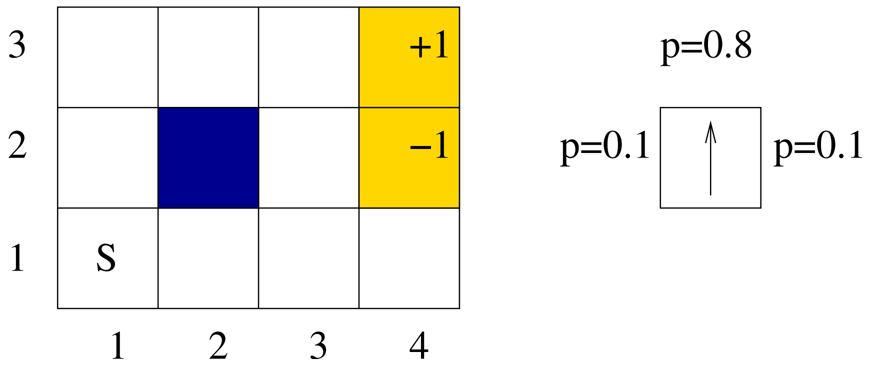
Making Complex Decisions

Chapter 17

Outline

- Sequential decision problems
- Value iteration algorithm
- Policy iteration algorithm

A simple environment



A simple environment (cont’d)

- The agent has to make a series of decisions (or alternatively it has to know what to do in each of the possible 11 states)
- The move action can fail
- Each state has a “reward”

What is different?

- uncertainty
- rewards for states (not just good/bad states)
- a series of decisions (not just one)

Issues

- How to represent the environment?
- How to automate the decision making process?
- How to make useful simplifying assumptions?

Markov decision process (MDP)

- It is a specification of a sequential decision problem for a fully observable environment.
- It has three components
  - S_0: the initial state
  - $T(s,a,s')$: the transition model
  - $R(s)$: the reward function
- The rewards are *additive*.

Transition model

- It is a specification of outcome probabilities for each state and action pair
- $T(s,a,s')$ denotes the probability of ending up in state s' if action a is applied in state s
- The transitions are *Markovian*: $T(s,a,s')$ depends only on s, not on the history of earlier states

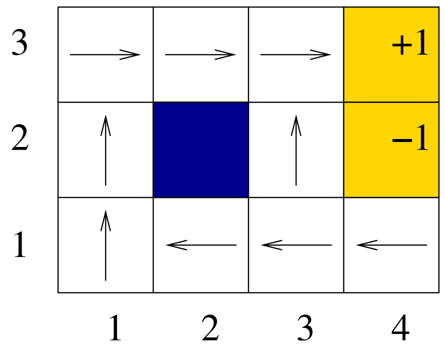
Utility function

- It is a specification of agent preferences
- The utility function will depend on a sequence of states (this is a sequential decision problem, but still the transitions are Markovian)
- There is a negative/positive finite reward for each state given by R(s)

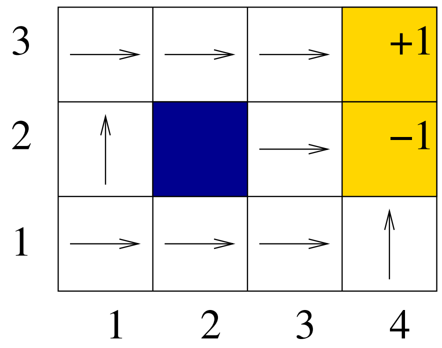
Policy

- It is a specification of a solution for an MDP
- It denotes what to do at any state the agent might reach
- $\pi(s)$ denotes the action recommended by the policy π for state s
- The quality of a policy is measured by the expected utility of the possible environment histories generated by that policy
- π^* denotes the optimal policy
- An agent with a complete policy is a reflex agent

Optimal policy when R(s) = -0.04



Optimal policy when R(s) < -1.6284



Optimal policy when -0.4278 < R(s) < -0.0850

3	→	→	→	+1
2	↑		↑	−1
1	↑	→	↑	←
	1	2	3	4

Optimal policy when -0.0221 < R(s) < 0

3	→	→	→	+1
2	↑		←	−1
1	↑	←	←	↓
	1	2	3	4

Optimal policy when R(s) > 0

3	*	*	←	+1
2	*		←	−1
1	*	*	*	↓
	1	2	3	4

Finite vs. infinite horizon

- A *finite horizon* means that there is fixed time N after which nothing matters (the game is over)
  - $\forall k \geq 0\, U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N])$
  - The optimal policy for a finite horizon is *nonstationary*, i.e., it could change over time
- An *infinite horizon* means that there is no deadline
  - There is no reason to behave differently in the same state at different times, i.e., the optimal policy is *stationary*
  - It is easier than the nonstationary case

Stationary preferences

- It means that the agent's preferences between state sequences do not depend on time
- If two state sequences $[s_0, s_1, s_2, \ldots]$ and $[s'_0, s'_1, s'_2, \ldots]$ begin with the same state (i.e., $s_0 = s'_0$) then the two sequences should be preference-ordered the same way as the sequences $[s_1, s_2, \ldots]$ and $[s'_1, s'_2, \ldots]$

Algorithms to solve MDPs

- Value iteration
  - Initialize the value of each state to its immediate reward
  - Iterate to calculate values considering sequential rewards
  - For each state, select the action with the maximum expected utility
- Policy iteration
  - Get an initial policy
  - Evaluate the policy to find the utility of each state
  - Modify the policy by selecting actions that increase the utility of a state. If changes occurred, go to the previous step

Value Iteration Algorithm

function VALUE-ITERATION (*mdp*, ε)
returns a utility function
  inputs:
    mdp, an MDP with states S, transition model T, reward function R, discount γ
    ε, the maximum error allowed in the utility of a state
  local variables:
    U, U', vectors of utilities for states in S, initially zero
    δ, 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
    $U'[s] \leftarrow R[s] + \gamma max_a \sum_{s'} T(s,a,s')U[s']$
    if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \varepsilon(1-\gamma)/\gamma$
return U

State utilities with $\gamma = 1$ and R(s) = -0.04

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Optimal policy using value iteration

To find the optimal policy choose the action that maximizes the expected utility of the subsequent state

$$\pi^*(s) = argmax_a \sum_{s'} T(s,a,s')U(s')$$

Properties of value iteration

- The value iteration algorithm can be thought of as propogating information through the state space by means of local updates
- It converges to the correct utilities
- We can bound the error in the utility estimates if we stop after a finite number of iterations, and we can bound the policy loss that results from executing the corresponding MEU policy

More on value iteration

The value iteration algorithm we looked at is solving the standard *Bellman equations* using *Bellman updates*.
Bellman equation

$$U(s) = R(s) + \gamma\, max_a \sum_{s'} T(s,a,s')U(s')$$

Bellman update

$$U_{i+1}(s) = R(s) + \gamma\, max_a \sum_{s'} T(s,a,s')U_i(s')$$

More on value iteration (cont'd)

If we apply the Bellman update infinitely often, we are guaranteed to reach an equilibrium, in which case the final utility values must be solutions to the Bellman equations. In fact, they are also the unique solutions, and the corresponding policy is optimal.

Policy iteration

- With Bellman equations, we either need to solve a nonlinear set of equations or we need to use an iterative method
- Policy iteration starts with a initial policy and performs iterations of evaluation and impovement on it

Policy Iteration Algorithm

function POLICY-ITERATION (*mdp*)
returns a policy
  inputs:
    mdp, an MDP with states S, transition model T
  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
      if $\max_a \sum_{s'} T(s,a,s')U[s'] > \sum_{s'} T(s,\pi(s),s')U[s']$ **then**
        $\pi(s) \leftarrow argmax_a \sum_{s'} T(s,a,s')U[s']$
        unchanged? $\leftarrow$ false
  until *unchanged?*
  return π

Properties of Policy Iteration

- Implementing the POLICY-EVALUATION routine is simpler than solving the standard Bellman equations because the action in each state is fixed by the policy
- The simplified Bellman equation is

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s')U_i(s')$$

Properties of Policy Iteration (cont'd)

- The simplified set of Bellman Equations is linear (n equations with n unknowns can be solved in $O(n^3)$ time)
- If n^3 is prohibitive, we can use *modified policy iteration* which uses the simplified Bellman update k times

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s')U_i(s')$$

Issues revisited (and summary)

- How to represent the environment?
  (transition model)
- How to automate the decision making process?
  (Policy iteration and value iteration)
  Can also use *asynchronous policy iteration* and work on a subset of states
- How to make useful simplifying assumptions?
  (Full observability, stationary policy, infinite horizon etc.)