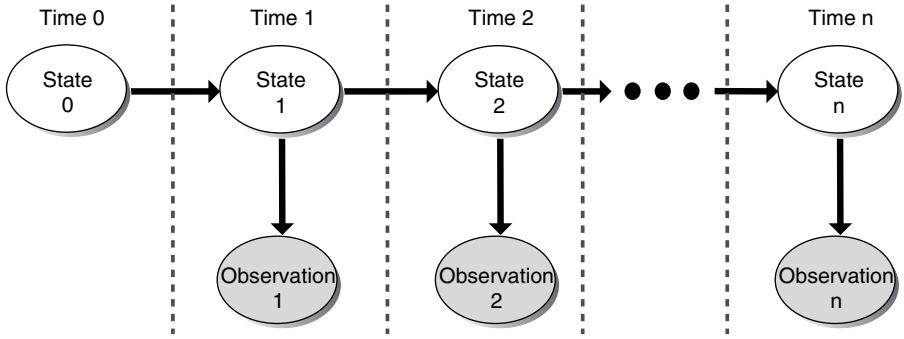

Figure 2.2 A generic hidden Markov model.

© 2007 by Taylor & Francis Group, LLC

Often, the size of the state or the size of the observation can be quite large. For example, imagine that we now want to track not only the location of the resident, but also her heart rate, walking speed, and the activity she is performing. Imagine that we also had several additional sensors that give us extra information about these additional aspects of the patient's state. Just as adding more symptoms to our probability distribution in the BN example caused the size of the distribution to sky-rocket, so does the size of the state and observation nodes explode when we try to monitor these additional elements.

A dynamic Bayesian network (DBN) extends an HMM in exactly the same way that BNs extend a probability distribution. DBNs allow the state and observation nodes in an HMM time slice to be broken into smaller pieces, much like the BN breaks a probability distribution into smaller pieces. The result is a more compact representation that requires more complex algorithms for inference.

2.3.2.3 *Inference in dynamic models*

The goal of inference in dynamic models such as HMMs and DBNs is to use a sequence of observations to infer a sequence of states. In other words, the observation nodes in each time slice are used as evidence and the probability distributions for the state nodes in each time slice are the desired result.

Many types of inference methods exist for HMMs and DBNs, both exact and approximate, but the concepts presented in this book can be understood without knowledge of how they work. However, we will mention one approximate method that is both intuitively appealing and often used. This method is called *particle filtering* and it is used to estimate the probability distribution for the sequence of state nodes that best explain the observations by using a set of "guesses." Each "guess" (called a particle) is a sequence of states. In our resident-monitoring example, a particle would be a sequence of rooms and hallways (with no probabilities) that indicate the resident's path. Each particle is given a weight based on how well it matches the recorded observations. By taking a weighted average of these particles, a distribution over each state can be obtained. The complexity of the algorithm lies in how each guess is obtained and how the set of guesses is maintained.

Probabilistic reasoning methods have been used in a number of very successful applications, both inside and out of the healthcare arena. For instance, Bayesian inference techniques have been widely used in the design of automated medical diagnosis systems for more than twenty years (see references 9, 10, and 11), while much of the recent work on activity recognition for healthcare intervention and delivery makes use of HMMs and DBNs (see references 2, 3, and 6).

2.3.3 *Machine learning*

The discussion above describes some ways we can represent uncertain knowledge and gives hints about how to reason about such knowledge. The techniques above all require a large number of probabilities to relate the

© 2007 by Taylor & Francis Group, LLC

information they are reasoning about, such as conditions to symptoms or actions to sensor firings. But where do the probabilities come from?

Probabilities can be estimated when there are many examples of each situation (or outcome) that is being modeled. For example, if we record the condition of thousands of patients that enter a hospital, we might easily be able to define a probability distribution over the most common conditions. However, for relatively rare conditions, thousands of patients may not be enough to create an accurate distribution. In addition, for many applications it is infeasible to obtain enough relevant examples, even if they exist in principle. Although we will not delve into the details of machine-learning techniques, we will briefly discuss the broad classes of techniques developed by the field: supervised learning, unsupervised learning, and reinforcement learning.

2.3.3.1 *Supervised learning*

One goal of machine learning is *classification*; it aims to sort input objects into one of several categories. For example, it may be desirable for a computer to learn to classify different skin diseases by using photographs. Classification problems are often solved by *supervised learning* techniques, which learn how to classify inputs using a set of example inputs and classifications. In other words, the algorithm is given "the answers" for an ideally large set of examples. A supervised learning technique attempts to build a model that maps known input to the correct classifications. Once this model is created, it can be used to classify other inputs for which the answers are unknown.

Supervised learning has been very successful and has been applied to a very wide range of problems, both inside and outside healthcare. As just a few examples, supervised learning techniques have been used to recognize credit card fraud,[12] to recognize potentially problematic traffic intersections,[13] and to learn the preferences of computer-system users.[14] In the medical domain, these techniques have successfully classified pregnant women who are at high risk for C-sections,[12] predicted recurrence of prostate cancer,[15] and learned to recognize potentially dangerous conditions in patients being monitored in intensive care.[16]

2.3.3.2 *Unsupervised learning*

As humans, we often have difficulty making sense out of mountains of data. For example, imagine recording the hallway sensor data in our resident-monitoring example for several months. A person looking at the raw data would most likely be overwhelmed by the sheer volume of data and would not be able to extract any useful information.

The goal of unsupervised learning is for a computer-based system to autonomously discover patterns in data. Unlike the supervised case, in which the answers for a training set of data are given in advance, the answers and even the forms of the answers are left for the unsupervised learning techniques to discover. Months of raw data for our resident's movements

© 2007 by Taylor & Francis Group, LLC

may reveal that the resident typically spends most of her time in one wing of the assisted living facility or that she is frequently up and about late at night. Such patterns could potentially be used to help improve the resident's quality of life, such as encouraging her to participate in social activities that take place in different parts of the facility or by correcting her medicine to help her sleep better at night.

Both supervised and unsupervised learning can be used to learn the parameters of an HMM or DBN, such as one that describes a resident's movements. In the supervised case, examples of room sequences and associated observations would allow the model's probabilities to be estimated quickly. If such room sequences do not exist, unsupervised techniques have been developed that can estimate the probabilities using only sequences of observations. The basic idea underlying these techniques is to identify natural clusters of data that may correspond to significant features of whatever is being learned. It has been found that by iteratively forming clusters (i.e., "best guess" grouping of data) and then using these clusters to reanalyze the data, one can converge upon good models that turn out to correlate well with phenomena of interest.

2.3.3.3 *Reinforcement learning*

Reinforcement learning techniques are useful when a system must choose an action to perform in a given situation. The goal here is to choose the action that leads to the best possible outcome, or *reward*. Imagine a system designed to notify a nurse when specific indicators for a patient change. The action of notifying the nurse should only be taken when the change is significant enough that it outweighs any inconvenience or irritation a false alarm may cause the nurse. If notifications are too frequent, the nurse will eventually pay less heed to them. If they are too rare, significant changes may go unnoticed. Rather than encoding precisely what is significant and what is not, reinforcement learning can be used to incrementally improve its evaluation of significance, based on whether the nurse gives it positive or negative feedback for notifications it makes or fails to make.

The reinforcement learning framework involves an "agent" (in our example, the system that learns when to issue a notification to the nurse) situated in an environment from which it may sometimes receive feedback. In the simplest formulation, the agent–environment interaction is formulated as a *Markov decision process*. In a Markov decision process there are discrete steps, and at each step the agent first senses its environment, chooses an action to perform (e.g., issue a notification or do nothing), and then receives a payoff based on the action the agent took and what the outcome of that action was. Note that payoffs may be delayed or omitted, and thus the agent has at best incomplete knowledge about the consequences of its actions. The agent has to use its interaction with the environment to learn a near-optimal *policy*. A policy is the strategy for deciding what action to take in each situation. Reinforcement learning algorithms have to address the fundamental challenge of temporal credit

© 2007 by Taylor & Francis Group, LLC

assignment (i.e., determining which subset of a long and complex sequence of actions was responsible for the good or bad long-term performance of the agent). Deciding which action to take can become difficult, especially in more complex reinforcement learning problems where the agent's sensors do not give it complete state information because of problems with noise and there is a variable, undeterminable amount of time between actions.

Until recently, reinforcement learning had been successfully applied only to a relatively narrow range of problems. Perhaps most notably, reinforcement learning was used in the design of a very successful computer backgammon game.[17] However, as algorithms and computing power improve, reinforcement learning techniques have begun to be used much more widely, including in certain pervasive healthcare applications. An example of this is a recent study that explored the feasibility of using reinforcement learning to create policies for deciding when to issue reminders to cognitively impaired people.[18]

2.3.4 *Automated planning*

We just described reinforcement learning, a technique for inferring correct policies for action through a process of (informed) trial and error. However, sometimes we want our systems to directly decide what actions they should perform by reasoning about actions whose outcomes systems already know, rather than by trying out alternatives to see what happens. For instance, a decision support system in a hospital might have knowledge about the various procedures that have been recommended for a patient and might want to schedule those procedures based on current information about the availability of different equipment and personnel. Usually it is not possible to schedule the procedures in any arbitrary order and certain procedures may need to precede other ones. Moreover, there may be contingent dependencies as well, with the need to schedule a specific procedure only if some other procedure has a particular outcome.

Automated planning techniques enable a system to construct plans of actions that achieve specified goals. In these systems, the computer typically has knowledge of a basic set of actions that can be performed, where each action is specified in terms of its *preconditions* (i.e., the thing or things that must be true before an action is done) and its *effects* (i.e., the thing or things that will become true if an action is performed). Given a specification of the current situation and of the goals to be achieved, both of which are expressed in the same language as the action preconditions and effects, the planning system then constructs a plan that is guaranteed to achieve the goal when performed in the initial state. Although this is quite simple when there are only a small number of possible actions and a small number of possible goals, planning can quickly become very time-consuming when the number of actions and goals grows, as in real-world situations. Additionally, the technique described above assumes that the computer's environment is

© 2007 by Taylor & Francis Group, LLC

completely observable—that is, it knows with certainty what the state of the world is and what the probable results of its actions are. This is not the case in the real world where noise and other sources of interference can cause input to be unreliable. The problems encountered when applying planning techniques are exacerbated when the planning system also has to reason about the temporal duration of actions or constraints on the times at which the goals are achieved, about actions whose effects are conditional on the circumstances in which they are performed, or about actions whose effects are uncertain. Significant research has gone into developing efficient algorithms for all these cases.

Many of these planning algorithms are based on the idea of *searching*. In a *state-space search*, the planning system first considers the initial state (i.e., the state of the environment at the time the plan will begin to be executed). It then reasons about what actions could be performed in that state by identifying all the possible actions whose preconditions are true in that initial state. The planning system reasons about what the effect will be of applying each possible action, decides which effects look most promising (using what is called an *evaluation heuristic*), and then repeats the process, essentially stringing together sequential actions into a plan. At any point in time, it can temporarily or permanently abandon the sequence it is currently considering and turn to another partial sequence. In another variation, a state-space search proceeds backward by starting with the goal state, considering what actions could lead to that state, and then reasoning about what preconditions must have been true for each of those actions to have been performed. The planning system will repeat this until the initial state is reached, essentially constructing the sequence of actions that form the plan by starting with the last action in the sequence and adding successively earlier actions.

A commonly used alternative to a state-space search is a *plan-space search.* Here, the constructed plans are not complete sequences but rather are only partially ordered with respect to one another (i.e., the plan includes constraints that each specify when some action must precede another). Other approaches to planning include the use of a structure called a *plan graph*, which approximates the effects of sequences of actions to facilitate more rapid plan construction. This technique translates the problem into one of logical inference where the system generates a proof that a sequence of actions will achieve the goal and then uses the sequence that was constructed as part of the proof to decide which actions to take.

When it is important for planners to reason about the times at which actions are performed or goals achieved, *constraint-satisfaction processing* is also used. A constraint-satisfaction problem (CSP) consists of a set of variables (V) and a set of domains (D), one for each variable, and a set of constraints on the ways in which the values from the domains can be legally assigned to the variables. An extremely simple example is shown in Figure 2.3. Here, V has two variables. b represents the time breakfast is finished and m represents the time frame in which medicine must be taken.

© 2007 by Taylor & Francis Group, LLC

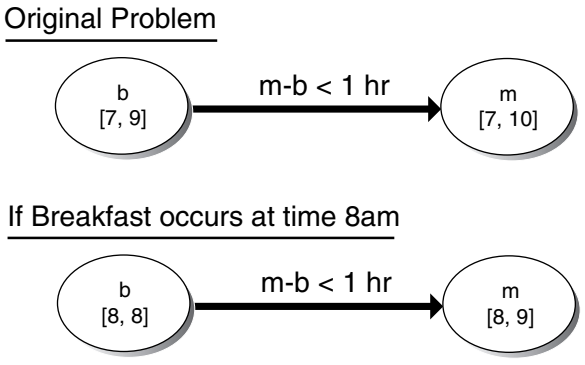

Figure 2.3 A very simple CSP and the result of propagating a constraint in the CSP.

The domain of *b* is [7, 9], representing the fact that breakfast can be eaten any time between 7 a.m. and 9 a.m. The only constraint is that medicine must be taken within one hour of eating, therefore $m - b < 1$. Consequently, the domain of m is [7, 10]. Should the system recognize that breakfast is finished at 8 a.m., then constraint propagation will cause the domain of *m* to shrink to the interval [8, 9] indicating that medicine must be taken by 9 a.m. In this case, we say that the legal solutions to the problem are those that assign a value between 8 and 9 to *m*. Again, this is a simple example and when there are many variables and many constraints, it is very difficult to find a legal solution to a CSP. Indeed, it can even be difficult to decide whether a valid solution exists. However, a great deal of research in computer science has produced algorithms that often perform well in practice on these types of problems.[19]

2.4 Privacy and security

The "everywhere, anytime" philosophy of pervasive computing is a double-edged sword: although it makes information available to users in more places and at more times, it also provides added opportunities for those who wish to steal or corrupt information. In fact, in many areas of information technology a trade-off exists between security and convenience. Typing in passwords or waiting for authorization to use resources is often an unwelcome hassle, but one we accept for a certain level of security. This trade-off is extremely important in pervasive healthcare applications. Busy healthcare professionals do not have time to constantly enter passwords or perform other types of validation when using technology. People who are ill or cognitively impaired may be unable to deal with onerous means of ensuring security. On the other hand, patients and their advocates will not accept the use of any information system that places their personal information at risk to those who could use the information against them.

Given the importance of security in healthcare applications, we cannot expect to simply "add security" once all the other problems are solved.

© 2007 by Taylor & Francis Group, LLC

Security must be a principle concern designed into devices and services from the beginning. The computer industry has learned tough lessons about the dangers of viewing security as an afterthought.

2.4.1 Privacy

When most people think of security, they think of the need for privacy: an assurance that a piece of information is not read by entities other than those who are intended to have access to it. The privacy of information is most vulnerable when it is traveling from one place to another. Information in pervasive computing applications usually travels through media that are shared by others: through the air, through fiber optic cables, or through other devices that direct the information to its destination. In any of these media, packets of information can be read by anyone with knowledge of the information infrastructure and the standards to which they adhere. Thus, in another example of the trade-off between convenience and security, the same standards that allow diverse devices to communicate also allow third parties to eavesdrop on that communication.

The general approach to providing many aspects of security is called *cryptography*. Cryptography is the process of converting a message into a code before transmission (encryption) and converting the code back into the message at the destination (decryption). If two parties have a secret "key" for encrypting a message on one end and decrypting the message on the other end, then the message will not be interpretable by a third party if intercepted. Thus, much of the research on privacy revolves around developing complex security keys that are difficult to "break." Other research focuses on methods for distributing keys and for enabling two parties to decide which key to use. If a third party intercepts the message that contains the key, then they will know how to decode messages that were encrypted with that key. The key itself could be encrypted, but with what key?

There are two main types of key strategies used in cryptography: symmetric and asymmetric. Symmetric strategies require a single key for both the sender and the receiver devices. The sender encrypts the message using the key and an encryption algorithm and the receiver decrypts it using the same key and algorithm. These approaches are usually very fast and are good for sending large amounts of data. The drawback is that the key somehow must be determined and distributed securely.

Asymmetric strategies use two keys: one for encryption called the *public key* and the other for decryption called the *private key*. The problem of determining and distributing keys is avoided in this scheme by allowing each device to have its own public and private keys, which are mathematically related in such a way that a message encoded using the public key can only be decoded using the matching private key. The public key is distributed freely, while the private key is only known by its owner. Using this system, if device X wants to send a message to device Y, X encrypts the message using Y's public key. When the message reaches Y, Y uses its private key to

© 2007 by Taylor & Francis Group, LLC

decrypt it. Because the message can only be decoded using an undistributed private key, third parties can only understand intercepted messages if they guess the private key.

Guessing the key sounds impossible, but because computers excel at quickly trying different combinations, the key must have many digits to make trying all possible combinations next to impossible. The longer the key, the harder it is to guess. As computers become faster and better at cracking codes, the keys must grow in length. If longer is more secure, you may ask why the keys are not made as long as possible. Asymmetric strategies are much slower than their symmetric counterparts and their speed depends on the length of the key. Therefore, a trade-off between speed and security must be made.

2.4.2 Authentication

Authentication is the process of proving an identity. Secure systems require this process to prevent third parties from accessing resources by using another's identity. Authentication is most commonly achieved through passwords. Systems assume that if you know your username and password, you are who you say you are. However, passwords, like keys, can be guessed, either by trying all possible passwords or by using knowledge about the person who chose the passwords. Passwords are often required to be at least some minimum length (e.g., eight characters) to make the first method more difficult. Nonalphabetic characters, like punctuation marks or numbers, are often required to prevent people from simply using their dog's name or other easily guessable passwords.

A password system can be strengthened by asking additional personal questions. However, this adds inconvenience. If the protected resource is very important, such as a bank account, then the inconvenience may be tolerated. For often-used resources like an e-mail account, the extra security may not be worth the trouble. Cryptography is (and should be) used in the authentication process as well. If passwords are not encrypted, then they are vulnerable to theft. Encryption also prevents those who administrate computing systems from viewing stored passwords. Biometric data provides an alternative to passwords. Physical characteristics of a person, including their fingerprints, voices, and eye patterns, cannot be guessed or easily faked.

2.4.3 Authorization

While privacy deals mostly with ensuring that communication is secure (i.e., that unknown people are not able to access information), authorization aims to guarantee that resources are only used by those authorized to use them. Authorization occurs after authentication: once a system determines who the user is, it can determine which resources the user can access.

When a system has many users and many resources, it becomes unwieldy to define which users have access to which resources. Therefore,

© 2007 by Taylor & Francis Group, LLC

authorization research focuses on allowing flexible specification of authorization policies. Under these policies, users and resources are categorized into groups, allowing an administrator to allow or deny entire user groups access to groups of resources.

2.4.4 Integrity

When one party sends a message to another, it is often important to verify that the message sent is the same one that is received. Even though a third party may not be able to understand a message (thanks to encryption), the third party may still be able to corrupt the message. Even without a malicious third party, messages may get lost or corrupted by the network infrastructure or by other programs on the devices simultaneously sending and receiving. Therefore, a secure communication channel must have built-in ways to ensure the integrity of a message.

Integrity, like privacy, may be ensured using a "keylike" system. Before a message is sent, the message is run through a *hash function*, which computes a single number based on the entire message. For example, a simple hash function might count the number of vowels in the message and multiply that by the number of time the letter "t" occurs. This number is tacked on to the end of the message before transmission. When the message is received, it is run through the same hash function. If the number calculated by the hash function matches the number at the end of the message, then the message is believed to be intact. Of course, some hash functions are better than others. The function described above is probably more likely to be a better identifier for a message than one that simply counts the number of characters before the first nonalphabetic character.

If a third party knows the hash function, it could modify the message and modify the hash number as well. Therefore, the hash function itself must be distributed securely before messages are transmitted.

2.4.5 Caveat user

A computer system is never completely secure. It is always possible for a third party to guess a key, a hash value, or a password. However, we can control the likelihood of security being broken by choosing a longer key, choosing a longer password, or adding more hurdles before access is granted. In general there is a trade-off between security and convenience. Where a system needs to be in this trade-off depends on many factors, including the cost of a security break and the cost of the added inconvenience.

2.5 Summary

The field of computation is changing rapidly. Since they were first invented, computers have inevitably become smaller, faster, and cheaper. However, we

© 2007 by Taylor & Francis Group, LLC

have now moved into an era in which they have also become pervasive. They are no longer devices that sit in machine rooms or even on desktops but are instead embedded throughout our environments, where they are connected to one another either through wires or, more frequently, wirelessly. When combined with the power of advanced algorithms for automated reasoning, planning, and learning, as well as techniques for ensuring privacy and security, these pervasive devices will lead to intelligent applications that will radically change the way healthcare is delivered.

2.6 To learn more

There is a large amount of technical literature available on all the topics described in this chapter. Several books provide good overviews on pervasive computing,[20] wireless sensor networks,[21] and network protocols.[22] There are also numerous books available with details about each of the different protocols—for example, on TCP/IP,[23] HTTP,[24] and Bluetooth.[25] A widely used textbook provides a comprehensive survey of AI techniques.[26] There are also a number of good texts on probabilistic reasoning,[27,28] machine learning,[29,30] automated planning,[31] and constraint satisfaction processing.[19] Discussion of computer security techniques can be found in several sources.[32,33]

References

1. Karen Zita Haigh, Liana M. Kiff, Janet Myers, Valerie Guralnik, Christopher W. Geib, John Phelps, and Tom Wagner, "The Independent LifeStyle Assistant™ (I.L.S.A.): AI Lessons Learned." In *The Sixteenth Innovative Applications of Artificial Intelligence Conference*, pp. 852–857, 2004.
2. Emmanuel M. Tapia, Stephen S. Intille and Ken Larson, "Activity Recognition in the Home Setting Using Simple and Ubiquitous Sensors," *Proceedings of PERVASIVE,* pp. 158–175, 2004.
3. Matthai Philipose, Kenneth P. Fishkin, Michael Perkowitz, Donald J. Patterson, Dirk Hahnel, Dieter Fox, and Henry Kautz, "Inferring ADLs from Interactions with Objects," *IEEE Pervasive Computing* 3:50–56, 2004.
4. Martha E. Pollack, Laura Brown, Dirk Colbry, Colleen E. McCarthy, Cheryl Orosz, Bart Peintner, Sailesh Ramakrishnan, and Iaonnis Tsamardinos, "Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment," *Robotics and Autonomous Systems,* 44:273–282, 2003.
5. Jennifer Borger, Pascal Poupart, Jesse Hoey, Craig Boutilier, Geoff Fernie, and Alex Mihailidis, "Decision-Theoretic Approach to Task Assistance for Persons with Dementia," *Proceedings of the 9th International Joint Conference on Artificial Intelligence,* pp. 1293–1299, 2005.
6. Donald J. Patterson, Lin Liao, Krzysztof Gajos, Michael Collier, Nik Livic, Katherine Olson, Shiaokai Wang, Dieter Fox, and Henry Kautz, "Opportunity Knocks: A System to Provide Cognitive Assistance with Transportation Services," *Sixth International Conference on Ubiquitous Computing*, Nottingham, England, 2004.

© 2007 by Taylor & Francis Group, LLC

7. Bodymedia, www.bodymedia.com.
8. Bruce G. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, 1984.
9. Randolph A. Miller, Harry E. Pople, Jr., and Jack D. Myers, "Internist-1: An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine," *New England Journal of Medicine*, 307(8):468–476, 1982.
10. Michael Schwe, B. Middleton, David E. Heckerman, Max Henrion, Eric J. Horvitz, and Gregory F. Cooper, "Probabilistic Diagnosis Using a Reformulation of the INTERNIST-1/QMR Knowledge Base I: Probabilistic Model and Inference Algorithms," *Methods of Information in Medicine*, 30:241–255, 1991.
11. David E. Heckerman, and B. N. Nathwani, "An Evaluation of the Diagnostic Accuracy of Pathfinder," *Computer and Biomedical Research*, 25:56–74, 1992.
12. Tom M. Mitchell, "Does Machine Learning Really Work?" *AI Magazine*, 18(3):11–20, 1997.
13. Saso Dzeroski, Nico Jacobs, M. Molina, and C. Moure, "ILP Experiments in Detecting Traffic Problems," *Proceedings of the 10th European Conference on Machine Learning*, pp. 61–66, 1998.
14. Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billisus, "Machine Learning for User Modeling," *User Modeling and User-Adapted Interaction*, 11:19–22, 2001.
15. Bla Zupan, Janez Demar, Michael W. Kattan, J. Robert Beck, and Ivan Bratko, "Machine Learning for Survival Analysis: A Case Study on Recurrence of Prostate Cancer," *Artificial Intelligence in Medicine*, 20(1):59–75, 2000
16. Katharina Morik, Michael Imboff, Peter Brockhausen, Thorsten Joachims, and Ursula Gather, "Knowledge Discovery and Knowledge Validation in Intensive Care," *Artificial Intelligence in Medicine*, 19(3):225–249, 2000.
17. Gerald Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, 38(3):58–68, 1995.
18. Matthew Rudary, Satinder Singh, and Martha E. Pollack, "Adaptive Cognitive Orthotics: Combining Reinforcement Learning and Constraint-Based Temporal Reasoning," *21st International Conference on Machine Learning*, July 2004.
19. Rina Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.
20. Uwe Hansmann, Lothar Merk, Martin S. Nicklous, and Thomas Stober, *Pervasive Computing*, 2nd ed., Springer-Verlag, 2003.
21. Feng Zhao and Leonidis Guibas, *Wireless Sensor Networks*, Morgan Kaufmann, 2004.
22. James F. Kurose and Keith W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, 3rd ed., Addison-Wesley, 2004.
23. Behrouz A. Forouzan, *TCP/IP Protocol Suite*, 2nd ed., McGraw-Hill, 2003.
24. Brian Totty and David Gourley, *HTTP: The Definitive Guide*, O'Reilly, 2002.
25. Robert Morrow, *Bluetooth Operation and Use*, McGraw-Hill, 2002.
26. Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed., Prentice-Hall, 2002.
27. Judea Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.
28. Arnaud Doucet and Nando de Freitas, editors, *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, 2001.
29. Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

© 2007 by Taylor & Francis Group, LLC

30. Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.
31. Paolo Traverso, Malik Gallab, and Dana Nau, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.
32. Abraham Silberschatz and Peter Baer Glavin, *Operating Systems Concepts*, 5th ed., John Wiley and Sons, 1999.
33. Charles P. Pfleeger and Shari Lawrence Pfleeger, *Security in Computing*, 3rd ed., Prentice-Hall, 2002.

© 2007 by Taylor & Francis Group, LLC