L18: Markov decision processes: Definition and online solutions

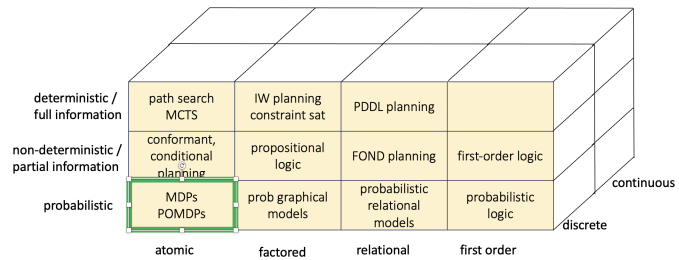
AIMA4e: Chapter 17.1, 17.2.4
More detail: Kochenderfer 9.1, 9.3, 9.5, 9.6, 9.9.1

What you should know after this lecture

- What is an MDP?
- Online solution of MDPs: exact method: expectimax
- Approximate methods:
  - MCTS works here too (with some modifications)
  - Deterministic approximations: most likely outcome, all-outcomes

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)

Markov decision process

- Focus on sequential decision making: agent takes multiple steps
- Reward function maps states (and actions) to real numbers
- Next state depends probabilistically on the state and action
- The current state is exactly known
- Although we don't know, right now, what the next state will be, we will be able to observe it exactly, correctly, when it arises
- Reward function and transition distribution are fixed and known

MDPs: formal definition

- $\mathcal{S}$: 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
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: Reward function defined most generally as a function of state, action, next state

Sometimes also

- $s_0 \in \mathcal{S}$: initial state
- $\gamma \in [0, 1]$: discount factor (defined in a few slides)

MDPs: utility, reward, and action

We want to find a way of selecting actions in an MDP that is optimal in some sense. Maximize expected utility!
Pick the action now that will get us the state sequence with the best utility, in expectation, assuming we pick all future actions optimally as well!

$$a^* = \operatorname*{argmax}_a \mathop{\mathbb{E}}_{s_1,\ldots,s_H} [U(s_1,\ldots,s_H) \mid a_0 = a, \quad \text{optimal future actions}]$$

Assuming that our utility over sequences is additive (big assumption!) and stationary (big assumption!)[1], then

$$a^* = \operatorname*{argmax}_a \mathop{\mathbb{E}}_{s_1,\ldots,s_H} \left[\sum_{t=0}^{H} \gamma R(s_t, a_t, s_{t+1}) \mid a_0 = a, \quad \text{optimal future actions} \right]$$

[1] If your reward is non-stationary, then how much you value something depends on the time. You could potentially address this by putting t in the state.

MDPs: horizon

Different ways to think about how long the agent will be acting

- Finite horizon: H is finite
  - non-stationary policy may select different actions depending on number of steps remaining
- Receding horizon
  - On every step, act as if you have finite H steps remaining
- Indefinite horizon
  - Let $H = \infty$
  - But assume (at least) that the optimal policy has finite $\mathbb{E}_{s_1,\ldots,s_H}\left[\sum_{t=0}^{\infty} r_t\right]$
  - Usually in domains where there are terminal states, and the optimal strategy eventually terminates after a number of steps that is finite in expectation.

MDPs: more horizon

- Infinite-horizon discounted
  - Most typical infinite-horizon model
  - Requires discount factor $0 \leqslant \gamma < 1$
  - Optimize

$$\mathop{\mathbb{E}}_{s_1,\ldots,s_H}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

  - As if process terminates with prob $1-\gamma$ after every step
- Infinite-horizon averaged
  - Useful for problems where the agent is supposed to live forever
  - Optimize

$$\lim_{H\to\infty} \frac{1}{H} \mathop{\mathbb{E}}_{s_1,\ldots,s_H}\left[\sum_{t=0}^{H} r_t\right]$$

We will focus on finite horizon, receding horizon, and infinite-horizon discounted models.

MDPs: policies

We could specify a way of behaving as a policy

$$\pi : \mathcal{S} \times \mathcal{A} \times \ldots \mathcal{S} \times \mathcal{A} \to \mathcal{A}$$

that maps the history of actions and observations into the next action. But! Because in the graphical model for an MDP, s_t separates s_{t+1} from $s_0, a_0, \ldots, s_{t-1}, a_{t-1}$, it is sufficient to base our choice of a_t on s_t only, so we will consider policies that depend only on the current state (and possibly time–more on this later).
In some problems (games, POMDPs) the optimal policy has to be randomized. But,
Theorem: In MDPs, there exist deterministic optimal policies.

$$\pi^*(s_0, H) = \underset{a}{\text{argmax}} \underset{s_1,\ldots,s_H}{\mathbb{E}} \left[\sum_{t=0}^{H} r_t \mid a_0 = a, \text{ optimal future actions} \right]$$

Also true in infinite / indefinite-horizon cases, with no dependence on H in the policy.

Finite-horizon value functions and policies

Utility of being in state s with H steps left to go:[2]

$$V_H^*(s) = \max_a \mathop{\mathbb{E}}_{s_1,\ldots,s_H} \left[\sum_{t=0}^{H} r_t \mid a_0 = a, \quad \text{optimal future actions} \right]$$

$$V_0^*(s) = 0$$

$$V_1^*(s) = \max_a \sum_{s'} P(s' \mid s, a) R(s, a, s')$$

$$V_H^*(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V_{H-1}^*(s') \right]$$

Note that the optimal policy depends on the number of steps left to go!

$$\pi_H^*(s) = \operatorname*{argmax}_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V_{H-1}^*(s') \right]$$

[2] AIMA and KAlg use U for this but every single other text uses V. Well...in operations research they often use costs c instead of rewards, try to minimize the expected sum of costs, and use J instead of V.

Infinite-horizon discounted value functions

- Crucial idea: no matter how long you have been executing so far, you have the same expected number of future steps: $1/(1-\gamma)$.
- So, any state s always has the same optimal expected sum of rewards, no matter how long you have been executing.
- Utility of being in state s with discount factor γ:

$$V^*(s) = \max_a \mathop{\mathbb{E}}_{s_1,\ldots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid a_0 = a, \quad \text{optimal future actions} \right]$$

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

- Utility is finite, bounded by $R_{\max}/(1-\gamma)$ where $R_{\max}$ is the largest value in the range of R.
- Optimal policy is stationary!

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

Q and V

Sometimes it's easier to define values in terms of Q instead of V. No new ideas, really, just bookkeeeping!
Value of starting in state s, taking action a and then continuing optimally:

- Finite horizon:

$$Q^*_H(s,a) = \sum_{s'} P(s' \mid s,a)\left[R(s,a,s') + V^*_{H-1}(s')\right]$$

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

- Infinite horizon:

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

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

Useful because if you know Q, you can derive π without knowing T or R.

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. [3]

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,
- $\pi(s, h) = \text{argmax}_a\, Q(s, a, h)$

[3] 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)

MCTS for MDPs

Okay to ignore details.
Main differences from deterministic version we looked at earlier:

- *children* is dictionary from chance nodes to action
- We count how many times each action has been tried in each state
- A chance node has a set of children that are resulting state-nodes

MCTS$(s_0, (\mathcal{A}, \mathsf{T}, \mathsf{R}, \gamma, \mathsf{H}), \mathit{iters})$

$\mathit{root} = \textsc{StateNode}(s_0, \mathit{horizon} = \mathsf{H}, \mathit{parent} = \textbf{None}, \mathit{children} = \{\}, \mathsf{N} = 0)$
for $\mathit{iter} \in \{1, \ldots, \mathit{iters}\}$:
    $\mathit{leaf} = \textsc{select}(\mathit{root})$
    $\mathit{child} = \textsc{expand}(\mathit{leaf}, \mathcal{A}, \mathsf{T})$
    $\mathit{value} = \textsc{simulate}(\mathit{leaf}, \mathcal{A}, \mathsf{T}, \mathsf{R})$
    $\textsc{backup}(\mathit{leaf}, \mathit{value})$
$\mathit{max\_child} = \mathsf{max}(\mathit{root.children}, \mathit{key} = \lambda n.\ n.\mathsf{N})$
return $\mathit{root.children}[\mathit{max\_child}].\mathit{action}$

Monte-Carlo Tree Search (Cont)

EXPAND$(n, \mathcal{A}, T)$

```
    // Unless remaining horizon is 0, add child chance and state nodes and return one
 1  if n.horizon = 0:
 2      return n
 3  for a ∈ A:
 4      c' = ChanceNode(parent = n, children = { }, U = 0, N = 0)
 5      n.children[c'] = a
 6  c, a = random_choice(n.children)
 7  s' = T(n.s, a)
 8  n' = StateNode(s', n.horizon − 1, parent = c, children = { }, N = 0)
 9  c.children[s'] = n'
10  return
```

SIMULATE$(n, \mathcal{A}, T, R)$

```
    // Randomly finish path and return cumulative reward
1  s = n.state; total_reward = 0
2  for h ∈ (n.horizon, ..., 1):
3      a = random_choice(A) // or use some rollout_policy
4      s' ~ P(s' | s, a)
5      total_reward += R(s, a, s')
6      s = s'
7  return total_reward
```

Monte-Carlo Tree Search (Cont)

```
SELECT(n)
1  c, a = max(n.children, key = λc.UCB(n.N, c.N, c.U))
2  s' ~ P(s' | n.state, c, a)
3  if not s' in c.children:
4        c.children[s'] = StateNode(s', H − 1, parent = c, children = {}, N = 0)
5        return c.children[s']
6  return SELECT(c.children[s'])

BACKUP(n, v_below)
   // Add value v to n's statistics and pass it up
1  n.N += 1
2  if n.parent:
3        a = n.parent.action                                  // Action that led to n
4        v = γ · v_below + R(n.parent.parent.s, a, n.s)  // Value of executing a in parent
5        n.parent.U += v
6        BACKUP(n.parent.parent, v)
```

Monte-Carlo Tree Search (Cont)

There are many strategies for doing backups. Could instead: directly update the U at leaf using value from simulate. But then work back up the path letting

$$U(s, a) += \sum_{s'} R(s, a, s') + \gamma \max_{a'} U(s', a')/N(s', a')$$

BACKUP(n, v_below)

```
    // Add value v to n's statistics and pass it up
1   n.N += 1
2   if n.parent:
3       a = n.parent.action                                  // Action that led to n
4       v = γ · v_below + R(n.parent.parent.s, a, n.s)       // Value of executing a in parent
5       n.parent.U += v
6       BACKUP(n.parent.parent, MAX(n.parent.parent.children, λc. c.U/c.N))
```

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.

Next time

- Finding optimal policies, offline, for MDPs, which can be executed very efficiently online