The algorithms we have described so far require updating the utility or policy for all states at once. It turns out that this is not strictly necessary. In fact, on each iteration, we can pick *any subset* of states and apply *either* kind of updating (policy improvement or simplified value iteration) to that subset. This very general algorithm is called **asynchronous policy iteration**. Given certain conditions on the initial policy and initial utility function, asynchronous policy iteration is guaranteed to converge to an optimal policy. The freedom to choose any states to work on means that we can design much more efficient heuristic algorithms—for example, algorithms that concentrate on updating the values of states that are likely to be reached by a good policy. This makes a lot of sense in real life: if one has no intention of throwing oneself off a cliff, one should not spend time worrying about the exact value of the resulting states.

ASYNCHRONOUS POLICY ITERATION

17.4 PARTIALLY OBSERVABLE MDPS

The description of Markov decision processes in Section 17.1 assumed that the environment was **fully observable**. With this assumption, the agent always knows which state it is in. This, combined with the Markov assumption for the transition model, means that the optimal policy depends only on the current state. When the environment is only **partially observable**, the situation is, one might say, much less clear. The agent does not necessarily know which state it is in, so it cannot execute the action $\pi(s)$ recommended for that state. Furthermore, the utility of a state s and the optimal action in s depend not just on s, but also on *how much the agent knows* when it is in s. For these reasons, **partially observable MDPs** (or POMDPs—pronounced "pom-dee-pees") are usually viewed as much more difficult than ordinary MDPs. We cannot avoid POMDPs, however, because the real world is one.

PARTIALLY OBSERVABLE MDP

17.4.1 Definition of POMDPs

To get a handle on POMDPs, we must first define them properly. A POMDP has the same elements as an MDP—the transition model $P(s' \mid s, a)$, actions $A(s)$, and reward function $R(s)$—but, like the partially observable search problems of Section 4.4, it also has a **sensor model** $P(e \mid s)$. Here, as in Chapter 15, the sensor model specifies the probability of perceiving evidence e in state s.[3] For example, we can convert the 4×3 world of Figure 17.1 into a POMDP by adding a noisy or partial sensor instead of assuming that the agent knows its location exactly. Such a sensor might measure the *number of adjacent walls*, which happens to be 2 in all the nonterminal squares except for those in the third column, where the value is 1; a noisy version might give the wrong value with probability 0.1.

In Chapters 4 and 11, we studied nondeterministic and partially observable planning problems and identified the **belief state**—the set of actual states the agent might be in—as a key concept for describing and calculating solutions. In POMDPs, the belief state b becomes a *probability distribution* over all possible states, just as in Chapter 15. For example, the initial

[3] As with the reward function for MDPs, the sensor model can also depend on the action and outcome state, but again this change is not fundamental.

belief state for the 4×3 POMDP could be the uniform distribution over the nine nonterminal states, i.e., $\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \rangle$. We write $b(s)$ for the probability assigned to the actual state s by belief state b. The agent can calculate its current belief state as the conditional probability distribution over the actual states given the sequence of percepts and actions so far. This is essentially the **filtering** task described in Chapter 15. The basic recursive filtering equation (15.5 on page 572) shows how to calculate the new belief state from the previous belief state and the new evidence. For POMDPs, we also have an action to consider, but the result is essentially the same. If $b(s)$ was the previous belief state, and the agent does action a and then perceives evidence e, then the new belief state is given by

$$b'(s') = \alpha\, P(e \mid s') \sum_s P(s' \mid s, a) b(s) \;,$$

where α is a normalizing constant that makes the belief state sum to 1. By analogy with the update operator for filtering (page 572), we can write this as

$$b' = \text{FORWARD}(b, a, e) \;. \tag{17.11}$$

In the 4×3 POMDP, suppose the agent moves *Left* and its sensor reports 1 adjacent wall; then it's quite likely (although not guaranteed, because both the motion and the sensor are noisy) that the agent is now in (3,1). Exercise 17.13 asks you to calculate the exact probability values for the new belief state.

The fundamental insight required to understand POMDPs is this: *the optimal action depends only on the agent's current belief state.* That is, the optimal policy can be described by a mapping $\pi^*(b)$ from belief states to actions. It does *not* depend on the *actual* state the agent is in. This is a good thing, because the agent does not know its actual state; all it knows is the belief state. Hence, the decision cycle of a POMDP agent can be broken down into the following three steps:

1. Given the current belief state b, execute the action $a = \pi^*(b)$.
2. Receive percept e.
3. Set the current belief state to $\text{FORWARD}(b, a, e)$ and repeat.

Now we can think of POMDPs as requiring a search in belief-state space, just like the methods for sensorless and contingency problems in Chapter 4. The main difference is that the POMDP belief-state space is *continuous*, because a POMDP belief state is a probability distribution. For example, a belief state for the 4×3 world is a point in an 11-dimensional continuous space. An action changes the belief state, not just the physical state. Hence, the action is evaluated at least in part according to the information the agent acquires as a result. POMDPs therefore include the value of information (Section 16.6) as one component of the decision problem.

Let's look more carefully at the outcome of actions. In particular, let's calculate the probability that an agent in belief state b reaches belief state b' after executing action a. Now, if we knew the action *and the subsequent percept*, then Equation (17.11) would provide a *deterministic* update to the belief state: $b' = \text{FORWARD}(b, a, e)$. Of course, the subsequent percept is not yet known, so the agent might arrive in one of several possible belief states b', depending on the percept that is received. The probability of perceiving e, given that a was

performed starting in belief state b, is given by summing over all the actual states s' that the agent might reach:

$$\begin{aligned} P(e|a,b) &= \sum_{s'} P(e|a,s',b)P(s'|a,b) \\ &= \sum_{s'} P(e \mid s')P(s'|a,b) \\ &= \sum_{s'} P(e \mid s') \sum_{s} P(s' \mid s,a)b(s) \ . \end{aligned}$$

Let us write the probability of reaching b' from b, given action a, as $P(b' \mid b,a))$. Then that gives us

$$\begin{aligned} P(b' \mid b,a) &= P(b'|a,b) = \sum_{e} P(b'|e,a,b)P(e|a,b) \\ &= \sum_{e} P(b'|e,a,b) \sum_{s'} P(e \mid s') \sum_{s} P(s' \mid s,a)b(s) \ , \end{aligned} \tag{17.12}$$

where $P(b'|e,a,b)$ is 1 if $b' = \text{FORWARD}(b,a,e)$ and 0 otherwise.

Equation (17.12) can be viewed as defining a transition model for the belief-state space. We can also define a reward function for belief states (i.e., the expected reward for the actual states the agent might be in):

$$\rho(b) = \sum_{s} b(s)R(s) \ .$$

Together, $P(b' \mid b,a)$ and $\rho(b)$ define an *observable* MDP on the space of belief states. Furthermore, it can be shown that an optimal policy for this MDP, $\pi^*(b)$, is also an optimal policy for the original POMDP. In other words, *solving a POMDP on a physical state space can be reduced to solving an MDP on the corresponding belief-state space.* This fact is perhaps less surprising if we remember that the belief state is always observable to the agent, by definition.

Notice that, although we have reduced POMDPs to MDPs, the MDP we obtain has a continuous (and usually high-dimensional) state space. None of the MDP algorithms described in Sections 17.2 and 17.3 applies directly to such MDPs. The next two subsections describe a value iteration algorithm designed specifically for POMDPs and an online decision-making algorithm, similar to those developed for games in Chapter 5.

17.4.2 Value iteration for POMDPs

Section 17.2 described a value iteration algorithm that computed one utility value for each state. With infinitely many belief states, we need to be more creative. Consider an optimal policy π^* and its application in a specific belief state b: the policy generates an action, then, for each subsequent percept, the belief state is updated and a new action is generated, and so on. For this specific b, therefore, the policy is exactly equivalent to a **conditional plan**, as defined in Chapter 4 for nondeterministic and partially observable problems. Instead of thinking about policies, let us think about conditional plans and how the expected utility of executing a fixed conditional plan varies with the initial belief state. We make two observations:

1. Let the utility of executing a *fixed* conditional plan p starting in physical state s be $\alpha_p(s)$. Then the expected utility of executing p in belief state b is just $\sum_s b(s)\alpha_p(s)$, or $b \cdot \alpha_p$ if we think of them both as vectors. Hence, the expected utility of a fixed conditional plan varies *linearly* with b; that is, it corresponds to a hyperplane in belief space.
2. At any given belief state b, the optimal policy will choose to execute the conditional plan with highest expected utility; and the expected utility of b under the optimal policy is just the utility of that conditional plan:

$$U(b) = U^{\pi^*}(b) = \max_p b \cdot \alpha_p \ .$$

   If the optimal policy π^* chooses to execute p starting at b, then it is reasonable to expect that it might choose to execute p in belief states that are very close to b; in fact, if we bound the depth of the conditional plans, then there are only finitely many such plans and the continuous space of belief states will generally be divided into *regions*, each corresponding to a particular conditional plan that is optimal in that region.

From these two observations, we see that the utility function $U(b)$ on belief states, being the maximum of a collection of hyperplanes, will be *piecewise linear* and *convex*.

To illustrate this, we use a simple two-state world. The states are labeled 0 and 1, with $R(0) = 0$ and $R(1) = 1$. There are two actions: *Stay* stays put with probability 0.9 and *Go* switches to the other state with probability 0.9. For now we will assume the discount factor $\gamma = 1$. The sensor reports the correct state with probability 0.6. Obviously, the agent should *Stay* when it thinks it's in state 1 and *Go* when it thinks it's in state 0.

The advantage of a two-state world is that the belief space can be viewed as one-dimensional, because the two probabilities must sum to 1. In Figure 17.8(a), the x-axis represents the belief state, defined by $b(1)$, the probability of being in state 1. Now let us consider the one-step plans $[Stay]$ and $[Go]$, each of which receives the reward for the current state followed by the (discounted) reward for the state reached after the action:

$$\begin{aligned}
\alpha_{[Stay]}(0) &= R(0) + \gamma(0.9R(0) + 0.1R(1)) = 0.1 \\
\alpha_{[Stay]}(1) &= R(1) + \gamma(0.9R(1) + 0.1R(0)) = 1.9 \\
\alpha_{[Go]}(0) &= R(0) + \gamma(0.9R(1) + 0.1R(0)) = 0.9 \\
\alpha_{[Go]}(1) &= R(1) + \gamma(0.9R(0) + 0.1R(1)) = 1.1
\end{aligned}$$

The hyperplanes (lines, in this case) for $b \cdot \alpha_{[Stay]}$ and $b \cdot \alpha_{[Go]}$ are shown in Figure 17.8(a) and their maximum is shown in bold. The bold line therefore represents the utility function for the finite-horizon problem that allows just one action, and in each "piece" of the piecewise linear utility function the optimal action is the first action of the corresponding conditional plan. In this case, the optimal one-step policy is to *Stay* when $b(1) > 0.5$ and *Go* otherwise.

Once we have utilities $\alpha_p(s)$ for all the conditional plans p of depth 1 in each physical state s, we can compute the utilities for conditional plans of depth 2 by considering each possible first action, each possible subsequent percept, and then each way of choosing a depth-1 plan to execute for each percept:

$$[Stay;\ \textbf{if}\ Percept = 0\ \textbf{then}\ Stay\ \textbf{else}\ Stay]$$
$$[Stay;\ \textbf{if}\ Percept = 0\ \textbf{then}\ Stay\ \textbf{else}\ Go] \ldots$$

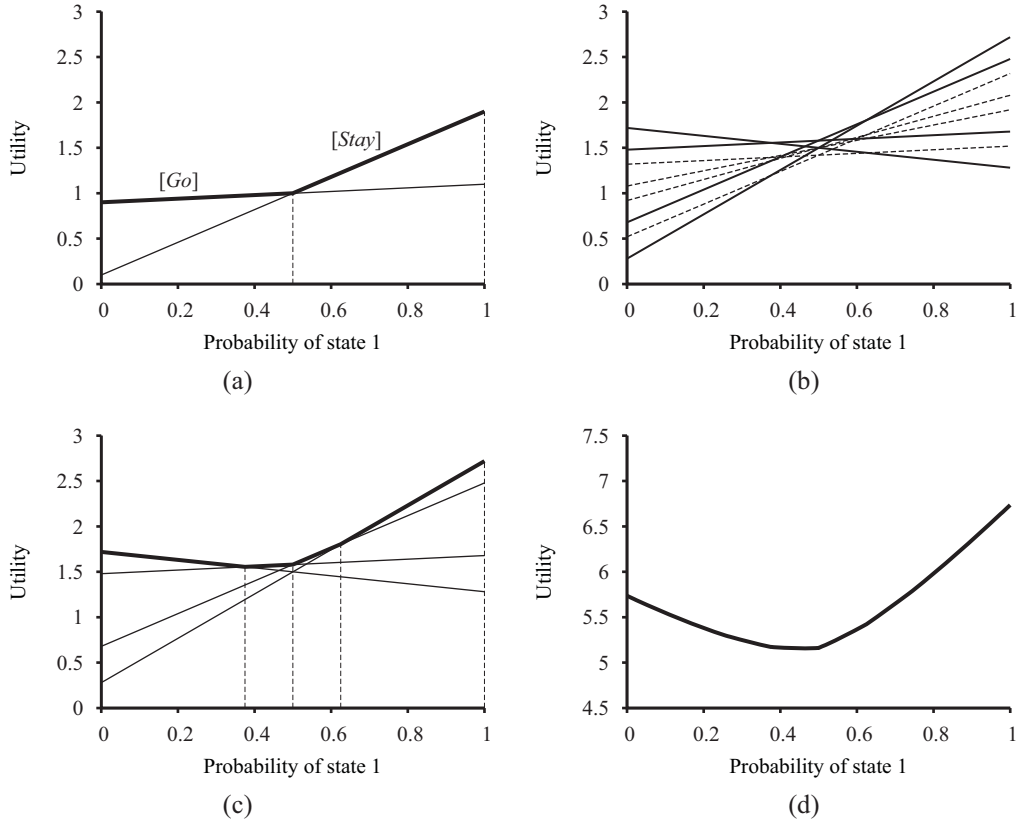

Figure 17.8 (a) Utility of two one-step plans as a function of the initial belief state $b(1)$ for the two-state world, with the corresponding utility function shown in bold. (b) Utilities for 8 distinct two-step plans. (c) Utilities for four undominated two-step plans. (d) Utility function for optimal eight-step plans.

There are eight distinct depth-2 plans in all, and their utilities are shown in Figure 17.8(b). Notice that four of the plans, shown as dashed lines, are suboptimal across the entire belief space—we say these plans are **dominated**, and they need not be considered further. There are four undominated plans, each of which is optimal in a specific region, as shown in Figure 17.8(c). The regions partition the belief-state space.

DOMINATED PLAN

We repeat the process for depth 3, and so on. In general, let p be a depth-d conditional plan whose initial action is a and whose depth-$d-1$ subplan for percept e is $p.e$; then

$$\alpha_p(s) = R(s) + \gamma \left(\sum_{s'} P(s' \mid s, a) \sum_{e} P(e \mid s') \alpha_{p.e}(s') \right) . \tag{17.13}$$

This recursion naturally gives us a value iteration algorithm, which is sketched in Figure 17.9. The structure of the algorithm and its error analysis are similar to those of the basic value iteration algorithm in Figure 17.4 on page 653; the main difference is that instead of computing one utility number for each state, POMDP-VALUE-ITERATION maintains a collection of

function POMDP-VALUE-ITERATION($pomdp, \epsilon$) **returns** a utility function
  inputs: $pomdp$, a POMDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    sensor model $P(e \mid s)$, rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', sets of plans p with associated utility vectors α_p

  $U' \leftarrow$ a set containing just the empty plan $[\,]$, with $\alpha_{[\,]}(s) = R(s)$
  repeat
    $U \leftarrow U'$
    $U' \leftarrow$ the set of all plans consisting of an action and, for each possible next percept,
      a plan in U with utility vectors computed according to Equation (17.13)
    $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
  until MAX-DIFFERENCE(U, U') $< \epsilon(1-\gamma)/\gamma$
  return U

Figure 17.9 A high-level sketch of the value iteration algorithm for POMDPs. The REMOVE-DOMINATED-PLANS step and MAX-DIFFERENCE test are typically implemented as linear programs.

undominated plans with their utility hyperplanes. The algorithm's complexity depends primarily on how many plans get generated. Given $|A|$ actions and $|E|$ possible observations, it is easy to show that there are $|A|^{O(|E|^{d-1})}$ distinct depth-d plans. Even for the lowly two-state world with $d = 8$, the exact number is 2^{255}. The elimination of dominated plans is essential for reducing this doubly exponential growth: the number of undominated plans with $d = 8$ is just 144. The utility function for these 144 plans is shown in Figure 17.8(d).

Notice that even though state 0 has lower utility than state 1, the intermediate belief states have even lower utility because the agent lacks the information needed to choose a good action. This is why information has value in the sense defined in Section 16.6 and optimal policies in POMDPs often include information-gathering actions.

Given such a utility function, an executable policy can be extracted by looking at which hyperplane is optimal at any given belief state b and executing the first action of the corresponding plan. In Figure 17.8(d), the corresponding optimal policy is still the same as for depth-1 plans: *Stay* when $b(1) > 0.5$ and *Go* otherwise.

In practice, the value iteration algorithm in Figure 17.9 is hopelessly inefficient for larger problems—even the 4×3 POMDP is too hard. The main reason is that, given n conditional plans at level d, the algorithm constructs $|A| \cdot n^{|E|}$ conditional plans at level $d+1$ before eliminating the dominated ones. Since the 1970s, when this algorithm was developed, there have been several advances including more efficient forms of value iteration and various kinds of policy iteration algorithms. Some of these are discussed in the notes at the end of the chapter. For general POMDPs, however, finding optimal policies is very difficult (PSPACE-hard, in fact—i.e., very hard indeed). Problems with a few dozen states are often infeasible. The next section describes a different, approximate method for solving POMDPs, one based on look-ahead search.

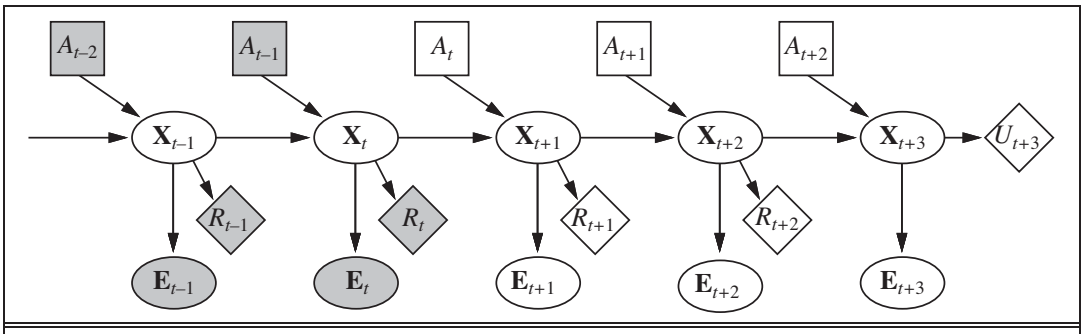

Figure 17.10 The generic structure of a dynamic decision network. Variables with known values are shaded. The current time is t and the agent must decide what to do—that is, choose a value for A_t. The network has been unrolled into the future for three steps and represents future rewards, as well as the utility of the state at the look-ahead horizon.

17.4.3 Online agents for POMDPs

In this section, we outline a simple approach to agent design for partially observable, stochastic environments. The basic elements of the design are already familiar:

- The transition and sensor models are represented by a **dynamic Bayesian network** (DBN), as described in Chapter 15.
- The dynamic Bayesian network is extended with decision and utility nodes, as used in **decision networks** in Chapter 16. The resulting model is called a **dynamic decision network**, or DDN.

DYNAMIC DECISION NETWORK

- A filtering algorithm is used to incorporate each new percept and action and to update the belief state representation.
- Decisions are made by projecting forward possible action sequences and choosing the best one.

DBNs are **factored representations** in the terminology of Chapter 2; they typically have an exponential complexity advantage over atomic representations and can model quite substantial real-world problems. The agent design is therefore a practical implementation of the **utility-based agent** sketched in Chapter 2.

In the DBN, the single state S_t becomes a set of state variables $\mathbf{X}_t$, and there may be multiple evidence variables $\mathbf{E}_t$. We will use A_t to refer to the action at time t, so the transition model becomes $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, A_t)$ and the sensor model becomes $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$. We will use R_t to refer to the reward received at time t and U_t to refer to the utility of the state at time t. (Both of these are random variables.) With this notation, a dynamic decision network looks like the one shown in Figure 17.10.

Dynamic decision networks can be used as inputs for any POMDP algorithm, including those for value and policy iteration methods. In this section, we focus on look-ahead methods that project action sequences forward from the current belief state in much the same way as do the game-playing algorithms of Chapter 5. The network in Figure 17.10 has been projected three steps into the future; the current and future decisions A and the future observations

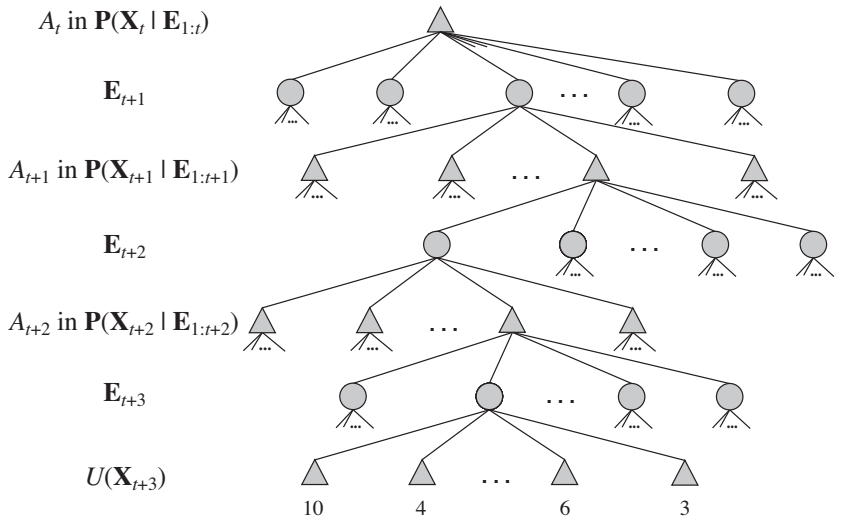

Figure 17.11 Part of the look-ahead solution of the DDN in Figure 17.10. Each decision will be taken in the belief state indicated.

$\mathbf{E}$ and rewards R are all unknown. Notice that the network includes nodes for the *rewards* for $\mathbf{X}_{t+1}$ and $\mathbf{X}_{t+2}$, but the *utility* for $\mathbf{X}_{t+3}$. This is because the agent must maximize the (discounted) sum of all future rewards, and $U(\mathbf{X}_{t+3})$ represents the reward for $\mathbf{X}_{t+3}$ and all subsequent rewards. As in Chapter 5, we assume that U is available only in some approximate form: if exact utility values were available, look-ahead beyond depth 1 would be unnecessary.

Figure 17.11 shows part of the search tree corresponding to the three-step look-ahead DDN in Figure 17.10. Each of the triangular nodes is a belief state in which the agent makes a decision A_{t+i} for $i = 0, 1, 2, \ldots$. The round (chance) nodes correspond to choices by the environment, namely, what evidence $\mathbf{E}_{t+i}$ arrives. Notice that there are no chance nodes corresponding to the action outcomes; this is because the belief-state update for an action is deterministic regardless of the actual outcome.

The belief state at each triangular node can be computed by applying a filtering algorithm to the sequence of percepts and actions leading to it. In this way, the algorithm takes into account the fact that, for decision A_{t+i}, the agent *will* have available percepts $\mathbf{E}_{t+1}, \ldots, \mathbf{E}_{t+i}$, even though at time t it does not know what those percepts will be. In this way, a decision-theoretic agent automatically takes into account the value of information and will execute information-gathering actions where appropriate.

A decision can be extracted from the search tree by backing up the utility values from the leaves, taking an average at the chance nodes and taking the maximum at the decision nodes. This is similar to the EXPECTIMINIMAX algorithm for game trees with chance nodes, except that (1) there can also be rewards at non-leaf states and (2) the decision nodes correspond to belief states rather than actual states. The time complexity of an exhaustive search to depth d is $O(|A|^d \cdot |\mathbf{E}|^d)$, where $|A|$ is the number of available actions and $|\mathbf{E}|$ is the number of possible percepts. (Notice that this is far less than the number of depth-d conditional

plans generated by value iteration.) For problems in which the discount factor γ is not too close to 1, a shallow search is often good enough to give near-optimal decisions. It is also possible to approximate the averaging step at the chance nodes, by sampling from the set of possible percepts instead of summing over all possible percepts. There are various other ways of finding good approximate solutions quickly, but we defer them to Chapter 21.

Decision-theoretic agents based on dynamic decision networks have a number of advantages compared with other, simpler agent designs presented in earlier chapters. In particular, they handle partially observable, uncertain environments and can easily revise their "plans" to handle unexpected evidence. With appropriate sensor models, they can handle sensor failure and can plan to gather information. They exhibit "graceful degradation" under time pressure and in complex environments, using various approximation techniques. So what is missing? One defect of our DDN-based algorithm is its reliance on forward search through state space, rather than using the hierarchical and other advanced planning techniques described in Chapter 11. There have been attempts to extend these techniques into the probabilistic domain, but so far they have proved to be inefficient. A second, related problem is the basically propositional nature of the DDN language. We would like to be able to extend some of the ideas for first-order probabilistic languages to the problem of decision making. Current research has shown that this extension is possible and has significant benefits, as discussed in the notes at the end of the chapter.

17.5 Decisions with Multiple Agents: Game Theory

This chapter has concentrated on making decisions in uncertain environments. But what if the uncertainty is due to other agents and the decisions they make? And what if the decisions of those agents are in turn influenced by our decisions? We addressed this question once before, when we studied games in Chapter 5. There, however, we were primarily concerned with turn-taking games in fully observable environments, for which minimax search can be used to find optimal moves. In this section we study the aspects of **game theory** that analyze games with simultaneous moves and other sources of partial observability. (Game theorists use the terms **perfect information** and **imperfect information** rather than fully and partially observable.) Game theory can be used in at least two ways:

GAME THEORY

1. **Agent design**: Game theory can analyze the agent's decisions and compute the expected utility for each decision (under the assumption that other agents are acting optimally according to game theory). For example, in the game **two-finger Morra**, two players, O and E, simultaneously display one or two fingers. Let the total number of fingers be f. If f is odd, O collects f dollars from E; and if f is even, E collects f dollars from O. Game theory can determine the best strategy against a rational player and the expected return for each player.[4]

[4] Morra is a recreational version of an **inspection game**. In such games, an inspector chooses a day to inspect a facility (such as a restaurant or a biological weapons plant), and the facility operator chooses a day to hide all the nasty stuff. The inspector wins if the days are different, and the facility operator wins if they are the same.