Robot Learning

MDP, POMDP

Prof. Songhwai Oh

ECE, SNU

AIMA Ch. 16 Making Complex Decisions

Markov Decision Processes

Markov Decision Processes

- **Markov decision process**: a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards
  - Set of states $\mathcal{S}$
  - Set of actions $Actions(s)$ in each state $s \in \mathcal{S}$
  - Transition model $P(s'|s, a)$
  - Reward function $R(s)$
- **Policy** π: $\mathcal{S} \to \{a \mid a \in Actions(s), s \in \mathcal{S}\}$ such that $\pi(s) \in Actions(s)$ for all s. $\pi(s)$ is the action recommended by the policy π for state s.
- **Optimal policy** π^*: a policy with the highest expected utility.

Example

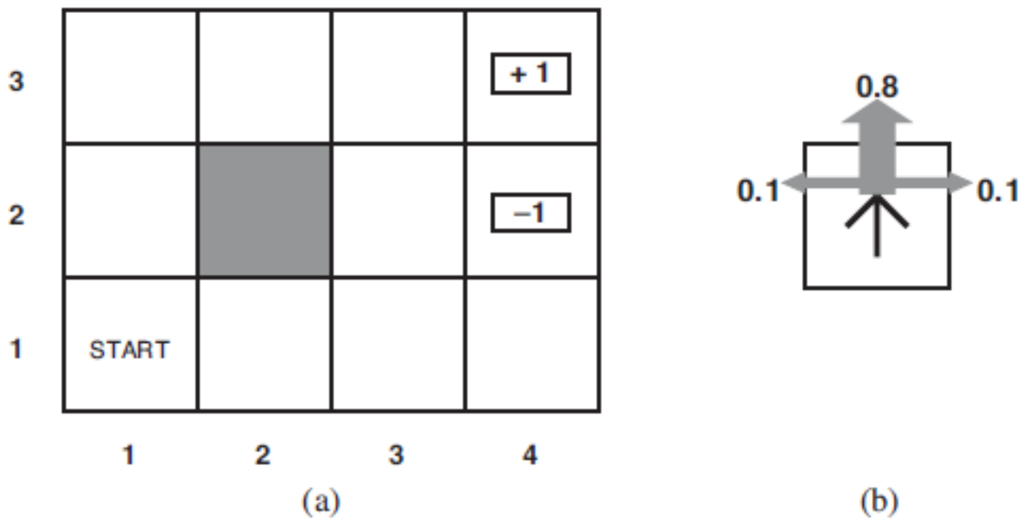
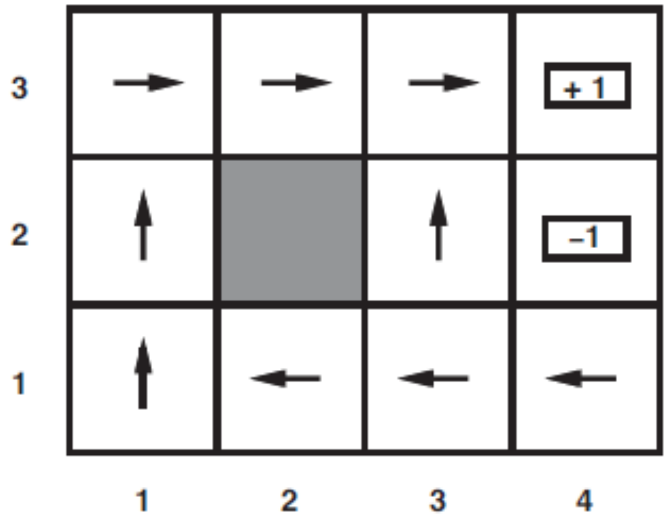


Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision 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.

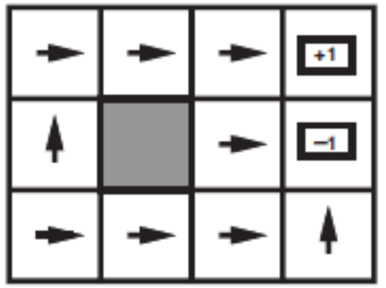
- $Actions(s) = A(s) = [Up, Down, Left, Right]$
- For a deterministic case, a solution is $[Up, Up, Right, Right, Right]$.
- For the transition model given above, the probability of reaching the goal state at $(4, 3)$ using the action sequence $[Up, Up, Right, Right, Right]$ is only $0.8^5 = 0.32768$.

Example: Optimal Policy

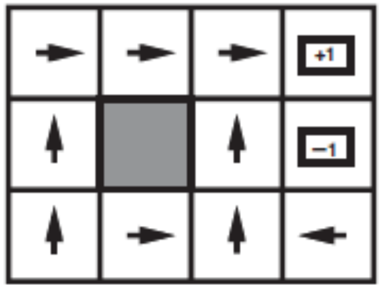
Optimal policy when R(s) = -0.04



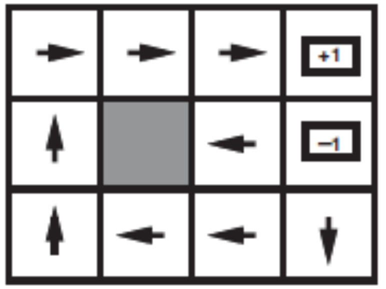
Optimal policies for different reward functions



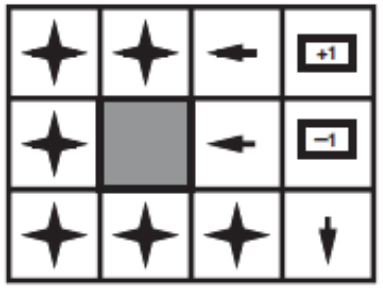
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$

Utilities Over Time

- Two types of problems
  - **Finite horizon problem**: the decision problem terminates at a fixed time N
  - **Infinite horizon problem**: there is no time limit
- With a finite horizon, the optimal action in a given state could change over time. Optimal policy for a finite horizon is nonstationary.
- Optimal policy for an infinite horizon is stationary.

Rewards Under Stationarity

- **Additive rewards**: $U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$.
- **Discounted rewards**: $U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$, where $\gamma \in [0, 1]$ is a discount factor.
- To avoid an infinite utility value for an infinite horizon problem:

    1. **Discounted utility**. Suppose that $|R(s)| \leq R_{max}$ for all s. Then, for $\gamma < 1$,

    $$U([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma}.$$

    2. If the agent is guaranteed to reach a terminal state (such policy is called a **proper policy**).
    3. **Average reward**.

Optimal Policy and Utilities

- Expected utility by executing π starting from s:

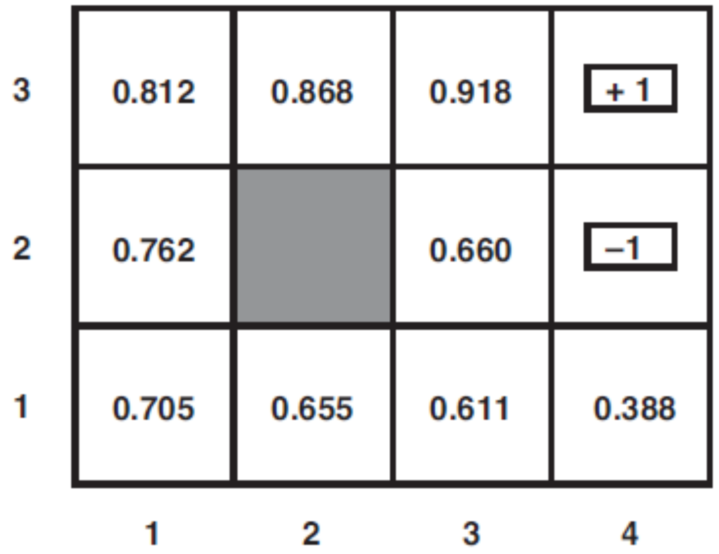
$$U^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t)\right]$$

- Optimal policy starting from s:

$$\pi_s^* = \arg\max_{\pi} U^{\pi}(s).$$

- **Fact**: The optimal policy is *independent* of the starting state for a discounted infinite horizon problem. $\rightarrow$ Optimal policy: π^*
- True utility of state s: $U^{\pi^*}(s) =: U(s)$. (cf. $R(s)$)
- Using the principle of maximum expected utility

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$



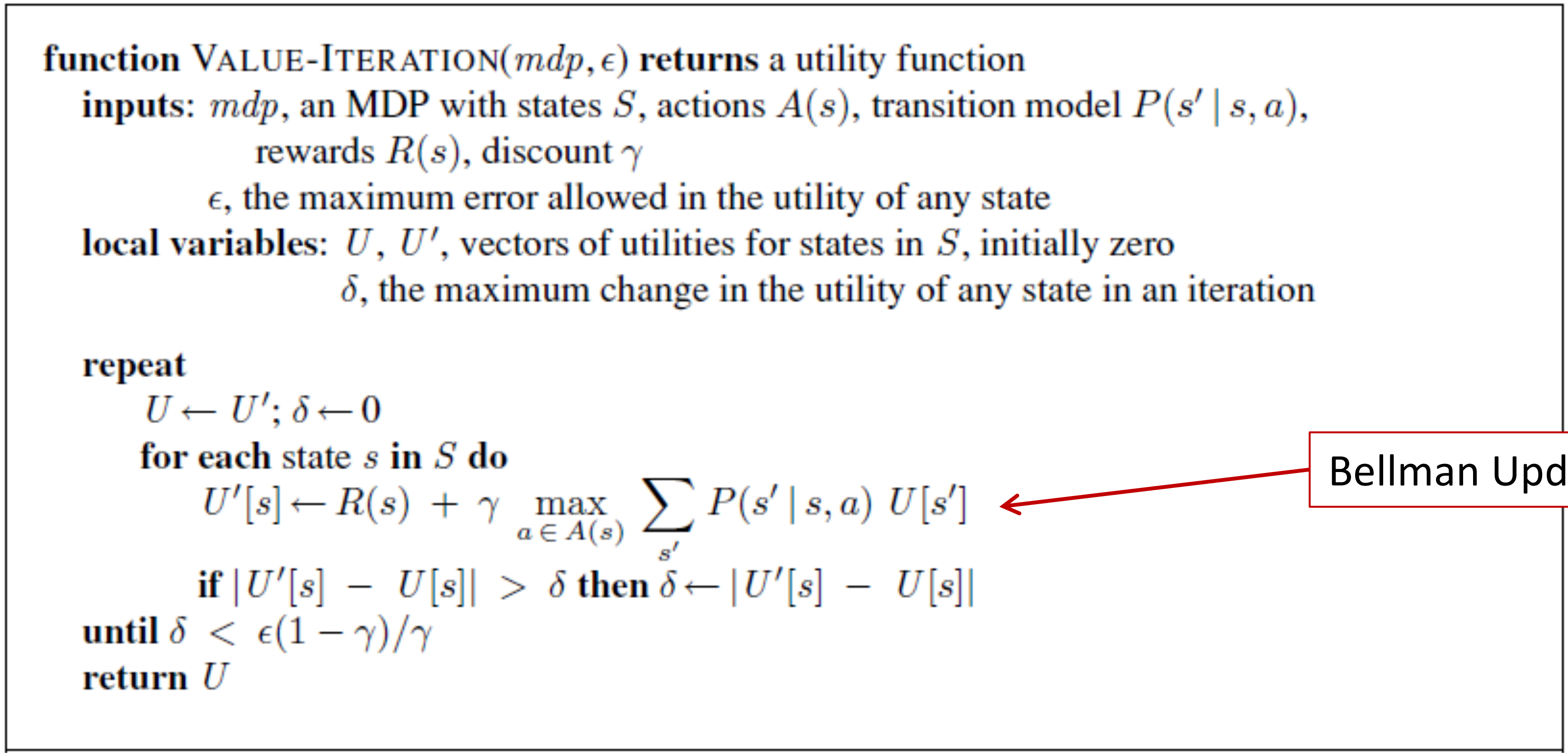
Utilities of states ($\gamma = 1$ and $R(s) = -0.04$ for nonterminal states)

Value Iteration

- **Bellman equation:**

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

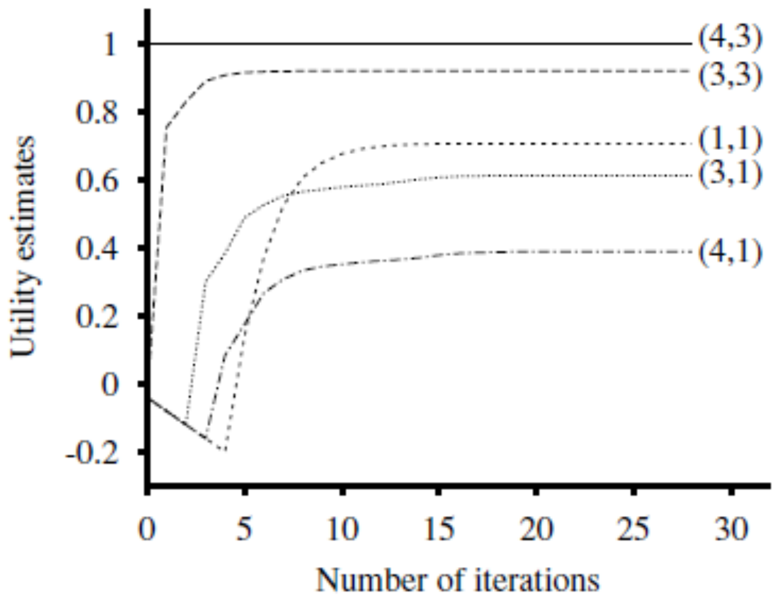
For n states, there are n Bellman equations and n variables (nonlinear equations).

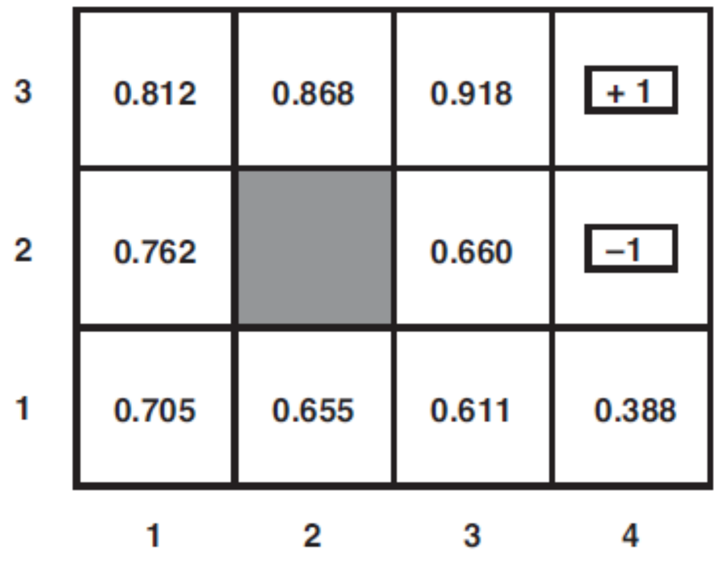


```
function VALUE-ITERATION($mdp, \epsilon$) returns a utility function
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
              rewards $R(s)$, discount $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: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

Bellman Update





Utilities of states ($\gamma = 1$ and $R(s) = -0.04$ for nonterminal states)

Convergence of Value Iteration

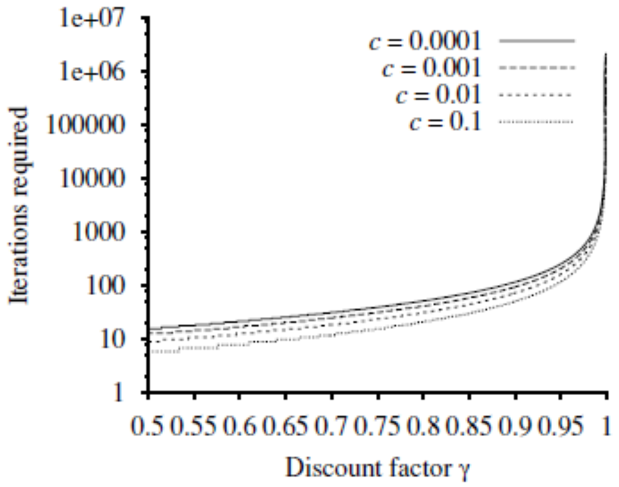
- **Contraction**: a function of one argument that, when applied to two different inputs in turn, produces two output values that are "closer together," by at least some constant factor, than the original inputs. (e.g., "divide by two")
    - A contraction has only one fixed point; if there were two 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.
- $U_{i+1} \leftarrow BU_i$ where B is an operator representing the Bellman update.
- **Fact**: For any U_i and U'_i,

$$\|BU_i - BU'_i\| \leq \gamma \|U_i - U'_i\|,$$

  where $\|U\| = \max_s |U(s)|$ (max-norm or l^∞ norm).
- Hence, the Bellman update is a contraction by a factor of γ and the value iteration always converges to a **unique** solution if $\gamma < 1$.

Convergence Rate

- Contraction: $\|BU_i - BU_i'\| \leq \gamma \|U_i - U_i'\|$
- For the true utilities U, $BU = U$ and $\|BU_i - U\| \leq \gamma \|U_i - U\|$.
- Hence, value iteration converges exponentially fast to the true value.
- Number of iterations needed for an error bound of ϵ:
  - Utilities are bounded by $\pm R_{max}/(1-\gamma)$
  - $\|U_0 - U\| \leq 2R_{max}/(1-\gamma)$, maximum initial error
  - After N iterations, the error is reduced to $\gamma^N \cdot 2R_{max}/(1-\gamma)$
  - Hence, we want $\gamma^N \cdot 2R_{max}/(1-\gamma) \leq \epsilon$ and it requires at most $N = \lceil \log(2R_{max}/\epsilon(1-\gamma))/\log(1/\gamma) \rceil$ iterations.



$c = \epsilon / R_{max}$

- Termination condition:

$$\text{if} \quad \|U_{i+1} - U_i\| < \epsilon(1-\gamma)/\gamma \quad \text{then} \quad \|U_{i+1} - U\| < \epsilon$$

- Current estimate U_i and the maximum expected utility policy π_i using U_i, where $\pi_i(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$
- **Policy loss** $\|U^{\pi_i} - U\|$: the most the agent can lose by executing π_i instead of the optimal policy π^*.
- In practice, it often occurs that π_i becomes optimal long before U_i has converged.

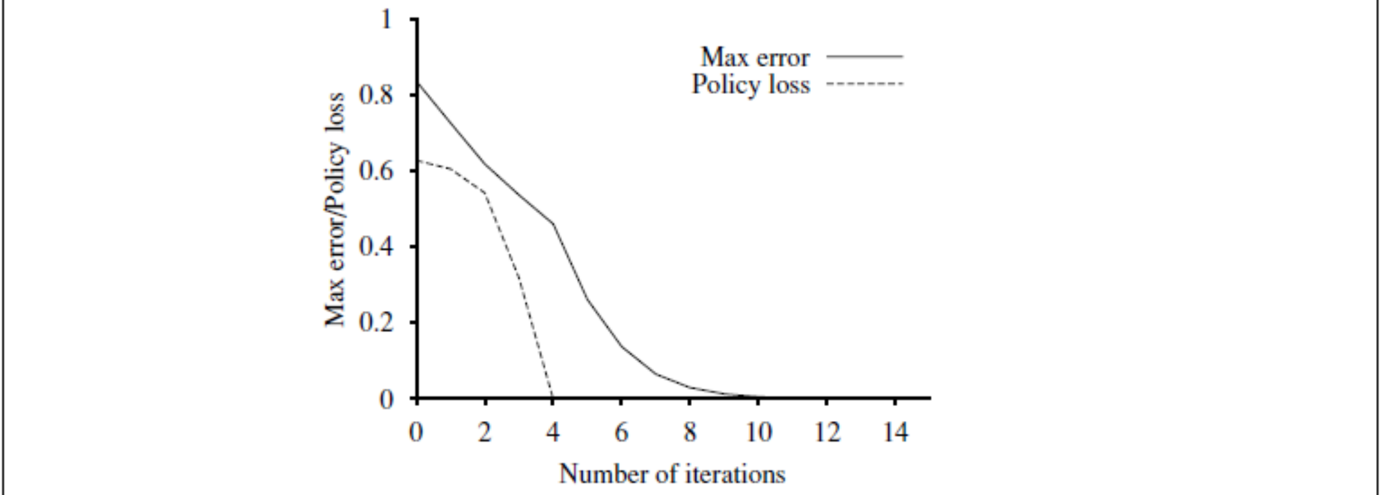


Figure 17.6 The maximum error $||U_i - U||$ of the utility estimates and the policy loss $||U^{\pi_i} - U||$, as a function of the number of iterations of value iteration.

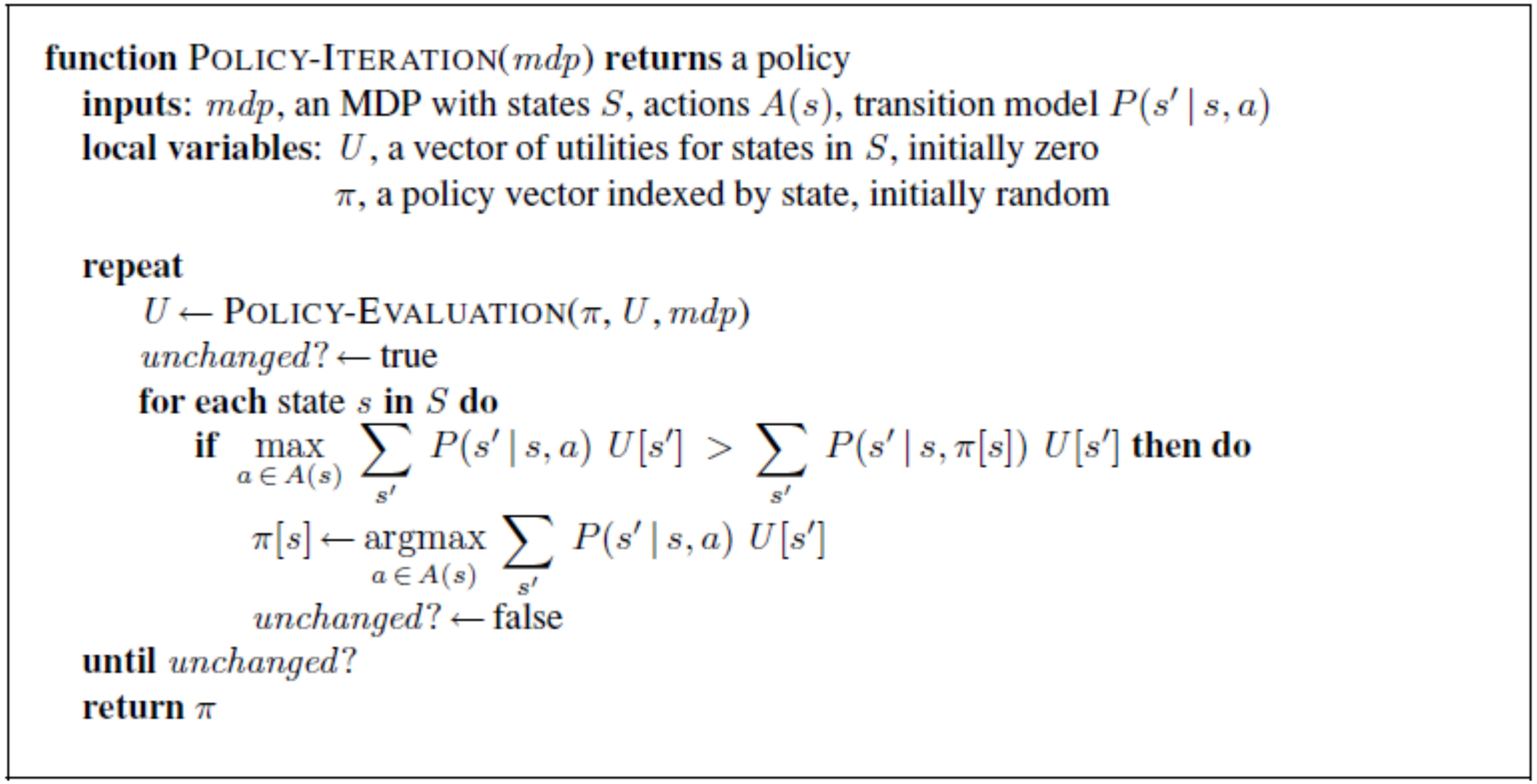
Policy Iteration

- Observations:

  1. It is possible to get an optimal policy even when the utility function estimate is inaccurate.
  2. If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise.

- **Policy iteration** algorithm: repeats followings (begin with initial policy π_0)

  1. **Policy evaluation**: given a policy π_i, calculate $U_i = U^{\pi_i}$, the utility of each state if π_i were to be executed.
  2. **Policy improvement**: calculate a new MEU policy π_{i+1}, using one-step look-ahead based on U_i.



```
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 $\pi$
```

- **Policy evaluation**:

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

For n states, there are n linear equations and n variables.

Partially Observable Markov Decision Process

POMDP

POMDPs

- **Partially observable Markov decision process (POMDP)**: has the same elements as an MDP but the agent does not know which state it is in, hence, partially observable.
  - Set of states $\mathcal{S}$
  - Set of actions $Actions(s)$ in each state $s \in \mathcal{S}$
  - Transition model $P(s'|s,a)$
  - Reward function $R(s)$
  - Sensor model $P(e|s)$
- **Belief state**: the set of actual states the agent might be in. The belief state b is a probability distribution over all possible states.
- Based on observations, we can update the belief state using filtering, i.e., $b' = Forward(b,a,e)$.

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

- In a POMDP, the optimal action depends only on the agent's current belief state.
- Optimal policy $\pi^* : belief\_state \rightarrow actions$ depends on the belief state (not the actual state the agent is in).
- Decision cycle of a POMDP

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

From POMDP to MDP

Probability of observing e, given that a was performed in belief state b:

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

Transition model for the belief-state space:

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

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

Reward function for belief states:

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

- $P(b'|b,a)$ and $\rho(b)$ define an observable MDP on the state of belief states.
- But we have a continuous state space – very complex to solve.

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.

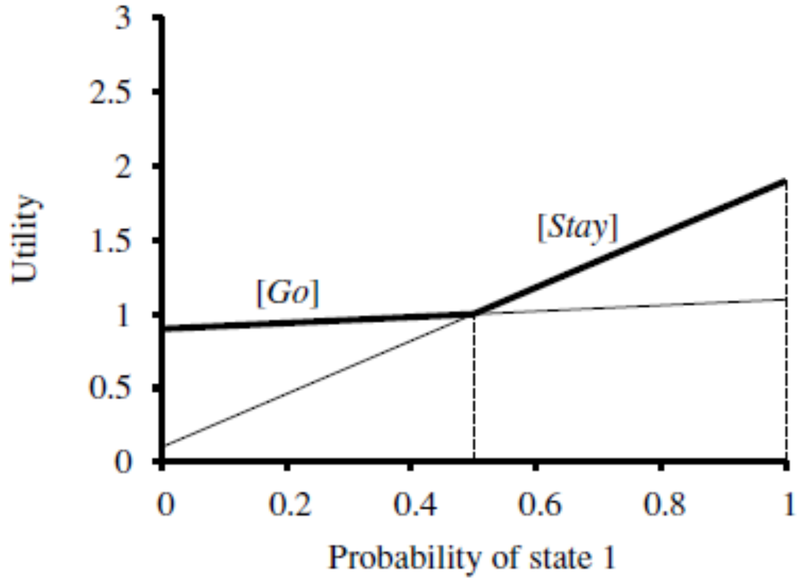
From these two observations, we see that the utility function $U(b)$ on belief states, being the maximum of a collection of hyperplanes, will be *piecewise linear* and *convex*.

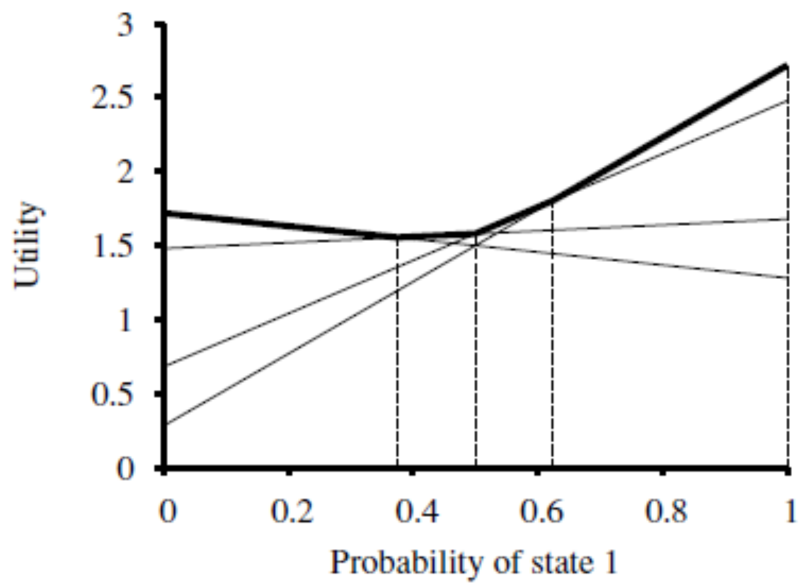
Example

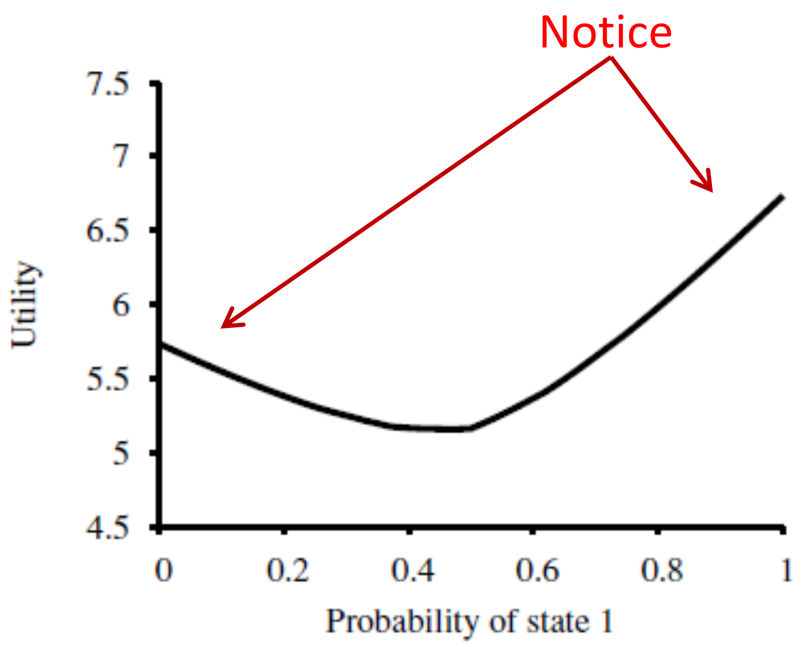
- Two-state world, $\mathcal{S} = \{0, 1\}$. Discount factor $\gamma = 1$.
- Rewards: $R(0) = 0, R(1) = 1$
- Actions: $Stay$ stays with probability (w.p.) 0.9, Go switches the state w.p. 0.9.
- Sensor reports the current state with probability 0.6.
- Expected utility of executing p in b is $b \cdot \alpha_p$ and $U^{\pi^*}(b) = \max_p b \cdot \alpha_p$.

One-step plans:

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







Two-step plans:

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

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

-> 8 distinct depth-2 plans

Eight-step plans

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

- Finding an optimal policy for a POMDP is in general extremely difficult (PSPACE-hard).

Dynamic Decision Networks (DDNs)

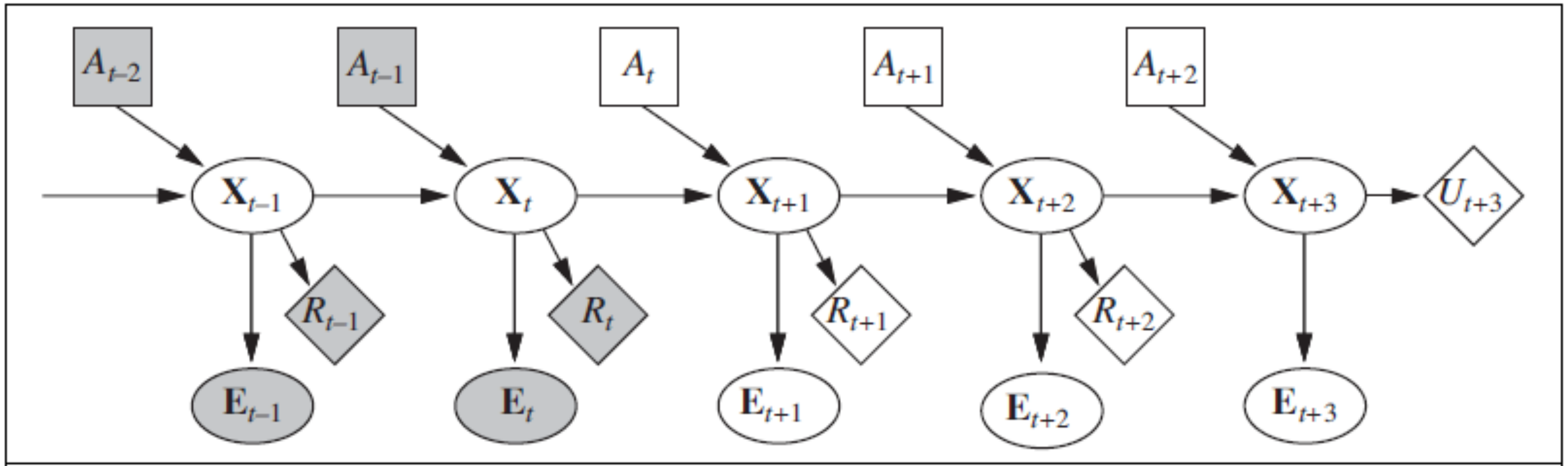


Figure 17.10 The generic structure of a dynamic decision network. Variables with known values are shaded. The current time is t and the agent must decide what to do—that is, choose a value for A_t. The network has been unrolled into the future for three steps and represents future rewards, as well as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

- Transition and sensor models are represented by a dynamic Bayesian network. Transition model: $P(X_{t+1}|X_t, A_t)$; Sensor model: $P(E_t|X_t)$.
- 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.

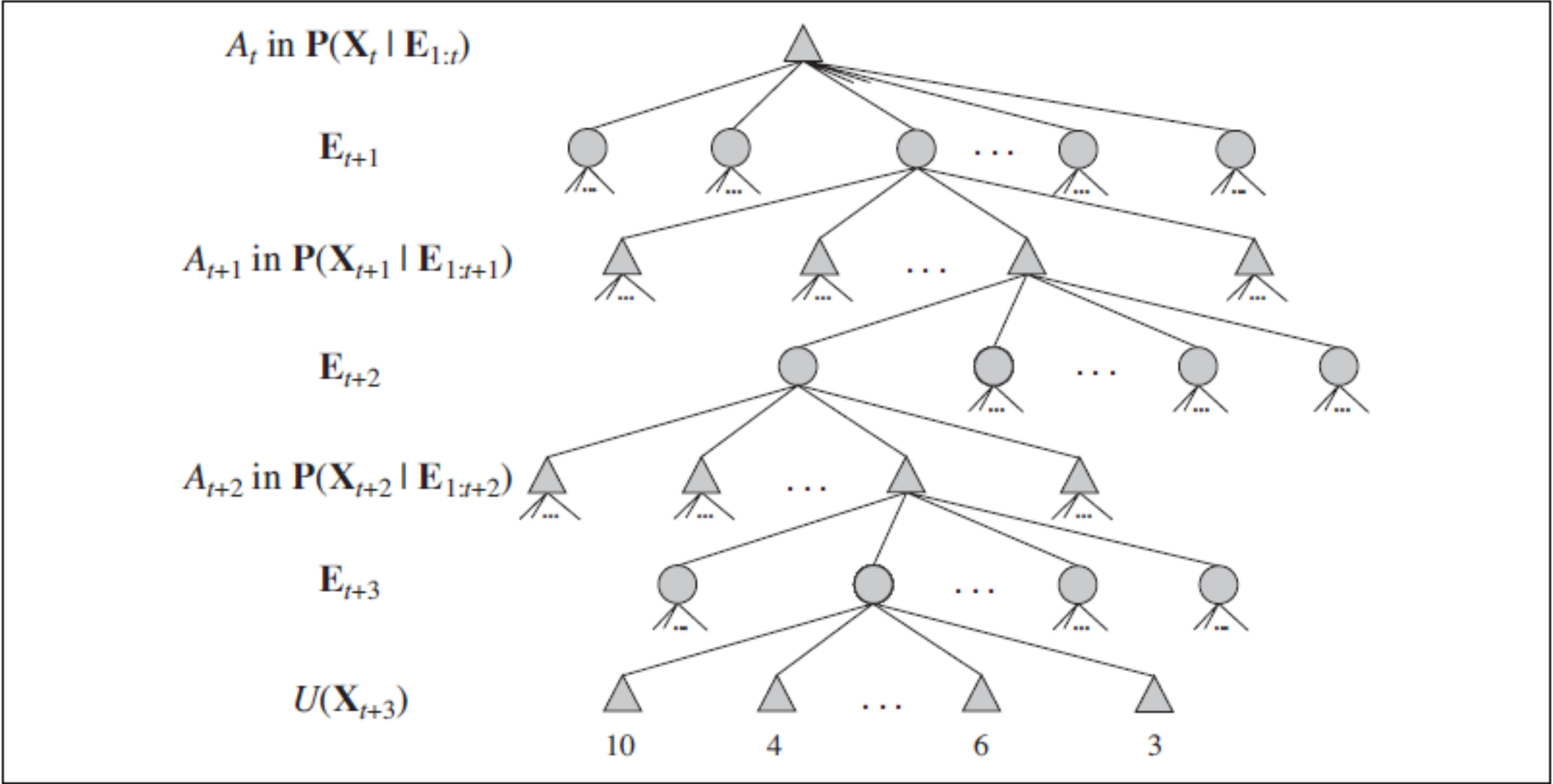

A_t in $\mathbf{P}(\mathbf{X}_t \mid \mathbf{E}_{1:t})$
$\mathbf{E}_{t+1}$
A_{t+1} in $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{E}_{1:t+1})$
$\mathbf{E}_{t+2}$
A_{t+2} in $\mathbf{P}(\mathbf{X}_{t+2} \mid \mathbf{E}_{1:t+2})$
$\mathbf{E}_{t+3}$
$U(\mathbf{X}_{t+3})$
10
4
6
3

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.

- Triangle node: a belief state in which the agent makes a decision A_{t+i}. The belief state can be computed using a filtering algorithm.
- Round node (chance node): choices by the environment, E_{t+i}.