UNIT-4

How an agent should make decisions so that it gets what it wants.

How utility theory combines with probability theory to yield a decision-theoretic agent—an agent that can make rational decisions based on what it believes and what it wants.

Such an agent can make decisions in contexts in which uncertainty and conflicting goals leave a logical agent with no way to decide: a goal-based agent has a binary distinction between good (goal) and bad (non-goal) states

COMBINING BELIEFS AND DESIRES UNDER UNCERTAINTY

Decision theory, in its simplest form, deals with choosing among actions based on the desirability of their *immediate* outcomes. the notation RESULT(s0, a) for the state that is the deterministic outcome of taking action a in state s0. The probability of outcome s, given evidence observations **e**, is written

P(RESULT(a)=s| a, **e**)

The agent's preferences are captured by a **utility function**, U(s), which assigns a single number to express the desirability of a state. The **expected utility** of an action given the evidence, EU(a|**e**), is just the average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$EU(a|\mathbf{e}) = \sum_{s'} P(\textsc{Result}(a) = s' \mid a, \mathbf{e})\, U(s')$$

The principle of **maximum expected utility** (MEU) says that a rational agent should choose

the action that maximizes the agent's expected utility:

$$action = \operatorname*{argmax}_{a} EU(a|\mathbf{e})$$

Computing the outcome utilities U(s) often requires searching or planning, because an agent may not know how good a state is until it knows where it can get to from that state. So, decision theory is not a panacea that solves the AI problem—but it does provide a useful framework.

Consider the environments that could lead to an agent having a given percept history, and consider the different agents that we could design. *If an agent acts so as to maximize a utility function that correctly reflects the performance measure, then the agent will achieve the highest possible performance score.*

THE BASIS OF UTILITY THEORY

The principle of Maximum Expected Utility (MEU) seems like a reasonable way to make decisions, but it is by no means obvious that it is the *only* rational way.

Constraints on rational preferences

the following notation to describe an agent's preferences:

the agent prefers A over B. $A \succ B$

the agent is indifferent between A and B. $A \sim B$

the agent prefers A over B or is indifferent between them. $A \succsim B$

A lottery L with possible outcomes S1, . . . , Sn that occur with probabilities p1, . . . , pn is written L = [p1, S1; p2, S2; . . . pn, Sn] . each outcome Si of a lottery can be either an atomic state or another lottery.

Contraints or Preferences:

Orderability: Given any two lotteries, a rational agent must either prefer one to the other or else rate the two as equally preferable. That is, the agent cannot avoid deciding.

$$\text{Exactly one of } (A \succ B),\ (B \succ A),\ \text{ or } (A \sim B) \text{ holds.}$$

Transitivity: Given any three lotteries, if an agent prefers A to B and prefers B to C, then the agent must prefer A to C.

$$(A \succ B) \land (B \succ C) \;\Rightarrow\; (A \succ C)$$

Continuity: If some lottery B is between A and C in preference, then there is some probability p for which the rational agent will be indifferent between getting B for sure and the lottery that yields A with probability p and C with probability 1 − p.

$$A \succ B \succ C \;\Rightarrow\; \exists p \;\; [p, A;\; 1-p, C] \sim B\,.$$

Substitutability: If an agent is indifferent between two lotteries A and B, then the agent is indifferent between two more complex lotteries that are the same except that B is substituted for A in one of them. This holds regardless of the probabilities and the other outcome(s) in the lotteries.

$$A \sim B \Rightarrow [p,A;\ 1-p,C] \sim [p,B;\ 1-p,C]\,.$$

This also holds if we substitute $\succ$ for ∼ in this axiom.

Monotonicity: Suppose two lotteries have the same two possible outcomes, A and B. If an agent prefers A to B, then the agent must prefer the lottery that has a higher probability for A

$$A \succ B \;\Rightarrow\; (p > q \;\Leftrightarrow\; [p, A;\; 1-p, B] \succ [q, A;\; 1-q, B])$$

Decomposability: Compound lotteries can be reduced to simpler ones using the laws of probability. This has been called the "no fun in gambling" rule because it says that two consecutive lotteries can be compressed into a single equivalent lottery.

$$[p, A;\; 1-p, [q, B;\; 1-q, C]] \sim [p, A;\; (1-p)q, B;\; (1-p)(1-q), C]$$

These constraints are known as the axioms of utility theory the axioms of utility we can derive the following consequences

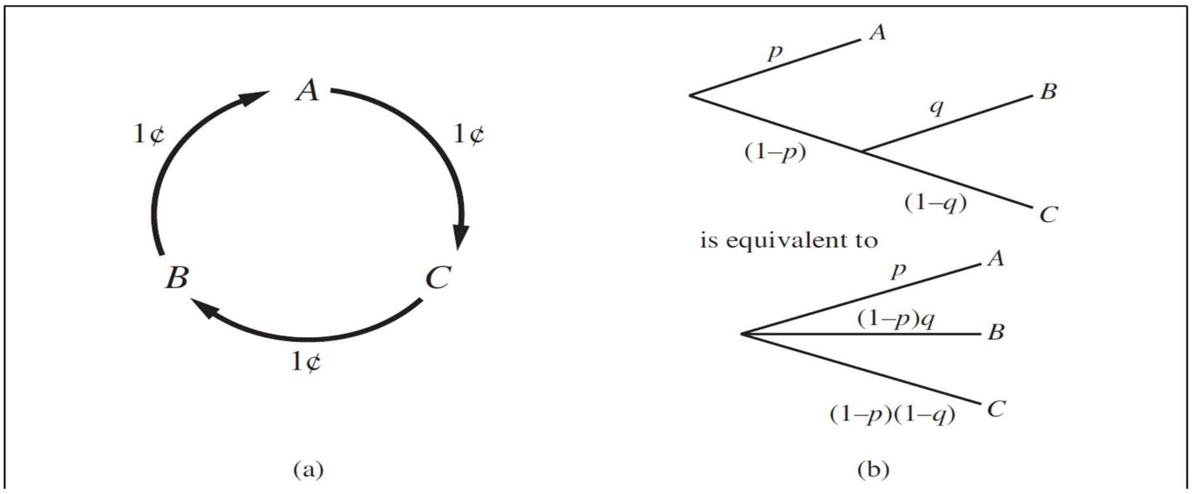
Existence of Utility Function: If an agent's preferences obey the axioms of utility, then there exists a function U such that U(A) > U(B) if and only if A is preferred to B, and U(A) = U(B) if and only if the agent is indifferent between A and B.

$U(A) > U(B) \Leftrightarrow A \succ B$

$U(A) = U(B) \Leftrightarrow A \sim B$

Expected Utility of a Lottery: The utility of a lottery is the sum of the probability of each outcome times the utility of that outcome.

$$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$$



UTILITY FUNCTIONS

Utility is a function that maps from lotteries to real numbers. We know there are some axioms on utilities that all rational agents must obey.

Utility assessment and utility scales

we want to build a decision-theoretic system that helps the agent make decisions or acts on his or her behalf, we must first work out what the agent's utility function is. This process, often called **preference elicitation**, A scale can be established by fixing the utilities of any two particular outcomes, just as we fix a temperature scale by fixing the freezing point and boiling point of water. Typically, we fix the utility of a “best possible prize” at U(S) = u_ and a “worst possible catastrophe” at U(S) = u⊥. **Normalized utilities** use a scale with u⊥ = 0 and u_ NORMALIZED = 1.

Given a utility scale between u_ and u⊥, we can assess the utility of any particular prize S by asking the agent to choose between S and a **standard lottery** [p, u_; (1−p), u⊥]. The probability p is adjusted until the agent is indifferent between S and the standard lottery.

Assuming normalized utilities, the utility of S is given by p. Once this is done for each prize,

the utilities for all lotteries involving those prizes are determined.

For example, driving in a car for 230 miles incurs a risk of one micromort; over the life of your car—say, 92,000 miles— that’s 400 micromorts. People appear to be willing to pay about $10,000 (at 2009 prices) more for a safer car that halves the risk of death, or about $50 per micromort. A number

of studies have confirmed a figure in this range across many individuals and risk types. Of course, this argument holds only for small risks. Most people won’t agree to kill themselves for $50 million.

Another measure is the **QALY**, or quality-adjusted life year. Patients with a disability are willing to accept a shorter life expectancy to be restored to full health. For example, kidney patients on average are indifferent between living two years on a dialysis machine and one year at full health.

The utility of money

Utility theory has its roots in economics, and economics provides one obvious candidate

for a utility measure: money The almost universal exchangeability of money for all kinds of goods and services suggests that money plays a significant role in human utility functions.

It will usually be the case that an agent prefers more money to less, all other things being equal. the agent exhibits a **monotonic preference** for more money. This does not mean that money behaves as a utility function, because it says nothing about preferences between *lotteries* involving money.

The host now offers you a choice: either you can take the $1,000,000 prize or you can gamble it on the flip of a coin. If the coin comes up heads, you end up with nothing, but if it comes up tails, you get $2,500,000.

Assuming the coin is fair, the **expected monetary value** (EMV) of the gamble is 1/2 (\$0)+ 1/2 (\$2,500,000) = \$1,250,000, which is more than the original \$1,000,000.

Suppose we use Sn to denote the state of possessing total wealth \$n, and that your current wealth is \$k. Then the expected utilities of the two actions of accepting and declining the gamble are

EU(Accept) = 1/2U(Sk) + 1/2U(Sk+2,500,000) ,

EU(Decline) = U(Sk+1,000,000)

To determine what to do, we need to assign utilities to the outcome states. Utility is not directly proportional to monetary value, because the utility for your first million is very high, whereas the utility for an additional million is smaller. On the other hand, a billionaire would most likely have a utility function that is locally linear over the range of a few million more, and thus would accept the gamble.

The data obtained for Mr. Beard's preferences are consistent with a utility function

U(Sk+n) = −263.31 + 22.09 log(n + 150,000)

For example, someone already \$10,000,000 in debt might well accept a gamble on a fair coin with a gain of \$10,000,000 for heads and a loss of \$20,000,000 for tails. This yields the S-shaped curve.

Expected utility and post-decision disappointment

The rational way to choose the best action, a∗, is to maximize expected utility:

a∗ = argmax EU(a|**e**)

we have calculated the expected utility correctly according to our probability model, and if the probability model correctly reflects the underlying stochastic processes that generate the outcomes, then, on average, we will get the utility we expect if the whole process is repeated many times.

Decision problem:

Address the computational issues involved in making decisions in a stochastic environment. we are concerned here with **sequential decision problems**, in which the agent's utility depends on a sequence of decisions.

SEQUENTIAL DECISION PROBLEMS

Suppose that an agent is situated in the 4×3 environment. Beginning in the start state, it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or –1. The actions available to the agent in each state are given by ACTIONS(s), A(s); in the 4×3 environment, the actions in every state are

Up, *Down*, *Left*, and *Right*. We assume for now that the environment is **fully observable**, so that the agent always knows where it is.

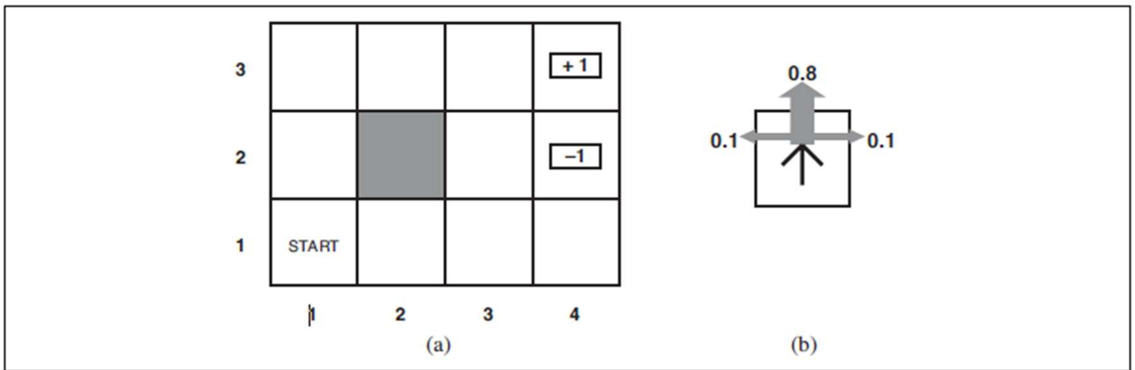


Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

If the environment were deterministic, a solution would be easy: [*Up, Up, Right, Right, Right*]. Unfortunately, the environment won't always go along with this solution, because the actions are unreliable.

Each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction. Furthermore, if the agent bumps into a wall, it stays in the same square. For example, from the start square (1,1), the action *Up* moves the agent to (1,2) with probability 0.8, but with probability 0.1, it moves right to (2,1), and with probability 0.1, it moves left, bumps into the wall, and stays in (1,1). In such an environment, the sequence [Up, Up, Right , Right , Right] goes up around the barrier and reaches the goal state at (4,3) with probability 0.8^5 =0.32768. There is also a small chance of accidentally reaching the goal by going the other way around with probability $0.1^4 \times 0.8$, for a grand total of 0.32776

The **transition model** describes the outcome of each action in each state. Here, the outcome is stochastic, so we write P(s'| s, a) to denote the probability of reaching state s' if action a is done in state s.

P(s' | s, a) as a big three-dimensional table containing probabilities. To complete the definition of the task environment, we must specify the utility function for the agent. Because the decision problem is sequential, the utility function will depend on a sequence of states—an **environment history**—rather than on a single state. we investigate how such utility functions can be specified in general; for now, we simply stipulate that in each state s, the agent receives a **reward** R(s) which may be positive or negative, but must be bounded. For our particular example, the reward is −0.04 in all states except the terminal states. The utility of an environment history is just (for now) the *sum* of the rewards received. For example, if the agent reaches the +1 state after 10 steps, its total utility will be 0.6. The negative reward of –0.04 gives the agent an incentive to reach (4,3) quickly.

A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a **Markov decision process**, and consists of a set of states (with an initial state s0); a set ACTIONS(s) of actions in each state; a transition model P(s' | s, a); and a reward function R(s).

We have seen that any fixed action sequence won't solve the problem, because the agent might end up in a state other than the goal. Therefore, a solution must specify what the agent should do for *any* state that the agent might reach. A solution of this kind is called a **policy.**

It is traditional to denote a policy by π, and π(s) is the action recommended by the policy π for state s.

If the agent has a complete policy, then no matter what the outcome of any action, the agent will always know what to do next.

Each time a given policy is executed starting from the initial state, of the environment may lead to a different environment history. The quality of a policy is therefore measured by the *expected* utility of the possible environment histories generated by that policy. An **optimal policy** is a policy that yields the highest expected utility. We use π∗ to denote an optimal policy. Given π∗, the agent decides what to do by consulting its current percept, which tells it the current state s, and then executing the action π∗(s). A policy represents the agent function explicitly and is therefore a description of a simple reflex agent, computed from the information used for a utility-based agent. Notice that, because the cost of taking a step is fairly small compared with the penalty for ending up in (4,2) by accident, the optimal policy for the state (3,1) is conservative. The policy recommends taking the long way round, rather than taking the shortcut and thereby risking entering (4,2).

The balance of risk and reward changes depending on the value of R(s) for the nonterminal States optimal policies for four different ranges of R(s). When $R(s) \leq -1.6284$, life is so painful that the agent heads straight for the nearest exit, even if the exit is worth –1. When $-0.4278 \leq R(s) \leq -0.0850$, life is quite unpleasant; the agent takes the shortest route to the +1 state and is willing to risk falling into the –1 state by accident. In particular, the agent takes the shortcut from (3,1). When life is only slightly dreary ($-0.0221 < R(s) < 0$), the optimal policy takes *no risks at all*. In (4,1) and (3,2), the agent heads directly away from the –1 state so that it cannot fall in by accident, even though this means banging its head against the wall quite a few times. Finally, if $R(s) > 0$, then life is positively enjoyable and the agent avoids *both* exits. As long as the actions in (4,1), (3,2), and (3,3) are as shown, every policy is optimal, and the agent obtains infinite total reward because it never enters a terminal state.

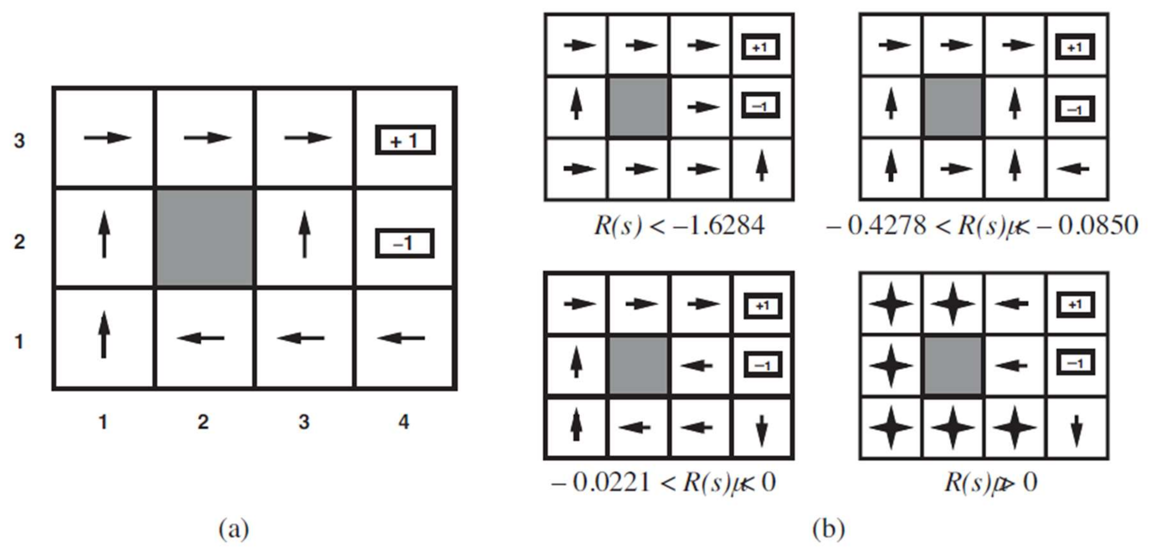


Figure 17.2 (a) An optimal policy for the stochastic environment with $R(s) = -0.04$ in the nonterminal states. (b) Optimal policies for four different ranges of $R(s)$.

The careful balancing of risk and reward is a characteristic of MDPs that does not arise in deterministic search problems; moreover, it is a characteristic of many real-world decision problems. For this reason, MDPs have been studied in several fields, including AI, operations research, economics, and control theory.

Utilities over time

The performance of the agent was measured by a sum of rewards for the states visited. This choice of performance measure is not arbitrary, but it is not the only possibility for the utility function on environment histories, which we write as Uh([s0, s1, . . . , sn]).

A finite horizon means that there is a *fixed* time N after which nothing matters—the game is over, so to speak. Thus, Uh([s0, s1, . . . , sN+k])=Uh([s0, s1, . . . , sN]) for all k > 0.

For example, suppose an agent starts at (3,1) in the 4×3 world, and suppose that N =3. Then, to have any chance of reaching the +1 state, the agent must head directly for it, and the optimal action is to go *Up*. On the other hand, if N =100, then there is plenty of time to take the safe route by going *Left*. *the optimal action in a given state could change over time.*

With no fixed time limit, on the other hand, there is no reason to behave differently in the same state at different times.

Additive rewards:

The utility of a state sequence is Uh([s0, s1, s2, . . .]) = R(s0) + R(s1) + R(s2) + · · · .

The 4×3 world in Figure 17.1 uses additive rewards. Notice that additivity was used implicitly in our use of path cost functions in heuristic search algorithms.

2. **Discounted rewards**: The utility of a state sequence is Uh([s0, s1, s2, . . .]) = R(s0) + γR(s1) + γ2R(s2) + · · · , where the **discount factor** γ is a number between 0 and 1.

With discounted rewards, the utility of an infinite sequence is *finite*. In fact, if γ < 1 and rewards are bounded by }Rmax, we have U_h([s0, s1, s2, . . .]) = $\sum_{t=0}^{\infty}$ γtR(st) ≤ $\sum_{t=0}^{\infty}$ γtRmax =

Rmax/(1 −γ)

The true utility of a state is just Uπ∗ (s)—that is, the expected sum of discounted rewards if the agent executes an optimal policy. R(s) is the "short term" reward for being in s, whereas U(s) is the "long term" total reward from s onward.

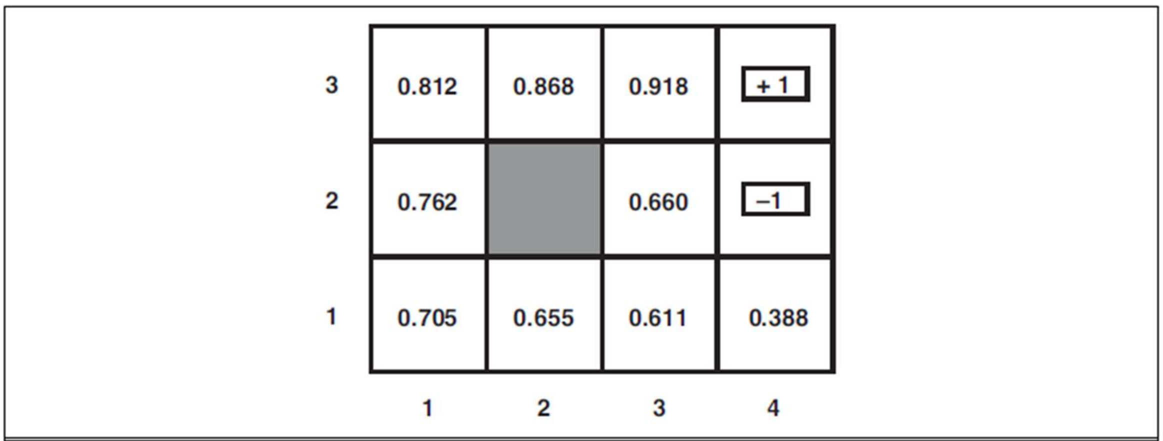


Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

The utility function U(s) allows the agent to select actions by using the principle of maximum expected utility. the action that maximizes the expected utility of the subsequent state:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s') \, .$$

VALUE ITERATION:

Value iteration, for calculating an optimal policy. The basic idea is to calculate the utility of each state and then use the state utilities to select an optimal action in each state.

The Bellman equation for utilities

The utility of being in a state as the expected sum of discounted rewards from that point onwards. From this, it follows that there is a direct relationship between the utility of a state and the utility of its neighbors: *the utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.* That is, the utility of a state is given by

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s') \, .$$

This is called the **Bellman equation**.

The utilities of the states—defined as the expected utility of subsequent state sequences—are

solutions of the set of Bellman equations. Let us look at one of the Bellman equations for the 4×3 world. The equation for the state (1,1) is

$$\begin{aligned} U(1, 1) = -0.04 + \gamma \max[\ & 0.8U(1, 2) + 0.1U(2, 1) + 0.1U(1, 1), && \text{(Up)} \\ & 0.9U(1, 1) + 0.1U(1, 2), && \text{(Left)} \\ & 0.9U(1, 1) + 0.1U(2, 1), && \text{(Down)} \\ & 0.8U(2, 1) + 0.1U(1, 2) + 0.1U(1, 1)\ ]. && \text{(Right)} \end{aligned}$$

The value iteration algorithm:

The Bellman equation is the basis of the value iteration algorithm for solving MDPs. If there are n possible states, then there are n Bellman equations, one for each state. The n equations contain n unknowns—the utilities of the states. So we would like to solve these simultaneous equations to find the utilities. There is one problem: the equations are *nonlinear*, because the "max" operator is not a linear operator. Whereas systems of linear equations can be solved quickly using linear algebra techniques, systems of nonlinear equations are more problematic.

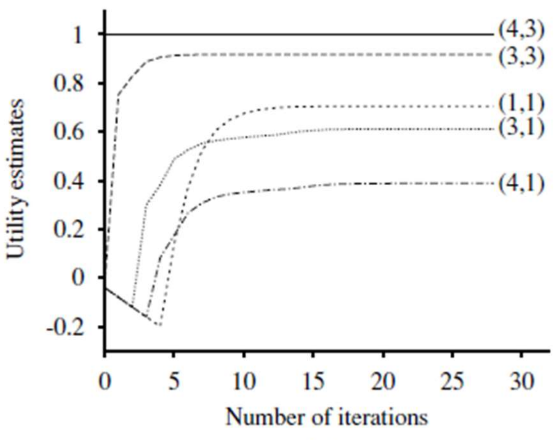
One thing to try is an *iterative* approach. We start with arbitrary initial values for the utilities, calculate the right-hand side of the equation, and plug it into the left-hand side—thereby updating the utility of each state from the utilities of its neighbors. We repeat this until we reach an equilibrium.

Let Ui(s) be the utility value for state s at the ith iteration. The iteration step, called a **Bellman update**, looks

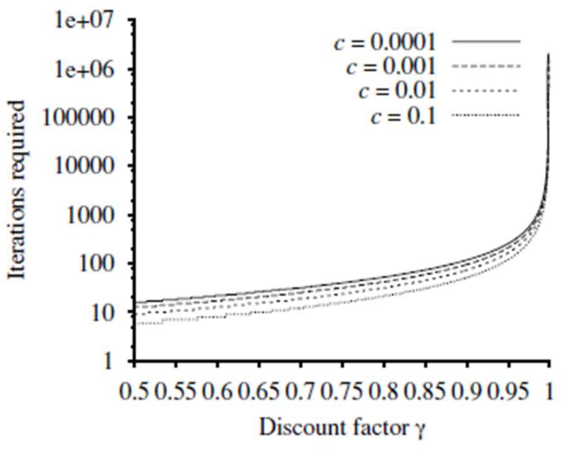
$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s') \, ,$$

```
function VALUE-ITERATION($mdp, \epsilon$) returns a utility function
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
               rewards $R(s)$, discount $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, vectors of utilities for states in $S$, initially zero
                   $\delta$, the maximum change in the utility of any state in an iteration

  repeat
      $U \leftarrow U'$; $\delta \leftarrow 0$
      for each state $s$ in $S$ do
          $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
          if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return $U$
```



(a)



(b)

Value iteration eventually converges to a unique set of solutions of the Bellman equations. we obtain some methods for assessing the error in the utility function returned when the algorithm is terminated early; The basic concept used in showing that value iteration converges is the notion of a **contraction**. A contraction is a function of one argument that, when applied to two different inputs in turn, produces two output values that are “closer together,” by at least some constant factor, than the original inputs. For example, the function “divide by two” is a contraction, because, after we divide any two numbers by two, their difference is halved. Notice that the “divide by two” function has a fixed point, namely zero, that is unchanged by the application of the function.

we can discern two important properties of contractions:

• A contraction has only one fixed point; if there were two fixed points they would not get closer together when the function was applied, so it would not be a contraction.

• When the function is applied to any argument, the value must get closer to the fixed point (because the fixed point does not move), so repeated application of a contraction always reaches the fixed point in the limit.

POLICY ITERATION

It is possible to get an optimal policy even when the utility function estimate is inaccurate. If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise. This insight suggests an alternative way to find optimal policies. The **policy iteration** algorithm alternates the following two steps, beginning from some initial policy π0:

Policy evaluation: Given a policy π_i, calculate U_i =U πi , the utility of each state if πi were to be executed.

Policy improvement: Calculate a new MEU policy πi+1, using one-step look-ahead based on Ui

The algorithm terminates when the policy improvement step yields no change in the utilities. At this point, we know that the utility function Ui is a fixed point of the Bellman update, so it is a solution to the Bellman equations, and πi must be an optimal policy. Because there are only finitely many policies for a finite state space, and each iteration can be shown to yield a better policy, policy iteration must terminate.

It turns out that doing so is much simpler than solving the standard Bellman equations (which is what value iteration does), because the action in each state is fixed by the policy. At the ith iteration, the policy πi specifies the action πi(s) in state s. This means that we have a simplified version of the Bellman equation relating the utility of s (under πi) to the utilities of its neighbors:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s') .$$

For example, suppose πi is the policy shown in Figure 17.2(a). Then we have πi(1, 1)=Up, πi(1, 2)=Up, and so on, and the simplified Bellman equations are

Ui(1, 1) = −0.04 + 0.8Ui(1, 2) + 0.1Ui(1, 1) + 0.1Ui(2, 1) ,

Ui(1, 2) = −0.04 + 0.8Ui(1, 3) + 0.2Ui(1, 2) ,

.

Instead, we can perform some number of simplified value iteration steps (simplified because the policy is fixed) to give a reasonably good approximation of the utilities. The simplified Bellman update for this process is

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s') ,$$

The resulting algorithm is called **modified policy iteration**.

```
function POLICY-ITERATION($mdp$) returns a policy
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: $U$, a vector of utilities for states in $S$, initially zero
                   $\pi$, a policy vector indexed by state, initially random

  repeat
      $U \leftarrow$ POLICY-EVALUATION($\pi$, $U$, $mdp$)
      $unchanged? \leftarrow$ true
      for each state $s$ in $S$ do
          if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ then do
              $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
              $unchanged? \leftarrow$ false
  until $unchanged?$
  return $\pi$
```

\

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.**

PARTIALLY OBSERVABLE MDPS

Markov decision processes 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 agent does not necessarily know which state it is in, so it cannot execute the action π(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.

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' | s, a), actions A(s), and reward function R(s)—but, like the partially observable search problems, it also has a **sensor model** P(e | s).

For example, we can convert the 4×3 world 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

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 the initial belief state for the 4×3 POMDP could be the uniform distribution over the nine nonterminal states, i.e., <1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9, 0, 0>. 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.

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

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

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=π∗(b).

2. Receive percept e.

3. Set the current belief state to FORWARD(b, a, e) and repeat.

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.

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

In particular, let's calculate the probability that an agent in belief state b reaches belief state b' after executing action a.

P(b' | b, a) and ρ(b) define an *observable* MDP on the space of belief states. Furthermore, it can be shown that an optimal policy for thisMDP, π∗(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.*

$$P(b' \mid b, a) = P(b'|a, b) = \sum_e P(b'|e, a, b) P(e|a, b)$$
$$= \sum_e P(b'|e, a, b) \sum_{s'} P(e \mid s') \sum_s P(s' \mid s, a) b(s) ,$$

Value iteration for POMDPs

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**, for nondeterministic and partially observable problems.

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 αp(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:

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

```
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 $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, sets of plans $p$ with associated utility vectors $\alpha_p$

  $U' \leftarrow$ 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 $U$
```