Sequential Decision Making

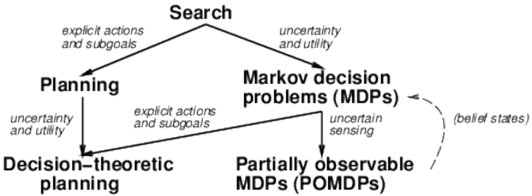
AIMA Chapters: 17.1, 17.2, 17.3.

Sutton and Barto, Reinforcement Learning: an Introduction, 2nd Edition: Chapters 3 and 4

Outline

♢ Sequential decision problems
♢ Value iteration
♢ Policy iteration
♢ POMDPs (basic concepts)
♢ Slides partially based on the Book "Reinforcement Learning: an introduction" by Sutton and Barto
♢ Thanks to Prof. George Chalkiadakis for providing some of the slides.

Sequential decision problems



Sequential decisions

Decisions are rarely taken in isolation, we have to decide on sequences of actions.

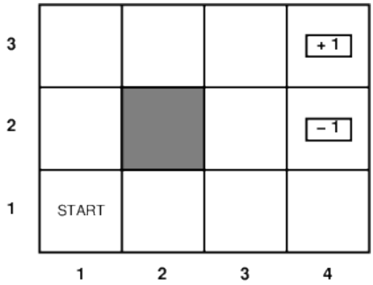
- to enroll in a course students should have an idea of what job they would like to do.

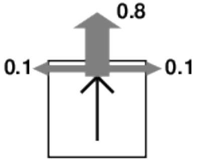
The value of an action goes beyond the immediate benefit (aka reward)

- Long term utility/opportunities: student goes to a lesson not only because he/she enjoys the lecture but also to pass the exam...
- Acquire information: student follows the first lesson to know how the exam modalities will be

Need a sound framework to make sequential decisions and face uncertainty!

Example problem: exploring a maze





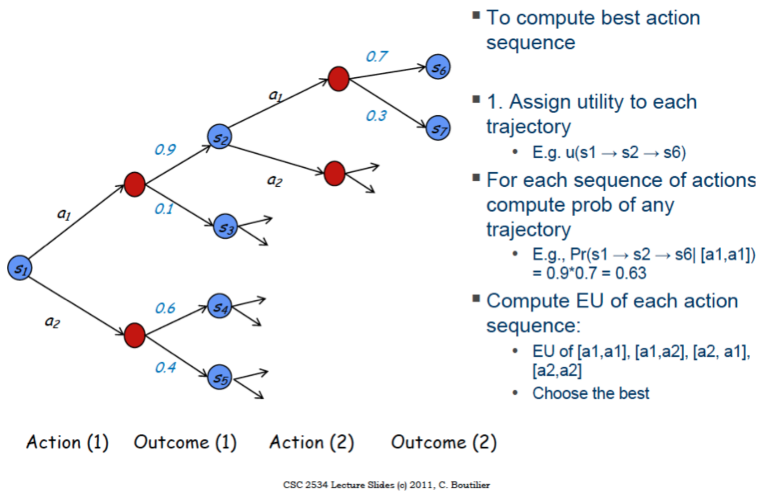
States $s \in S$, actions $a \in A$

<u>Model</u> $T(s, a, s') \equiv P(s'|s, a) =$ probability that a in s leads to s'

<u>Reward function</u> $R(s)$ (or $R(s, a)$, $R(s, a, s')$)

$$= \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

A simple approach



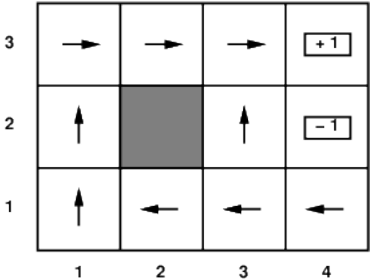
Example: computing the value for a sequence of actions in the maze scenario.

Issues with this approach

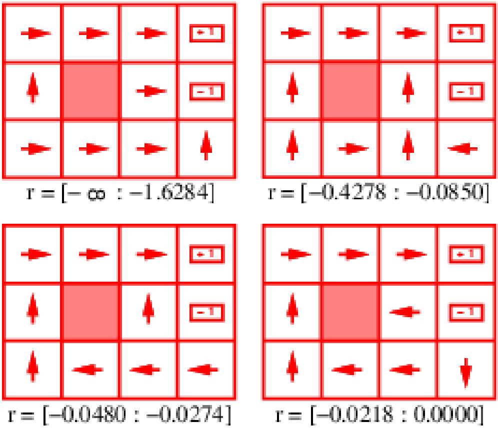
- conceptual: evaluating all sequence of actions **without considering real outcome** is not the right thing to do:
  - It may be better to do a_1 again if I end up to s_2, but best to do a_2 if I end up at s_3
- practical: utility for a sequence is typically harder to estimate than utility of single states
- computational: k actions, t stages, n outcomes per action: $k^t n^t$ possible trajectories to evaluate

The need for policies

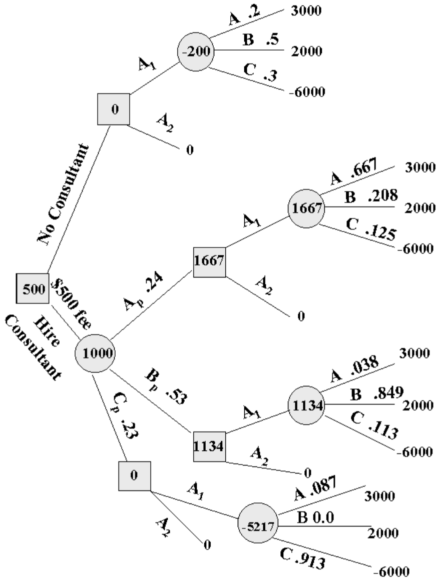
In search problems, aim is to find an optimal sequence
Considering uncertainty, aim is to find an optimal policy $\pi(s)$
  i.e., best action for every possible state s
  (because can't predict where one will end up)
The optimal policy maximizes (say) the expected sum of rewards
Optimal policy when state penalty $R(s)$ is -0.04:



Risk and reward



Decision trees



Solving a decision tree

- Backward induction/rollback (a.k.a. expectimax)
  - Main idea: start from leaves and use MEU
- Value of a leaf node C is given :
  $EU(C) = V(C)$
- Value of a chance node, not leaf (i.e., circles) C :
  $EU(C) = \sum_{D \in \text{Child}(C)} Pr(D)EU(D)$
- Value of a decision node (i.e., squares) C :
  $EU(D) = \max_{C \in \text{Child}(D)} EU(C)$
- Policy: maximise utility of decision node:
  $\pi(D) = \arg\max_{C \in \text{Child}(D)} EU(C)$

Markov Decision Processes

- MDPs: a general class of non-deterministic search problem
  - more compact than decision trees.
- Four components: $\langle S, A, R, Pr \rangle$
- S a (finite) set of states ($|S| = n$)
- A a (finite) set of actions ($|A| = m$)
- Transition function
  $p(s'|s,a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\}$
- Real valued reward function
  $r(s', a, s) = \mathbb{E}[R_{t+1} | S_{t+1} = s', A_t = a, S_t = s]$

Why Markov ?

- Andrey Markov (1856-1922)

- Markov Chain: given current state future is independent from the past
- In MDPs past actions/states are irrelevant when taking decision in a given state.

Markov Property and other assumptions

- Markov Dynamics (history independence)

$Pr\{R_{t+1}, S_{t+1} | S_0, A_0, R_1, \cdots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$

Markov property:

$Pr\{R_{t+1}, S_{t+1} | S_t, A_t\}$

- Stationary (not dependent on time)

$Pr\{R_{t+1}, S_{t+1} | S_t, A_t\} = Pr\{R_{t'+1}, S_{t'+1} | S_{t'}, A_{t'}\} \forall\, t, t'$

- Full observability: we can not predict exactly which state we will reach but we know where we are

MDP: recycling robot

Possible actions:

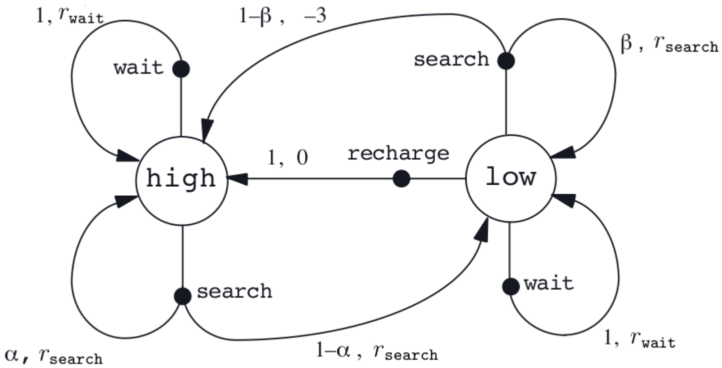
- search for a can (high chance, may run out of battery)
- wait for someone to bring a can (low chance, no battery depletion)
- go home to recharge its battery

Agent decides based on battery level $\{low, high\}$

Action set considering states:

- $A(high) = \{search, wait\}$
- $A(low) = \{search, wait, recharge\}$

Recycling robot, transition graph



α = probability of maintaining a high battery level when performing a search action
β = probability of maintaining a low battery level when performing a search action

Policies

- Non-stationary policy
  - $\pi : S \times T \rightarrow A$
  - $\pi(s, t)$ action at state s with t states to go.
- Stationary policy
  - $\pi : S \rightarrow A$
  - $\pi(s)$ action for state s (regardless of time)
- Stochastic policy
  - $\pi(a|s)$ probability of choosing action a in state s

Utility of state sequences

Need to understand preferences between sequences of states
Typically consider stationary preferences on reward sequences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots] \Leftrightarrow [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

Theorem: there are only two ways to combine rewards over time.

1) Additive utility function:
$U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$

2) Discounted utility function:
$U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$
where γ is the discount factor

Value of a Policy

- How good is a policy ? How do we measure accumulated reward ?
- Value function $V : S \to \Re$
  - Associates a value considering accumulated rewards
- $v_\pi(s)$ denotes value of policy π for state s
  - expected accumulated reward over horizon of interest

Dealing with infinite utilities

- Problem: infinite state sequences (infinite horizon problems) have infinite accumulated rewards
- Solutions:
  - Choose a finite horizon
    - Terminate episodes after a fixed T steps
    - Produces non-stationary policies
  - Absorbing states: guarantee that for every policy a terminal state will eventually be reached
  - Use discounting: $\forall\, 0 < \gamma < 1$
    - $U([r_0, \cdots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\max}}{1-\gamma}$

More on discounting

- smaller $\gamma \rightarrow$ shorter horizons
- Better sooner than later: sooner rewards have higher utility than later rewards
- Example: $\gamma = 0.5$
  - $U([r_1 = 1, r_2 = 2, r_3 = 3]) = 1*1+0.5*2+0.25*3 = 2.375$
  - $U([1, 2, 3]) = 2.375 < U([3, 2, 1]) = 4.125$

Common formulation of value

- Finite horizon T = total expected reward given π
- Infinite horizon, discounted: sum of accumulated discounted rewards given π.
- Also: average reward per time step
- Example: effect of discounting in a linear maze.

Solving MDPs

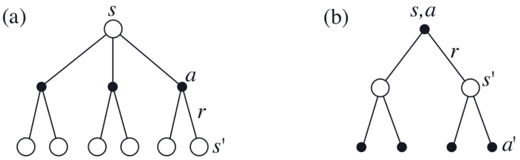
- what is an optimal plan, or sequence of actions?
- MDPs: we want an optimal policy $\pi^* : S \rightarrow A$
- An optimal policy maximizes expected utility if followed:
  - Defines a reflex agent

Values and Q-Values

- Value of a state s when following policy π: expected accumulated (discounted) reward when starting at s and following π everafter
  - $v_\pi(s) = \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$
- Q-value (action value or quality function): value of taking action a in state s following policy π
  - $q_\pi(s, a) = \sum_{s'} p(s'|a, s)(r(s, a, s') + \gamma v_\pi(s'))$
  - Note: $v_\pi(s) = q_\pi(s, \pi(s))$

Bellman equations for policy value

- value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.
- $v_\pi(s) = \sum_{s'} p(s'|\pi(s), s)(r(s, \pi(s), s') + \gamma v_\pi(s'))$
- can be considered as a self-consistency condition



Back-up diagrams for v_π and q_π

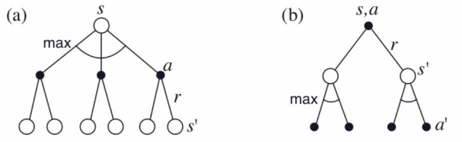
Example: Bellman update for given policy on simple linear maze.

Optimal policy

- $\pi_*(s)$ is an optimal policy iff $v_{\pi_*}(s) \geq v_\pi(s) \forall s, \pi$
- $v_*(s) = \max_\pi v_\pi(s)$ expected utility starting in s and acting optimally everafter
- optimal action-value function $q_*(a, s) = \max_\pi q_\pi(s, a)$
- Example: optimal policy for the maze scenario varying the rewards.

Bellman optimality equation

- $v_*(s)$ must comply with the self-consistency condition dictated by the Bellman equation
- $v_*(s)$ is the optimal value hence the consistency condition can be written in a special form
- The value of a state under an optimal policy must equal the expected return for the best action from that state
- $v_*(s) = \max_{a \in \mathcal{A}(s)} q_*(s,a) = \max_{a \in \mathcal{A}(s)} \sum_{s'} p(s'|a,s)(r(s,a,s') + \gamma v_*(s'))$
- Note: $\mathcal{A}(s)$: actions that can be performed in state s.



Back-up diagrams for v_* and q_*

Value iteration

- **Idea**: turn the Bellman optimality equation into an "update rule", combining policy evaluation (computing the value v_π of a given policy) and policy improvement (making π greedy with respect to v_π).
- the resulting method, Value Iteration, is a successive approximation, Dynamic Programming algorithm.
- Basic DP step: back-up state evaluations to solve the recurrence relations.

Value iteration: Bellman backup

Bellman backup:
$v^{k+1}(s) = \max_a \sum_{s'} p(s'|a, s)(r(a, s, s') + \gamma v^k(s'))$

- Back up the value of every state to produce new ($k + 1$ stage) value function estimates
- The optimality solution of $k + 1$ stage uses the solution to stage k problem

Value iteration: Algorithm

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

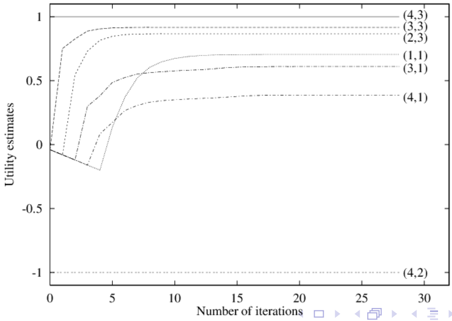
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$

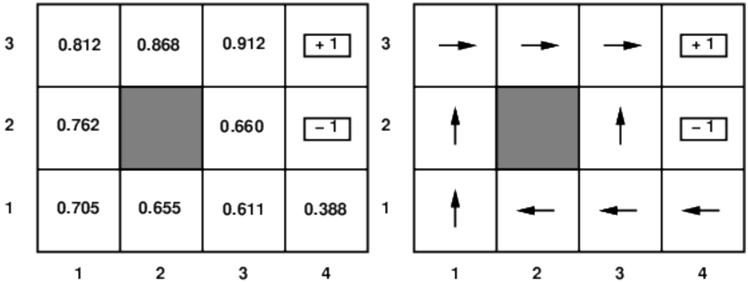
Value iteration: exploring a maze

Example of bellman back-up

$$v(1,1) = -0.04$$
$$+ \gamma \max\{0.8v(1,2) + 0.1v(2,1) + 0.1v(1,1), \quad \text{up}$$
$$0.9v(1,1) + 0.1v(1,2) \quad \text{left}$$
$$0.9v(1,1) + 0.1v(2,1) \quad \text{down}$$
$$0.8v(2,1) + 0.1v(1,2) + 0.1v(1,1)\} \quad \text{right}$$



Value iteration: exploring a maze



Policy is a greedy selection of best action for every state considering the MPDs dynamics

See policy for state $(3, 1)$, $\pi^*((3, 1)) =$ left but state with highest value is up.

Value iteration: discussion

- Value iteration is guaranteed to converge to the optimal value function
  - convergence can be guaranteed also for asynchronous versions (i.e., no need to do a systematic sweep of states) as long as updates of each states are done infinitely often.
- The infinite horizon optimal policy is stationary: optimal action at a state is the same at all times (efficient to store).
- Complexity per iteration is quadratic in the number of states and linear in the number of actions.
- Convergence rate is linear.

Policy iteration

Howard, 1960: search for optimal policy and utility values simultaneously

Algorithm:

$\pi \leftarrow$ an arbitrary initial policy

repeat until no change in π

compute utilities given π (policy evaluation)

update π as if utilities were correct (policy improvement)

Policy evaluation step

To compute utilities given a fixed π (policy evaluation):
$v(s) = \sum_{s'} p(s'|s, \pi(s))(r(s, \pi(s), s') + \gamma v(s'))$
Can be performed:

- by solving n simultaneous linear equations in n unknowns (solve in $O(n^3)$)
- iterative approximation

Policy improvement step

Given the value of all state ($v(s)$)

- greedily change the first action taken when in a state based on current value of states
- if the value of the state can be improved, the new action is adopted by the policy; thus, the performance of the policy is strictly improved.

Modified policy iteration

Policy iteration often converges in few iterations, but each is expensive

Idea: use a few steps of value iteration (but with π fixed) starting from the value function produced the last time to produce an approximate policy evaluation step.

Often converges much faster than pure VI or PI

Leads to much more general algorithms where Bellman value updates and Howard policy updates can be performed locally in any order

Policy improvement step

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad\Delta \leftarrow 0$
   $\quad$For each $s \in \mathcal{S}$:
   $\quad\quad temp \leftarrow v(s)$
   $\quad\quad v(s) \leftarrow \sum_{s'} p(s'|s,\pi(s))\Big[r(s,\pi(s),s') + \gamma v(s')\Big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad temp \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$
   $\quad$If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

The algorithm iterates policy evaluation and policy improvements steps until no improvements are possible. The policy is then guaranteed to be optimal.

Partial observability

POMDP has an observation model $O(s, e)$ defining the probability that the agent obtains evidence e when in state s

Agent does not know which state it is in $\implies$ makes no sense to talk about policy $\pi(s)$!!

Theorem (Astrom, 1965): the optimal policy in a POMDP is a function $\pi(b)$ where b is the belief state (probability distribution over states)

Can convert a POMDP into an MDP in belief-state space, where $T(b, a, b')$ is the probability that the new belief state is b' given that the current belief state is b and the agent does a.

Partial observability contd.

Solutions automatically include information-gathering behavior

If there are n states, b is an n-dimensional real-valued vector

$\Longrightarrow$ solving POMDPs is very (actually, PSPACE-) hard!

The real world is a POMDP (with initially unknown T and O)

Summary

$\diamondsuit$ MDPs can tackle planning problem with uncertainty

$\diamondsuit$ "Good" solution algorithms for MDPs (Value and Policy iteration): convergence, optimality, tractable

$\diamondsuit$ POMDPs = MDPs in belief state, represent a much more realistic setting but are intractable

$\diamondsuit$ Example: computing optimal policy for maze scenario.