22년 삼성 AI 전문가과정
6월 8일 수요일 4교시
장병탁

인공지능

13차시 : Utility-Based Agents

서울대학교 컴퓨터공학부
담당 교수: 장병탁

Seoul National University
Byoung-Tak Zhang

Lecture Overview

인공지능

13차시 : Utility-Based Agents

서울대학교 컴퓨터공학부
담당 교수: 장병탁

Seoul National University
Byoung-Tak Zhang

Introduction

- **Temporal Reasoning** (Previous lecture)
  - We use probability theory to quantify the degree of belief in elements of the belief state
  - We define basic inference tasks and describes the general structure of inference algorithms for temporal models

    → hidden Markov models, Kalman filters, dynamic Bayesian networks
- **Utility-Based Agents** (This lecture)
  - We examine methods for deciding what to do today, given that we may face another decision tomorrow
  - We are concerned here with sequential decision problems, in which the agent's utility depends on a sequence of decisions
  - Sequential decision problems incorporate utilities, uncertainty, and sensing, and include search and planning problems as special cases
  - We consider the algorithms for MDPs, and the algorithms for solving POMDPs

Sequential Decision Problems

Markov Decision Process (MDP)

- A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards
- Consists of
  - a set of states (with an initial state s_0)
  - a set $\text{Actions}(s)$ of actions in each state
  - a transition model $P(s'|s,a)$
  - a reward function $R(s,a,s')$

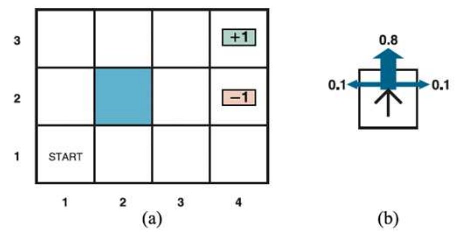


Figure 16.1 (a) A simple, stochastic 4 × 3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. Transitions into the two terminal states have reward +1 and –1, respectively, and all other transitions have a reward of –0.04.

- **Policy** is a solution that must specify what the agent should do for *any* state that the agent might reach.
- An **optimal policy** is a policy that yields the highest expected utility.

Solutions to MDPs

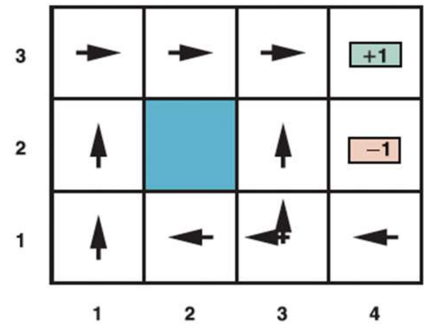
Solution to MDP: The Bellman Equation

- The **utility of a state** is the immediate reward for that state plus the expected discounted utility of the next state,

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

  - This is called the **Bellman Equation**.

- Bellman equations for the 4×3 world (for the state (1,1)):



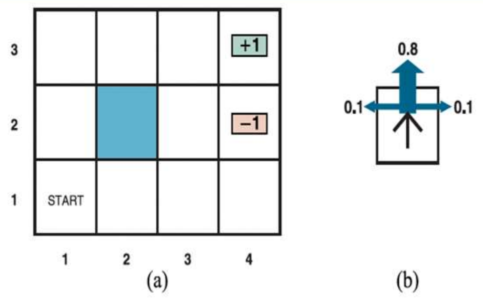
<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

Policy Iteration Algorithm

- Policy iteration uses a **simplified** (linear, no max operation) version of the **Bellman equation** relating the utility of s (under π_i) to the utilities of its neighbors:

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

Partially Observable MDP (POMDP)



The agent must decide what to do now!

- Action at time t: A_t
- State (unobservable): $\mathbf{X}_t$
- Evidence (observable): $\mathbf{E}_t$
- Reward (short-term): R_t
- Utility (long-term): U_t

- Transition model: $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, A_t)$
- Sensor model: $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$

Partially Observable Markov Decision Process (POMDP)

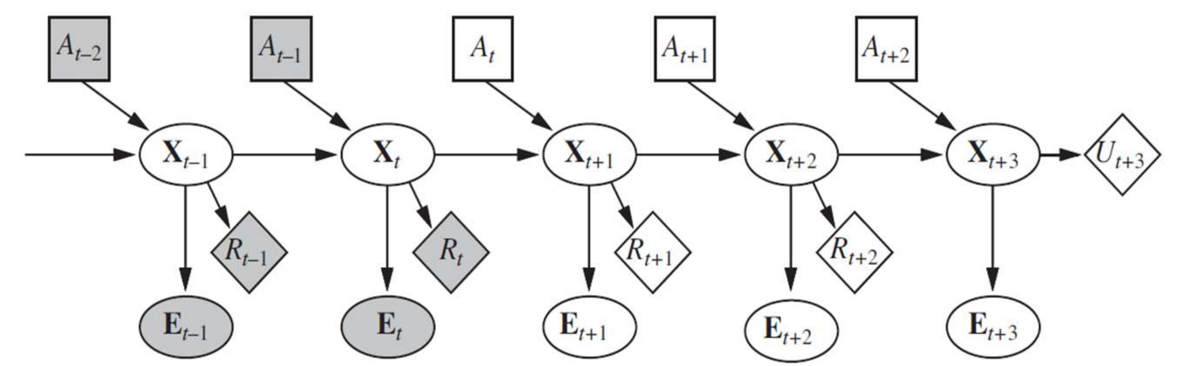


Figure 17.10 The generic structure of a dynamic decision network. Variables with known values are shaded. The current time is t and the agent must decide what to do—that is, choose a value for A_t. The network has been unrolled into the future for three steps and represents future rewards, as well as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

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

Partially Observable MDPs

Definition of POMDPs

- 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

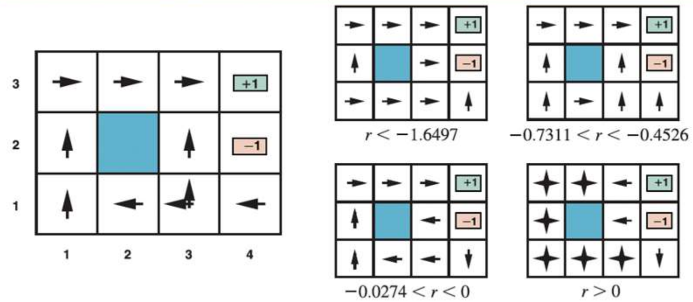
$$b'(s') = \alpha P(e|s') \sum_{s} P(s'|s,a)\, b(s)$$

  - here α is a normalizing constant that makes the belief state sum to 1.
  - $b' = \text{FORWARD}(b, a, e)$

Lecture 13. Utility-Based Agents

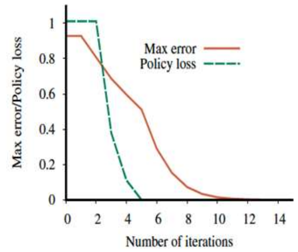
- Sequential Decision Problems
  - Utility function, **MDP**, **POMDP**
- Algorithms for MDPs
  - **Value Iteration**, **Policy Iteration**

    Finding optimal policies: Bellman equations
- Partially Observable MDPs
  - Decision-theoretic agents, Definition of POMDPs
- Algorithms for Solving POMDPs
  - Value Iteration for POMDPS
  - Dynamic decision networks = dynamic Bayesian net + decision net



(a) (b)

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson



Outline (Lecture 13)

13.1 Sequential Decision Problems

13.1 Sequential Decision Problems (1/7)

Utility & Expected utility

- The utility function assigns a single number to express the desirability of a state
- The expected utility of an action given the evidence, $EU(a)$, is just the expected utility average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$\boxed{EU(a) = \sum_{s'} P(\text{RESULT}(a) = s')\, U(s')\,.}$$

- The maximum expected utility (MEU) : a rational agent should choose the action that maximizes the agent's expected utility:

$$\boxed{action = \operatorname*{argmax}_{a} EU(a)\,.}$$

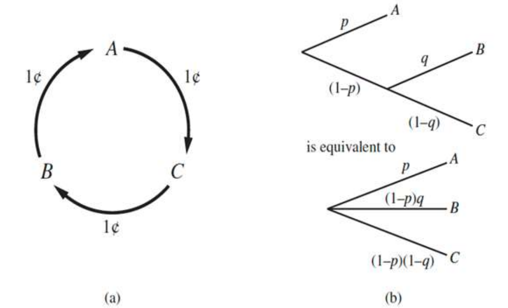


Figure 15.1 (a) Nontransitive preferences $A \succ B \succ C \succ A$ can result in irrational behavior: a cycle of exchanges each costing one cent. (b) The decomposability axiom.

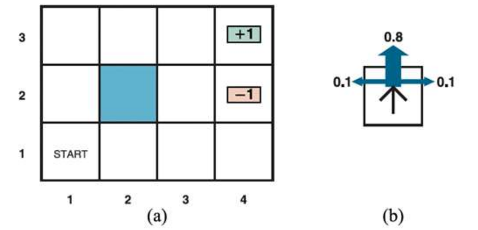
<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.1 Sequential Decision Problems (2/7)

1) Decision Process

- The utility function will depend on a sequence of states—an **environment history**—rather than on a single state.
  - Because the decision problem is sequential
- In each state s, the agent receives a **reward** $R(s)$
  - which may be positive or negative, but must be bounded.



(a) (b)

Figure 16.1 (a) A simple, stochastic 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. Transitions into the two terminal states have reward +1 and –1, respectively, and all other transitions have a reward of –0.04.

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.1 Sequential Decision Problems (3/7)

2) Markov Decision Process (MDP)

- A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards
- Consists of
  - a set of states (with an initial state s_0)
  - a set Actions(s) of actions in each state
  - a transition model $P(s'|s,a)$
  - a reward function $R(s,a,s')$
- **Policy** is a solution that must specify what the agent should do for *any* state that the agent might reach.
- An **optimal policy** is a policy that yields the highest expected utility.

13.1 Sequential Decision Problems (4/7)

2) Markov Decision Process (MDP) contd.

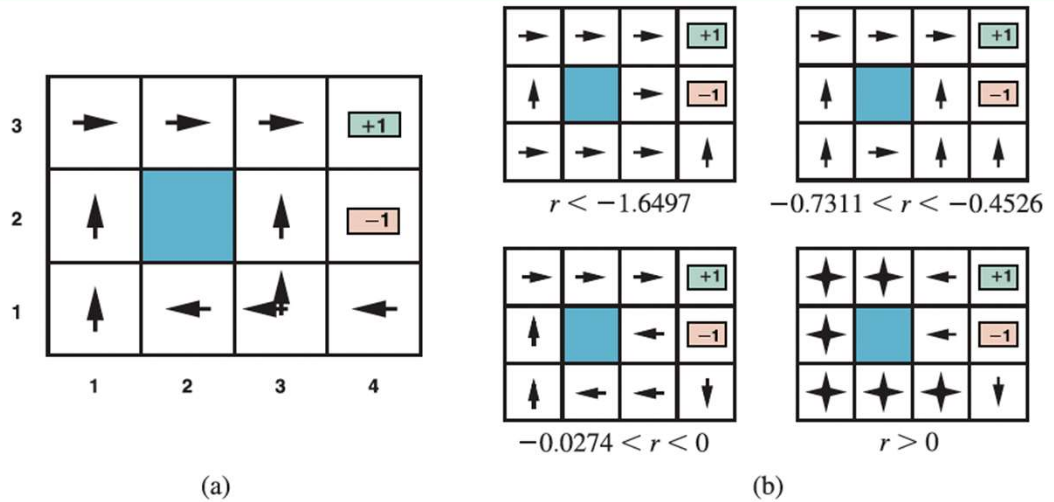


Figure 16.2 (a) The optimal policies for the stochastic environment with $r = -0.04$ for transitions between nonterminal states. There are two policies because in state (3,1) both *Left* and *Up* are optimal. (b) Optimal policies for four different ranges of r.

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.1 Sequential Decision Problems (5/7)

3) Utilities over Time

- The first question to answer is whether there is a **finite horizon** or an **infinite horizon** for decision making.
- **Finite horizon**
  - There is a *fixed* time N after which nothing matters.
  - The **optimal policy** for a finite horizon is **nonstationary**.
- **Infinite horizon**
  - With no fixed time limit, there is no reason to behave differently in the same state at different times.
  - The optimal action depends only on the current state, and the optimal policy is **stationary**.

13.1 Sequential Decision Problems (6/7)

4) Assigning Utilities to Sequences

- **Additive rewards**: $U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$
- **Discounted rewards**: $U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$
  - **discount factor** γ $(0 < \gamma < 1)$: When γ is close to 0, small weight on distant futures. When γ is 1, discounted rewards equivalent to additive rewards
- With discounted rewards, the utility of an infinite sequence is finite. If $\gamma < 1$ and rewards are bounded by $\pm R_{\max}$, we have
  - $U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1-\gamma)$
- **Proper policy:** a policy that is guaranteed to reach a terminal state.

5) Optimal Policies and the Utilities of States

- The expected utility obtained by executing π starting in s is given by
  - $U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})]$
- use π_s^* to denote one of these policies:
  - $\pi_s^* = \text{argmax}_{\pi}\, U^{\pi}(s)$

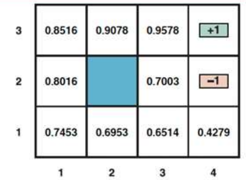


Figure 16.3 The utilities of the states in the 4×3 world with $\gamma = 1$ and $r = -0.04$ for transitions to nonterminal states.

- The utility function $U(s)$ allows the agent to select actions by using the principle of maximum expected utility--choose the action that maximizes the expected utility of the subsequent state:
  - $\pi^*(s) = \text{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma U(s')]$

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.2 Algorithms for MDPs

13.2 Algorithms for MDPs (1/11)

Value Iteration

1) The Bellman Equation for Utilities

- The **utility of a state** is the immediate reward for that state plus the expected discounted utility of the next state,

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

  - This is called the **Bellman Equation**.

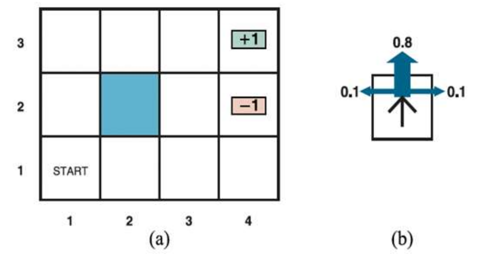


Figure 16.1 (a) A simple, stochastic 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. Transitions into the two terminal states have reward +1 and −1, respectively, and all other transitions have a reward of −0.04.

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

- Bellman equations for the 4×3 world (for (1,1)):

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

13.2 Algorithms for MDPs (2/11)

Value Iteration

2) The Value Iteration Algorithm

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

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow \max_{a \in A(s)}$ Q-VALUE(mdp, s, a, U)
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta \leq \epsilon(1-\gamma)/\gamma$
  return U

Bellman update

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

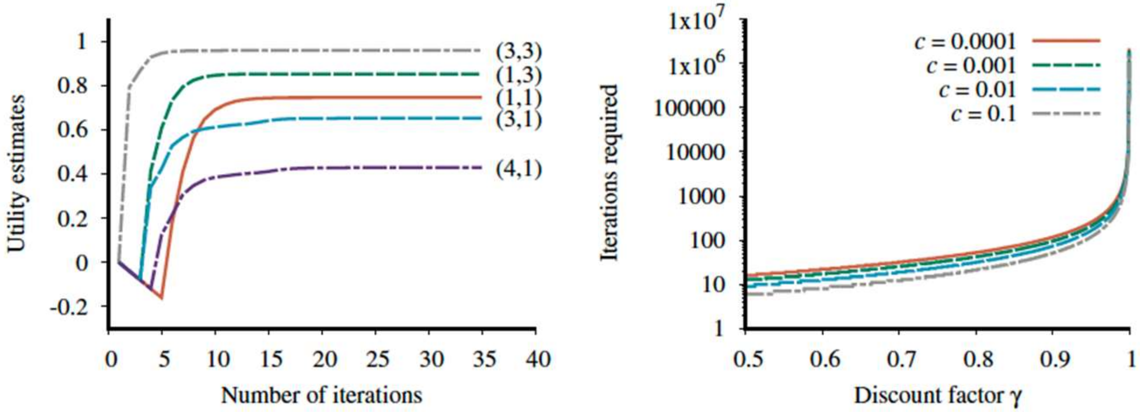
Figure 16.6 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (16.12).

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.2 Algorithms for MDPs (3/11)

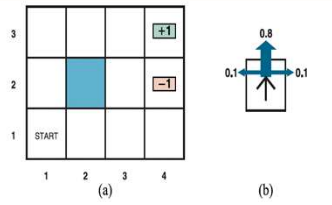
Value Iteration

2) The Value Iteration Algorithm (cont.)



(a) (b)

Figure 16.7 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c, as a function of the discount factor γ.



<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.2 Algorithms for MDPs (4/11)

Value Iteration

3) Convergence of Value Iteration (1/4)

- The basic concept used in showing that value iteration converges is the notion of a **contraction**.
- E.g the function "divide by two" is a contraction. This function has a fixed point, namely zero.
- Two important properties of contractions:
  - A contraction has only one fixed point
  - When the function is applied to any argument, the value must get closer to the fixed point

13.2 Algorithms for MDPs (5/11)

Value Iteration

3) Convergence of Value Iteration (2/4)

- B: the Bellman update operator that is applied simultaneously to update the utility of every state
- U_i: the vector of utilities for all the states at the i-th iteration
- The **Bellman update equation** can be written as $U_{i+1} \leftarrow BU_i$
- The **max norm** to measure distances between utility vectors: $\|U\| = \max_s |U(s)|$
- Let U_i and U_i' be any two utility vectors. Then we have

$$\|BU_i - BU_i'\| \leq \gamma \|U_i - U_i'\|$$

  - Bellman update is a contraction by a factor of γ on the space of utility vectors.
- In particular, we can replace U_i' with the true utilities U, for which $BU = U$.
- Then we obtain the inequality

$$\|BU_i - BU\| \leq \gamma \|U_i - U\|$$

  - if we view $\|U_i - U\|$ as the error in the estimate U_i, we see that the error is reduced by a factor of at least γ on each iteration.
- This means that value iteration converges exponentially fast.

13.2 Algorithms for MDPs (6/11)

Value Iteration

3) Convergence of Value Iteration (3/4)

- We can calculate the number of iterations required to reach a specified error bound as follows: $U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1-\gamma)$
  - First, the utilities of all states are bounded by $\pm R_{max}/(1-\gamma)$.
  - Maximum initial error is $\|U_0 - U\| \leq 2R_{max}/(1-\gamma)$.
  - Because the error is reduced by at least γ each time, $\gamma^N \cdot 2\, R_{max}/(1-\gamma) \leq \epsilon$.
  - Taking logs, $N = \left\lceil \log\left(\frac{2R_{max}}{\epsilon(1-\gamma)}\right) / \log(1/\gamma) \right\rceil$

13.2 Algorithms for MDPs (7/11)

Value Iteration

3) Convergence of Value Iteration (4/4)

- $U^{\pi_i}(s)$ is the utility obtained if π_i is executed starting in s
- The **policy loss** $\|U^{\pi_i} - U\|$ is the most the agent can lose by executing π_i instead of the optimal policy π^*.
- The policy loss of π_i is connected to the error in U_i by the following inequality:
  - If $\|U_i - U\| < \epsilon$ then $\|U^{\pi_i} - U\| < 2\epsilon\gamma/(1-\gamma)$

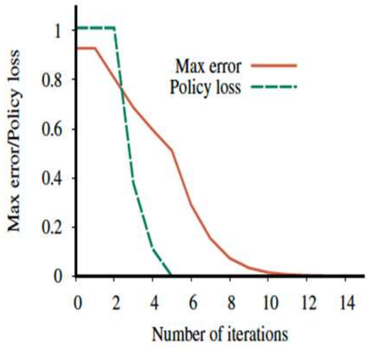


Figure 16.8 The maximum error $\|U_i - U\|$ of the utility estimates and the policy loss $\|U^{\pi_i} - U\|$, as a function of the number of iterations of value iteration on the 4×3 world.

- The policy π_i is optimal (policy loss = 0) when i = 4, even though the maximum error in U_i is still 0.46.

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.2 Algorithms for MDPs (8/11)

Policy Iteration

1) Policy Iteration Algorithm

- Alternates the following two steps, beginning from some initial policy π_0:
  - **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.
  - **Policy improvement:** Calculate a new MEU policy π_{i+1}, using one-step look-ahead based on U_i.
- The algorithm terminates when the policy improvement step yields no change in the utilities.

13.2 Algorithms for MDPs (9/11)

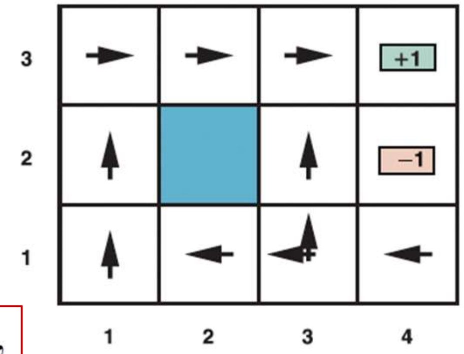
Policy Iteration

1) Policy Iteration Algorithm (contd.)

- Policy iteration uses a **simplified** (linear, no max operation) version of the **Bellman equation** relating the utility of s (under π_i) to the utilities of its neighbors:

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

- For example, suppose π_i is the policy shown in Figure 17.2(a) →
- Then we have $\pi_i(1,1) = Up$, $\pi_i(1,2) = Up$, and so on,
- and the simplified Bellman equations are



$$\boxed{\begin{aligned} U_i(1,1) &= -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)\ , \\ U_i(1,2) &= -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2)\ , \\ &\vdots \end{aligned}}$$

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.2 Algorithms for MDPs (10/11)

Policy Iteration

2) Modified Policy Iteration

- We can perform some number of simplified value iteration steps (simplified because the policy is fixed, no max operation) to give a reasonably good approximation of the utilities.
- The **simplified (linear) Bellman update** for this process is

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

  - This is repeated k times to produce the next utility estimate.
- The resulting algorithm is called **modified policy iteration**.
  - It is often much more efficient than standard policy iteration or value iteration.

13.2 Algorithms for MDPs (11/11)

Policy Iteration

2) Modified Policy Iteration

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

  repeat
     U ← POLICY-EVALUATION(π, U, mdp)
     unchanged? ← true
     for each state s in S do
        a* ← argmax_{a∈A(s)} Q-VALUE(mdp, s, a, U)
        if Q-VALUE(mdp, s, a*, U) > Q-VALUE(mdp, s, π[s], U) then
           π[s] ← a*; unchanged? ← false
  until unchanged?
  return π
```

Figure 16.9 The policy iteration algorithm for calculating an optimal policy.

- Updating the utility or policy(policy imp rovement or simplified value iteration)
  - for all states at once
  - pick any subset of states (asynchronous policy iteration)

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.3 Partially Observable MDPs

13.3 Partially Observable MDPs (1/5)

1) Definition of POMDPs

- When the environment is only **partially observable**, the situation is, one might say, much less clear.
- A **POMDP** has the same elements as an MDP
  - transition model $P(s'|s,a)$,
  - actions $A(s)$,
  - reward function $R(s,a,s')$
- But, like the partially observable search problems of Section 4.4, it also has a **sensor model** $P(e|s)$.
- The sensor model specifies the probability of perceiving evidence e in state s.

13.3 Partially Observable MDPs (2/5)

1) Definition of POMDPs (contd.)

- 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

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

  - where α is a normalizing constant that makes the belief state sum to 1.
  - $b' = \alpha \mathrm{FORWARD}(b, a, e)$

13.3 Partially Observable MDPs (3/5)

2) Cycle of a POMDP Agent

- 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.
- The **decision cycle of a POMDP agent** is like 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 $\mathrm{FORWARD}(b, a, e)$ and repeat.
- The POMDP belief-state space is **continuous**.

13.3 Partially Observable MDPs (4/5)

3) Outcome of Actions

- Calculate the probability that an agent in belief state b reaches belief state b' after executing action a.
- 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:

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

13.3 Partially Observable MDPs (5/5)

3) Outcome of Actions (cont.)

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

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

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

- Reward function for belief states is

$$\rho(b,a) = \sum_s b(s) \sum_{s'} P(s'|s,a)R(s,a,s')$$

13.4 Algorithms for Solving POMDPs

13.4 Algorithms for Solving POMDPs (1/5)

1) Value Iteration for POMDPs

Consider an optimal policy π^* and its application in a specific belief state b. The **policy** is exactly equivalent to a **conditional plan**, as defined in Chapter 4 for nondeterministic and partially observable problems.

- We make two observations:
  - $\alpha_p(s)$: **utility** of executing a fixed **conditional plan (= policy)** p starting in physical state s :

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

$$\alpha_{[Stay]}(A) = 0.9R(A, Stay, A) + 0.1R(A, Stay, B) = 0.1$$

$$\alpha_{[Stay]}(B) = 0.1R(B, Stay, A) + 0.9R(B, Stay, B) = 0.9$$

$$\alpha_{[Go]}(A) = 0.1R(A, Go, A) + 0.9R(A, Go, B) = 0.9$$

$$\alpha_{[Go]}(B) = 0.9R(B, Go, A) + 0.1R(B, Go, B) = 1.1$$

13.4 Algorithms for Solving POMDPs (2/5)

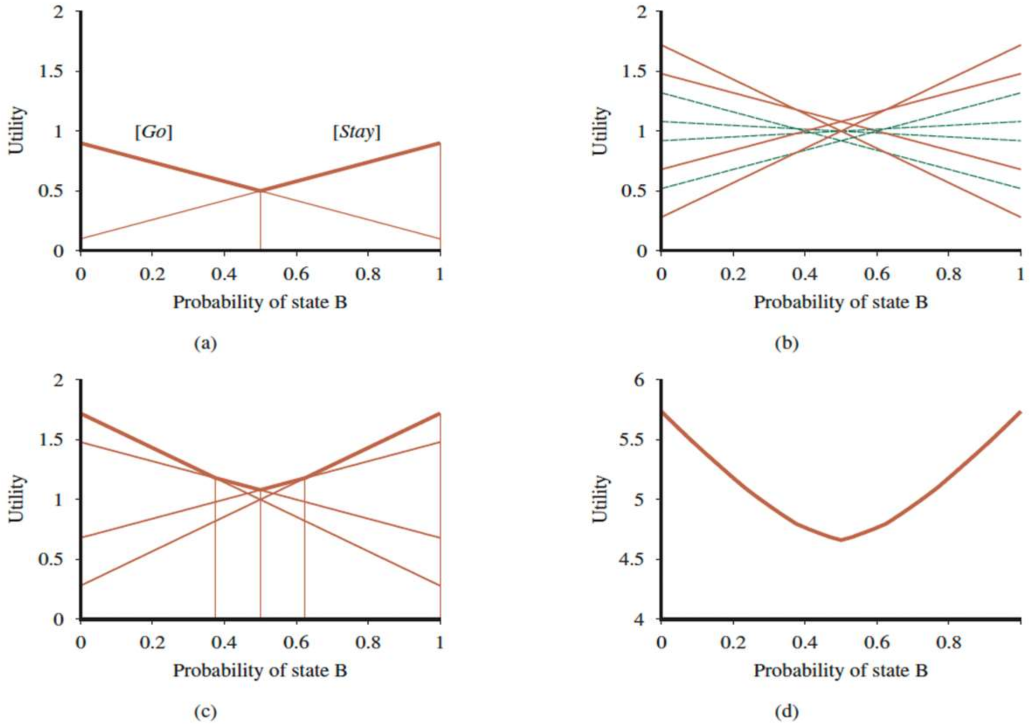


Figure 16.15 (a) Utility of two one-step plans as a function of the initial belief state $b(B)$ for the two-state world, with the corresponding utility function shown in bold. (b) Utilities for 8 distinct two-step plans. (c) Utilities for four undominated two-step plans. (d) Utility function for optimal eight-step plans.

- Undominated plans
- Hyperplane
- Piecewise linear
- Convex

(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.

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.4 Algorithms for Solving POMDPs (3/5)

4) Value Iteration for POMDPs

function POMDP-VALUE-ITERATION(*pomdp*, ϵ) **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 $R(s, a, s')$, 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 all one-step plans $[a]$, with $\alpha_{[a]}(s) = \sum_{s'} P(s' \mid s, a)\, R(s, a, s')$
  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 (16.18)
    $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
  until MAX-DIFFERENCE(U, U') $\leq \epsilon(1-\gamma)/\gamma$
  return U

Figure 16.16 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.

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.4 Algorithms for Solving POMDPs (4/5)

5) Online Agents for POMDPs

- The basic elements of the agent design for partially observable, stochastic environments:
  - The transition and sensor models are represented by a **dynamic Bayesian network** (DBN) (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.
  - 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.

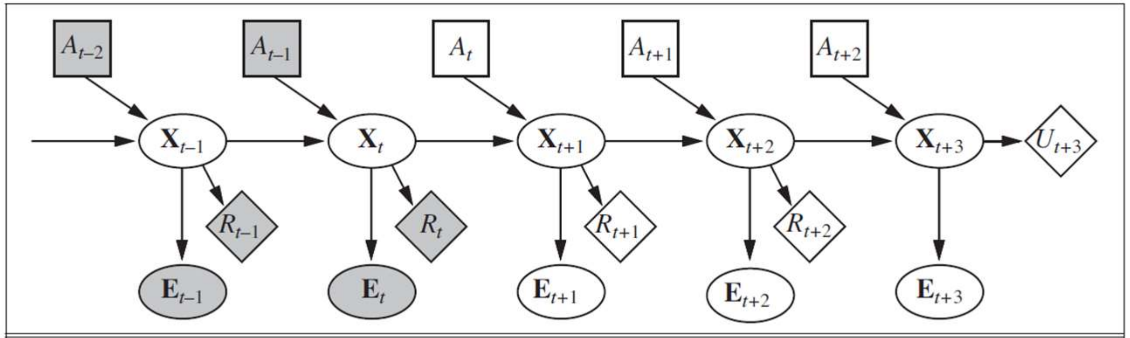


Figure 17.10 The generic structure of a dynamic decision network. Variables with known values are shaded. The current time is t and the agent must decide what to do—that is, choose a value for A_t. The network has been unrolled into the future for three steps and represents future rewards, as well as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

- Action at time t: A_t
- Reward: R_t
- Utility: U_t

- Transition model: $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, A_t)$
- Sensor model: $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

13.4 Algorithms for Solving POMDPs (5/5)

5) Online Algorithms for POMDPs

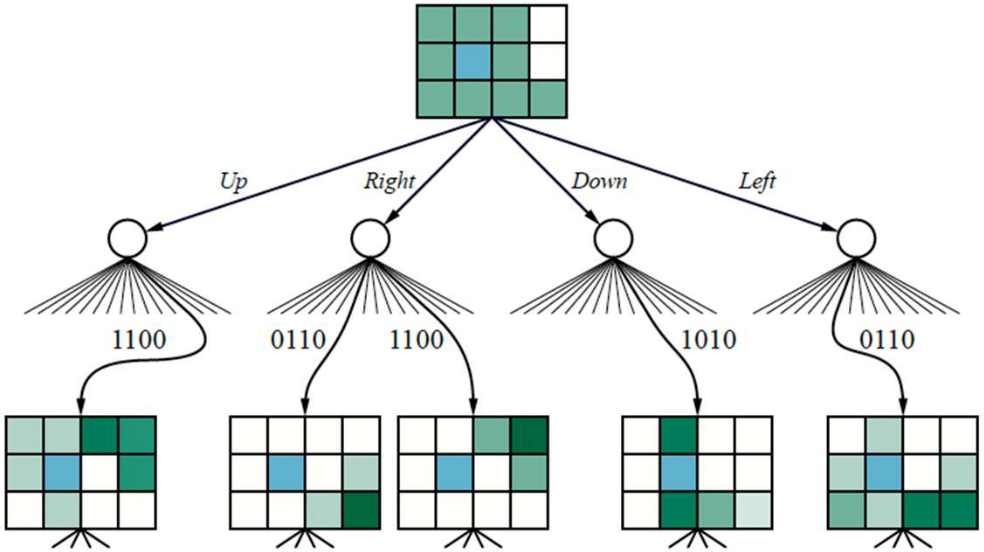


Figure 16.17 Part of an expectimax tree for the 4×3 POMDP with a uniform initial belief state. The belief states are depicted with shading proportional to the probability of being in each location.

<출처> Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson

Summary

1. **Sequential decision problems** in uncertain environments, also called **Markov decision processes**, or MDPs, are defined by a transition model specifying the probabilistic outcomes of actions and a **reward function** specifying the reward in each state.
2. The solution of an MDP is a **policy** that associates a decision with every state that the agent might reach. An **optimal policy** maximizes the utility of the state sequences encountered when it is executed.
3. The **value iteration algorithm** for solving MDPs works by iteratively solving the equations relating the utility of each state to those of its neighbors.
4. **Policy iteration** alternates between calculating the utilities of states under the current policy and improving the current policy with respect to the current utilities.
5. A decision-theoretic agent can be constructed for **POMDP** environments. The agent uses a **dynamic decision network** to represent the transition and sensor models, to update its belief state, and to project forward possible action sequences.