Artificial Intelligence

Markov Decision Processes (AIMA 17)

Christopher Simpkins

Kennesaw State University

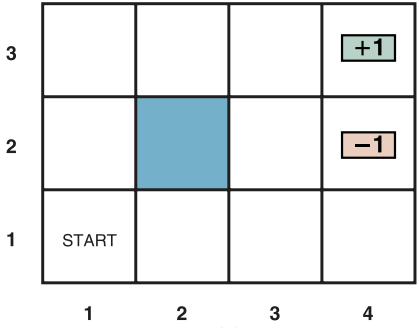
Sequential Decisions

In **sequential decision problems**, the agent's utility depends on a sequence of decisions.

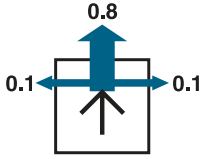
Sequential decision problems incorporate utilities, uncertainty, and sensing, and include search and planning problems as special cases.

- Markov decision processes (MDPs)
- k-Armed bandits
- Partially observable MPDs (POMDPs)

Markov Decision Processes



(a)



(b)

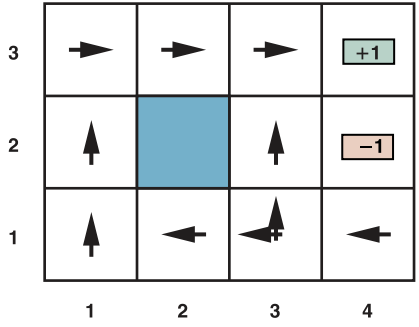
Markov Decision Processes (MDPs)

A Markov decision process (MDP) is a 4-tuple $(S, A, T(s, a, s'), R(s))$, where

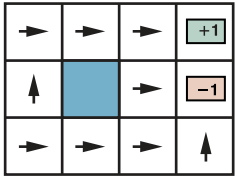
- S is a set of states,
- A, or $Action(s)$ is a set of actions, and
- $T(s, a, s')$, or $Pr(s' \mid s, a)$, is a transition function which gives the probability that executing action a in state s will result in s'.
- $R(s)$ is the reward the world provides to an agent for arriving in state s

Some definitions of MDPs include an initialization function, $I(s)$, which specifies the probability the the agent will start in some state $s \in S$, others specify a particular state from S as the start state.

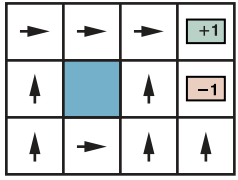
Markov Decision Processes



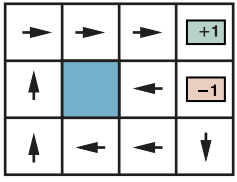
Markov Decision Processes



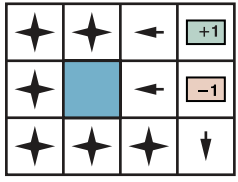
$r < -1.6497$



$-0.7311 < r < -0.4526$

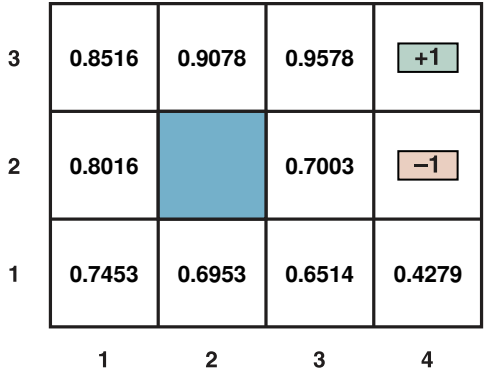


$-0.0274 < r < 0$

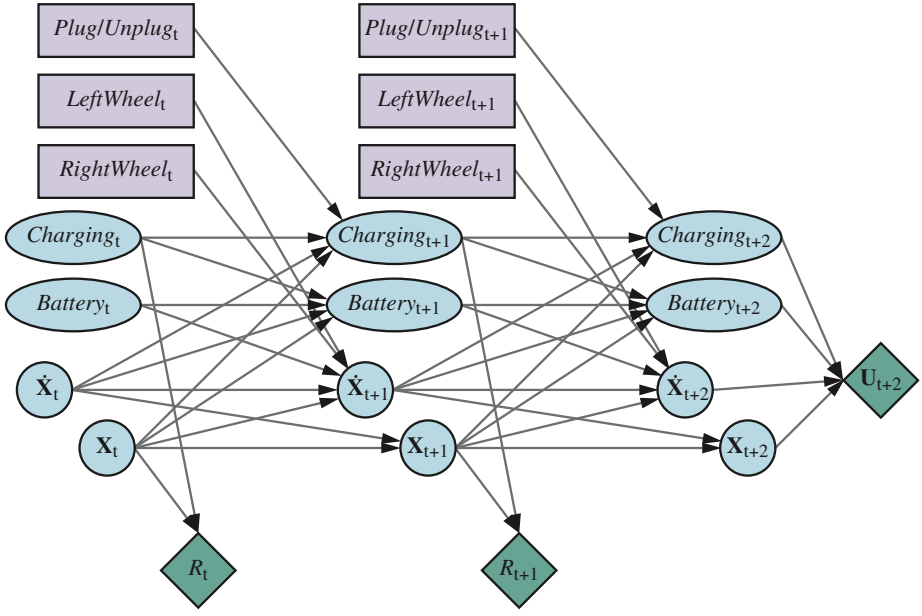


$r > 0$

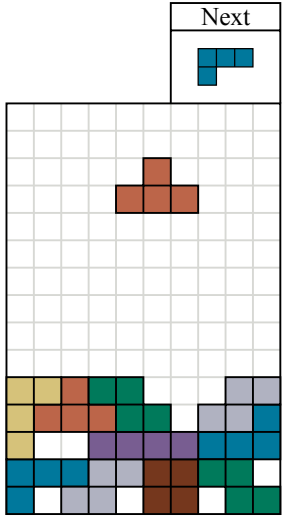
Markov Decision Processes



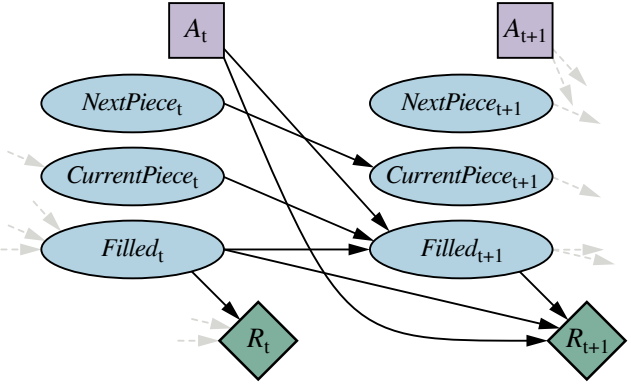
Markov Decision Processes



Markov Decision Processes



Markov Decision Processes



Bellman Optimality Equation

$$V(s) = R(s) + \max_{a \in A} \sum_{s'} T(s, a, s') V(s') \tag{1}$$

This equation is called the Bellman optimality equation [**?**, **?**]. There is one Bellman equation for each state – n equations in n unknowns (the values) for a state space of size n. However, since the max operator is nonlinear we cannot solve the system of simultaneous Bellman equations using linear algebra. One solution is to use an iterative dynamic programming approach: value iteration.

Bellman Value Update Rule

The value iteration algorithm initializes each state's value to a random value, then iteratively update these values by turning the Bellman equation into an update rule (the Bellman update):

$$V_{i+1}(s) \leftarrow R(s) + \max_{a \in A} \sum_{s'} T(s, a, s') V_i(s') \tag{2}$$

These updates are applied at the same time for all states, i.e., the values in iteration $i + 1$ are calculated from the values in iteration i. The value iteration algorithm is shown in Algorithm 1.

Algorithm 1 Value Iteration

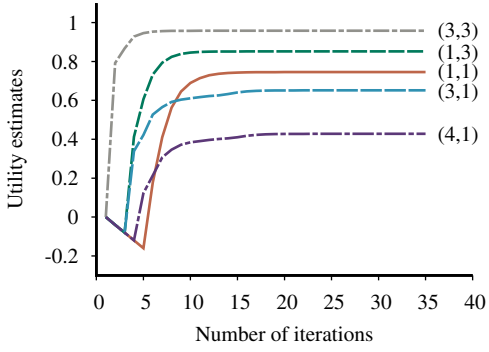
$V \leftarrow$ random initial values
repeat
  $V' \leftarrow V$
  for each $s \in S$ **do**
    $V'(s) \leftarrow R(s) + \max_{a \in A} \sum_{s'} T(s, a, s') V(s')$
  $V \leftarrow V'$
until V changes by a sufficiently small amount

Value Iteration Algorithm

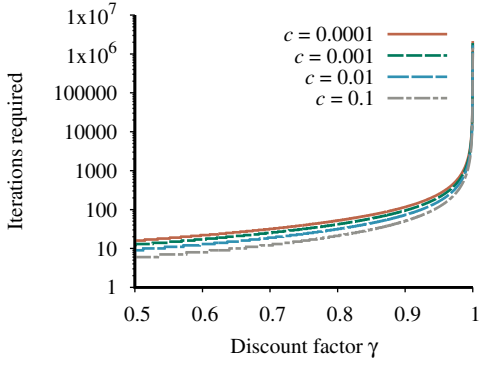
function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s,a)$,
    rewards $R(s,a,s')$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states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

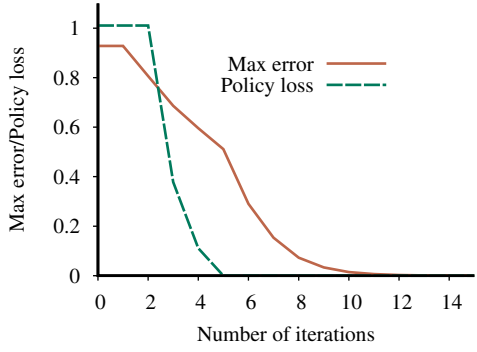
Markov Decision Processes



Markov Decision Processes



Markov Decision Processes



Policy Iteration

In policy iteration [**?**] we start with a random initial values and policy and alternate between two steps for each iteration i:

- **Policy evaluation.** Use policy π_i to calculate the values of each state using the discounted current values of their successor states. Since we are calculating the values under a particular policy, we drop the $\max$ operator:

$$V_{i+1}(s) = R(s) + \gamma \sum_{s'} T(s, a, s')V(s') \tag{3}$$

- **Policy improvement.** Calculate policy p_{i+1} using the values calculated in the previous step.

When policy improvement does not change the policy, an optimal policy has been found and policy iteration terminates.

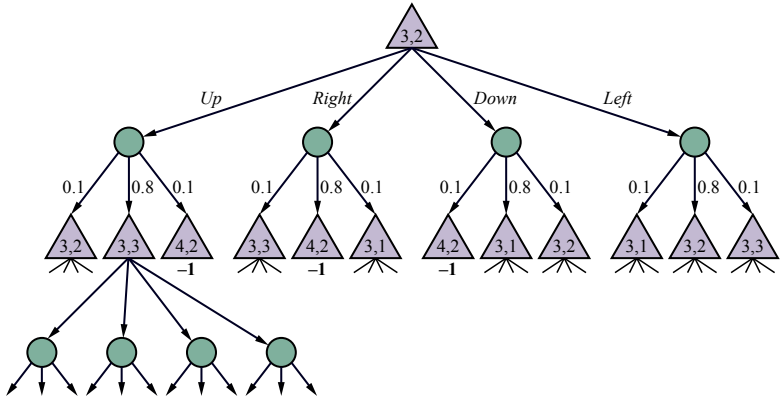
Note that since the update equation used in policy evaluation is linear, we can use linear algebra to solve the set of simultaneous linear equations in $O(n^3)$. This method works fine for smaller state spaces but may be too expensive for large state spaces. A solution to this problem is known as modified policy iteration [**?**, **?**], which combines policy iteration with value iteration by using a bounded number of Bellman updates to perform the policy evaluation step.

Markov Decision Processes

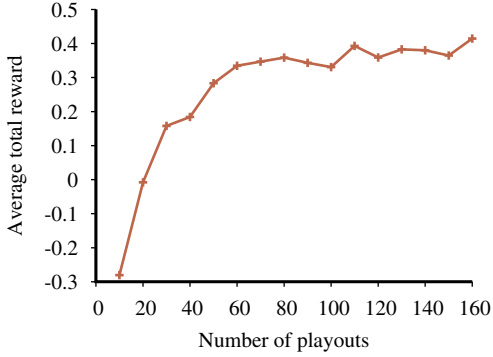
function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      $a^* \leftarrow \underset{a \in A(s)}{\text{argmax}}$ Q-VALUE(mdp, s, a, U)
      if Q-VALUE(mdp, s, a^*, U) $>$ Q-VALUE($mdp, s, \pi[s], U$) **then**
        $\pi[s] \leftarrow a^*$; $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

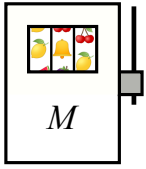
Markov Decision Processes



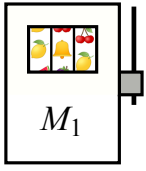
Markov Decision Processes



Markov Decision Processes

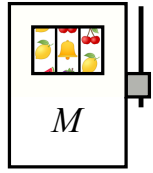


0, 2, 0, 7.2, 0, 0, 0, …

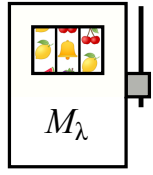


1, 1, 1, 1, 1, 1, 1, …

(a)



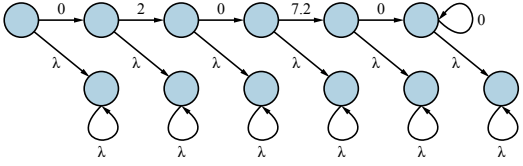
$R_0, R_1, R_2, R_3, R_4, \ldots$



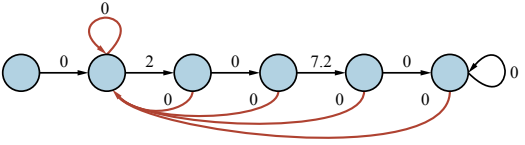
$\lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \ldots$

(b)

Markov Decision Processes

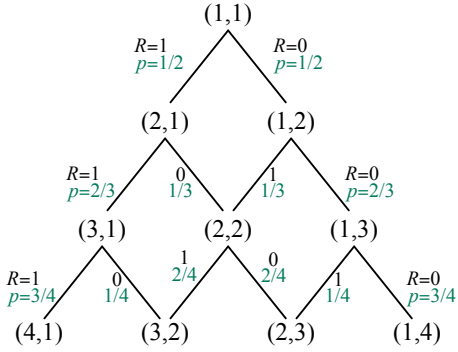


(a)

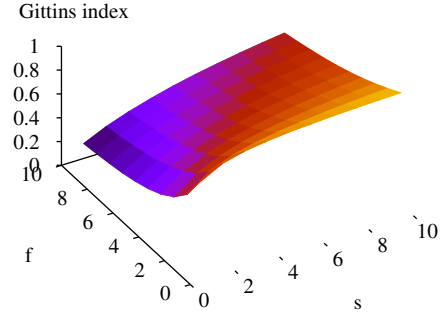


(b)

Markov Decision Processes

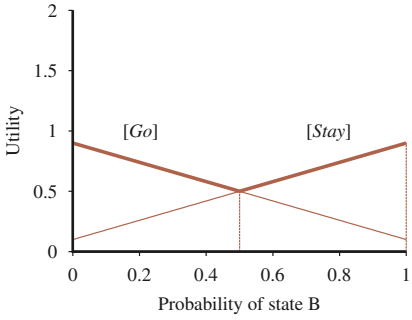


(a)

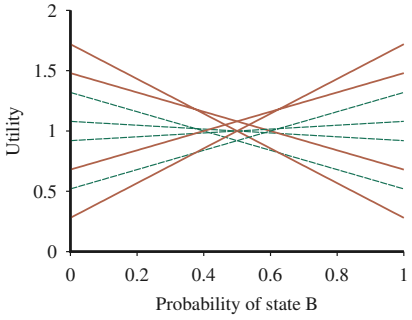


(b)

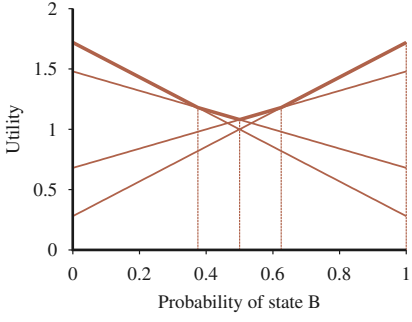
Markov Decision Processes

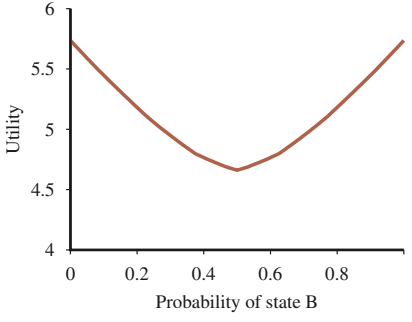


(a)



(b)



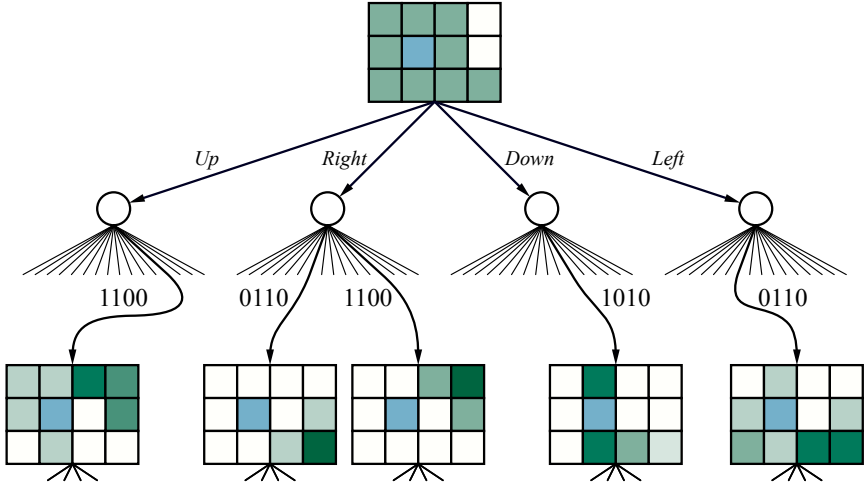


Markov Decision Processes

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 $R(s,a,s')$, 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

Markov Decision Processes



Markov Decision Processes