L13: Markov decision processes: exact offline solution

AIMA4e: Chapter 17.2.1–17.2.3
Kochenderfer: 7.2–7.7 (more detailed alternative to AIMA4e) and 7.8 (not covered in AIMA)

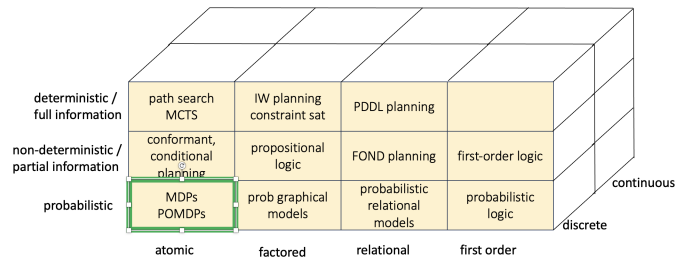
What you should know after this lecture

How to find optimal policies for MDPs!

- Value iteration
- Policy iteration
- Linear programming
- Linear quadratic regulators (for a class of continuous MDPs)

Probabilistic sequential decision-making

Probabilistic transitions
Atomic, discrete states
Full observability
Solution is a policy



- Agent can observe current state completely and correctly
- World dynamics are probabilistic and known to the agent
- Agent selects actions to maximize expected summed rewards over time
- Agent plans on-line to select next action based on current state (but still potentially thinking about a longer horizon)

Recall from last time

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^*(s') \right]$$

$$Q^*(s, a) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

$$Q^*(s, a) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a Q^*(s, a)$$

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a)$$

Solution strategies for MDPs

Two main categories

- Online action selection given current state s_0 via some form of search
- Offline solution to derive a complete policy π that can be executed online very efficiently (next lecture)

Expectimax

Searching to a finite depth:

- Use remaining horizon H for finite and receding horizon problems; set $\gamma = 1$
- For infinite horizon problems, if you search to depth H, the error in your value estimate is bounded by $\gamma^H R_{max}/(1-\gamma)$. Note. [1]

EXPECTIMAX$(s, \mathcal{A}, T, R, \gamma, H)$

```
1  def Q(s, a, H):
2        if H = 0 return 0
3        return Σ_s' T(s, a, s') (R(s, a, s') + γ max_a' Q(s', a', H − 1))
4  return argmax_a Q(s, a, H)
```

Can also extract entire policy tree:

- For any s encountered at level h of expectimax,
- $\pi(s, h) = \text{argmax}_a\, Q(s, a, h)$

[1] Why? Because at any point, $R_{max}/(1-\gamma)$ is the value you'd get if you got reward R_{max} on every step; but this big pile of value would be discounted by γ^d if you get it d steps in the future.

Less stupid expectimax

- Cache all (s, a, H) values computed and re-use when possible
- Don't expand zero-probability branches!
- sparse sampling algorithm: To reduce branching factor, sample $m << |S|$ elements s' from $P(s' \mid s, a)$ and average their values (will focus on the most likely outcomes) instead of doing full expectation over s'.

See also (optionally!) RTDP (real-time dynamic programming)

Stochastic shortest-paths problems

Min-cost path problems : stochastic shortest-path problems :: reward-maximization problems : MDPs

- $\mathcal{S}$: Set of possible world states (possibly infinite)
- $\mathcal{A}$: Set of possible actions (usually finite)
- $T : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$: Transition model maps state and action into probability distribution over next states
- $G \subset \mathcal{S}$: set of goal states

Sometimes also

- $s_0 \in \mathcal{S}$: initial state
- $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: cost function (treat as always = 1 if missing)

Objective is to find a policy that minimizes the expected cost to reach some state in G.

Convert to MDP with: $\gamma = 1$; $R(s, a, s') = 0$ if $s \in G$;
$R(s, a, s') = -C(s, a)$ if $s \notin G$; $T(s, a, s) = 1$ if $s \in G$.

Deterministic, open-loop approximations

More efficient (but also more approximate) strategies. Risk ignoring low probability but very bad outcomes.

- Rely on replanning (as in receding horizon control)
- Build a deterministic search problem and solve it
  - General MDP $\rightarrow$ reward-maximization problem
  - Stochastic shortest-paths problem $\rightarrow$ min-cost path problem
- Two general strategies:
  - Make T deterministic by assuming the most likely outcome will occur
  - Make T deterministic by allowing the search to choose among the outcomes, but assigning an additional cost of $-\log T(s, a, s')$ to the selected transition.

Cool result: if you have a stochastic shortest-paths problem, and you determinize and assign costs $-\log T(s, a, s')$ to the transitions, then the least cost path to a goal state is the open-loop action sequence that is most likely to reach a goal.

Policy evaluation

An important sub-problem: given policy π, what is the value of executing it?

$$V_\pi(s) = \sum_{s'} P(s' \mid s, \pi(s)) \left[R(s, \pi(s), s') + \gamma V_\pi(s') \right]$$

Note that:

- MDP + policy is a Markov chain
- Can define an iterative algorithm

PolicyEvaluation$(\pi, \mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

```
1  V_π(s) = 0 for s ∈ S
2  while True:
3        for s ∈ S:
4              V_π,new(s) = Σ_s' P(s' | s, π(s)) [R(s, π(s), s') + γ V_π(s')]
5        if |V_π − V_π,new| < ε:
6              return V_π,new
7        V_π = V_π,new
```

- Observe that values are defined by a set of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns (the $V_\pi(s)$ values). Let
  - V_π be a vector of the V_π values, for each state
  - T_π be a transition matrix, where $T_{ij} = P(s_j \mid s_i, \pi(s_i))$
  - R be a reward vector (simplified to match standard treatment), where $R_i = R(s_i, \pi(s_i))$

  Then

$$V_\pi = R + \gamma T_\pi V_\pi$$
$$(I - \gamma T_\pi) V_\pi = R$$
$$V_\pi = (I - \gamma T_\pi)^{-1} R$$

Unfortunately, there's not such an easy solution for finding the optimal value function, because the max operations make the system non-linear.

Finding optimal policy: Stupidest possible algorithm

- Given finite s and a, there are finitely many policies! We could enumerate them, but how would we decide which one is best?
- Because of the Markov property (future depends only on s_t), there's a helpful theorem
- For any MDP there exists at least one deterministic optimal policy π^* such that for all other policies π,

$$V_{\pi^*}(s) \geqslant V_{\pi}(s)$$

- This means we don't have to worry that some policies might be good in some parts of the state space and others good in other parts—there is at least one policy that's as good as the best policy at all states!

Finding optimal policy: Value iteration

Not so easy when we don't know the policy! System of equations isn't linear any more (it has $\max$ operations in it.)

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^*(s') \right]$$

ValueIteration$(\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

$V(s) = 0$ for $s \in \mathcal{S}$
while True:
    for $s \in \mathcal{S}$:
        $V_{new}(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V(s') \right]$
    if $|V_\pi - V_{new}| < \epsilon$:
        return V_{new}
    $V = V_{new}$

where $|V_1 - V_2| = \max_s |V_1(s) - V_2(s)|$.

Finding optimal policy: Q-Value iteration

For acting, it is more useful to have the $Q(s, a)$ values.

$$Q^*(s, a) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

QValueIteration$(\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

```
1  Q(s, a) = 0 for s ∈ S, a ∈ A
2  while True:
3      for s ∈ S, a ∈ A:
4          Q_new(s, a) = Σ_s' P(s' | s, a) [R(s, a, s') + γ max_a' Q(s', a')]
5      if |Q − Q_new| < ε:
6          return Q_new
7      Q = Q_new
```

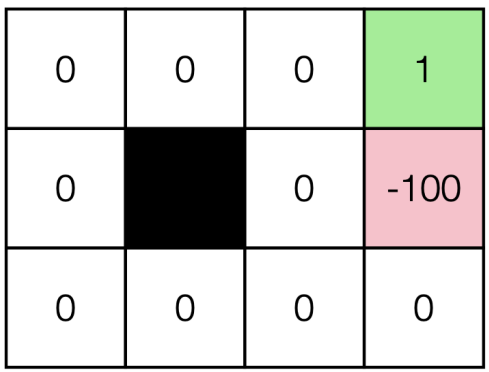
where $|Q_1 - Q_2| = \max_{s,a} |Q_1(s, a) - Q_2(s, a)|$.

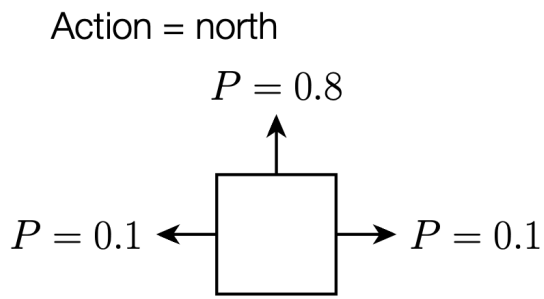
Cool facts about value iteration

- Guaranteed to converge to Q^*
- Max-norm error $|Q - Q^*|$ decreases monotonically per iteration
- Can initialize to any value
- When initialized to 0, iterations are finite-horizon value functions.
- Can execute "in place" (don't need a separate Q_{new})
- Can randomly pick (s, a) to update, rather than doing it systematically
- Serves as the basis for Q-learning
- If $|Q - Q_{new}| < \epsilon$ then $|Q - Q^*| < \epsilon\gamma/(1-\gamma)$
- Define greedy policy with respect to value function $\pi_Q(s) = \mathsf{argmax}_a\, Q(s, a)$. Then if $|Q(s, a) - Q^*(s, a)| < \epsilon$, $|V_{\pi_Q} - V^*| < 2\epsilon$.

Gridworld domain[2]

- Simple grid world with a goal state with reward 1 and a "bad state" with reward -100
- Actions move in the desired direction with probability 0.8, in one of the perpendicular directions with probability 0.1
- Taking an action that would bump into a wall leaves agent where it is.





[2] Example from Zico Kolter, CMU

Gridworld value iteration

Running value iteration with $\gamma = 0.9$

- One iteration

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

- Five iterations

0.809	1.598	2.475	3.745
0.268		0.302	-99.59
0	0.034	0.122	0.004

Gridworld value iteration

Running value iteration with $\gamma = 0.9$

- Ten iterations

2.686	3.527	4.402	5.812
2.021		1.095	-98.82
1.390	0.903	0.738	0.123

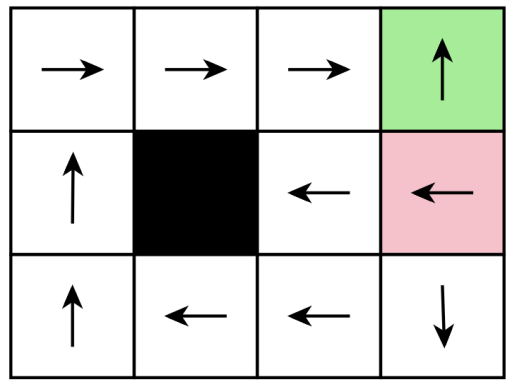
- 1000 iterations

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

Gridworld value iteration

Running policy iteration with $\gamma = 0.9$

- Resulting policy after 1000 iterations



Policy iteration

Actually, it usually happens that $\pi_Q = \pi^*$ long before Q is close to Q^*. So doing value iteration until convergence might be too much work. Let's try working explicitly in the space of policies without enumerating them all!

PolicyIteration$(\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

```
1  π(s) = a for s ∈ S and an arbitrary a ∈ A
2  while True:
3        Q = Q_π                          // Policy evaluation
4        π' = π_Q                         // Greedy policy wrt Q
5        if π = π':
6              return π
7        π = π'
```

- Worst-case complexity is bad, but often very good in practice.
- Interesting combinations of policy and value iteration (e.g., don't completely solve policy evaluation step–use some iterations of iterative policy evaluation instead.)

Gridworld policy iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

- One iteration

0.418	0.884	2.331	6.367
0.367		-8.610	-105.7
-0.168	-4.641	-14.27	-85.05

- Two iterations

5.414	6.248	7.116	8.634
4.753		2.881	-102.7
2.251	1.977	1.849	-8.701

Gridworld policy iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

- Three iterations (converged)

5.470	6.313	7.190	8.669
4.803		3.347	-96.67
4.161	3.654	3.222	1.526

Gridworld results

- Approximation of value function
  - Policy iteration: exact value function after three iterations
  - Value iteration: after 100 iterations, $\|V - V^*\|_2 = 7.1 \times 10^{-4}$
- Calculation of optimal policy
  - Policy iteration: three iterations
  - Value iteration: 12 iterations

In other words, value iteration converges to optimal policy long before it converges to correct value in this MDP (but, this property is highly MDP-specific)

Policy iteration or value iteration?

- Policy iteration requires fewer iterations that value iteration, but each iteration requires solving a linear system instead of just applying Bellman backup.
- In practice, policy iteration is often faster, especially if the transition probabilities are structured (e.g., sparse) to make solution of linear system efficient.
- Modified policy iteration (Puterman and Shin, 1978) solves linear system approximately, using backups very similar to value iteration, and often performs better than either value or policy iteration.

Linear programming

- Define variables V_i for the optimal value of state i.
- Minimize

$$\sum_i V_i$$

- Subject to, for all $s \in |\mathcal{S}|$, $a \in |\mathcal{A}|$

$$V_i \geqslant \sum_{s_j} P(s_j \mid s_i, a) \left[R(s_i, a, s_j) + \gamma V_j\right]$$

Linear programming

- Why do we minimize a weighted combination of the values? Shouldn't we maximize value?
- Value functions V that satisfy the constraints are upper bounds on the optimal value function V^*

$$V(s) \geqslant V^*(s) \quad \forall s$$

- Minimizing value ensures that we choose the lowest upper bound

$$\min_V V(s) = V^*(s) \quad \forall s$$

Only known algorithm with worst-case time complexity polynomial in $|\mathcal{S}|$ and $|\mathcal{A}|$. The complexity of policy iteration has a term that depends on $1/(1-\gamma)$.
In practice other methods are usually more efficient.

Linear quadratic regulator

There is no general-purpose solution method for continuous state and action MDPs. But there's an interesting and useful special case when the dynamics are linear-Gaussian (like the systems we did Kalman filtering on) and the rewards are quadratic.

- $\mathcal{S} = \mathbb{R}^n$ (often called X)
- $\mathcal{A} = \mathbb{R}^m$ (often called controls U)
- $s' \mid s, a = T_s s + T_a a + W$ where T_s is an $n \times n$ matrix, T_a is an $n \times m$ matrix, and W is drawn from a zero-mean finite-variance Gaussian.
- $R(s, a) = s^T R_s s + a^T R_a a$ where R_s is $n \times n$ and positive semidefinite and R_a is $m \times m$ and positive definite. This penalizes states and actions that deviate from 0. You need to define your states s so that 0 is a desired state and actions a so that 0 is a desired action.

Linear quadratic regulator: solutions

Good to know that these exist, but we won't study them:

- In finite-horizon case, there is a value iteration method that finds the exact optimal control sequence, using the dynamic Riccati equation
- Even cooler, in the infinite-horizon (non-discounted) case, there is a stationary optimal policy of the form $a = Fs$ where F is a fixed matrix (found by solving for a fixed-point of the Riccati equation).
- Even more cool, these same basic things work out in continuous time, where $\dot{s} = T_s s + T_a a + W$.

Next time

- Approximate value and policy iteration via reinforcement learning