Reinforcement Learning

Lecture 3: Value Iteration, Policy Iteration, Asynchronized DP

Instructor: Chongjie Zhang

Tsinghua University

Last Week

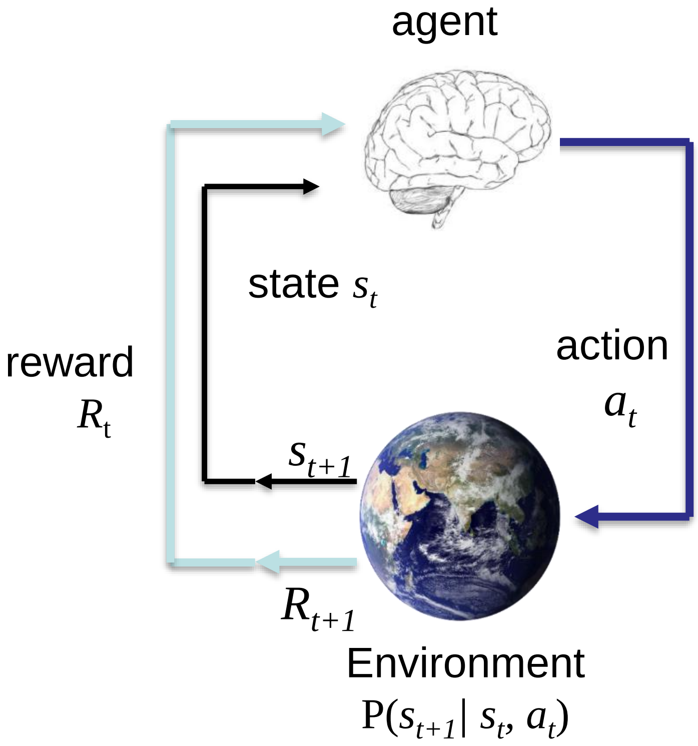
- Definition of Markov Decision Process (MDP)
  - Mathematical model for reinforcement learning
- Finite-Horizon MDPs
  - Solution: non-stationary policy and value function
  - Policy evaluation: backward dynamic programming
  - Policy optimization: backward value iteration
- Infinite-Horizon MDPs
  - Solution: stationary policy and value function
  - Policy evaluation: linear solver
  - Policy optimization: value iteration

Today's Outline

- Value iteration
- Policy iteration
- Asynchronized Dynamic Programming
- Extensions of MDPs

Markov Decision Process (MDP)

- An Finite MDP is defined by:
  - A finite set of states $s \in S$
  - A finite set of actions $a \in A$
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s'| s, a)
    - Also called the model or the dynamics
  - A reward function R(s)   (Sometimes R(s, a) or R(s, a, s'))
  - A start state
  - Maybe a terminal state
- A model for sequential decision making problem under uncertainty



Value Function for Infinite Horizon MDPs

- Discounted expected reward with $0 \leq \gamma < 1$
  - future rewards discounted by γ per time step

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R^t \mid \pi, s\right]$$

- Why?
  - Mathematically convenient to discount rewards
  - Avoids infinite returns in cyclic Markov processes
  - Uncertainty about the future may not be fully represented
  - If the reward is financial, immediate rewards may earn more interest than delayed rewards
  - Animal/human behavior shows preference for immediate reward

Computing the Optimal Policy

- How to compute the optimal policy? (or equivalently, the optimal value function?)

- Approach #1: Value Iteration
  - Initialize an estimate for the value function arbitrarily

$$\hat{V}(s) \leftarrow 0, \quad \forall s \in \mathcal{S}$$

  - Repeatedly update the estimate according to Bellman optimality equation

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s'), \quad \forall s \in \mathcal{S}$$

Convergence of value iteration

Theorem: Value iteration converges to optimal value: $\hat{V} \to V^\star$

Convergence of value iteration

Theorem: Value iteration converges to optimal value: $\hat{V} \rightarrow V^{\star}$

Convergence of value iteration

Theorem: Value iteration converges to optimal value: $\hat{V} \to V^\star$

Proof: For any estimate of the value function $\hat{V}$, we define the Bellman backup operator $B : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$

$$B\,\hat{V}(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\,\hat{V}(s')$$

We will show that Bellman operator is a *contraction*, that for any value function estimates $V_1,\ V_2$

$$\max_{s \in \mathcal{S}} |B V_1(s) - B V_2(s)| \leq \gamma \max_{s \in \mathcal{S}} |V_1(s) - V_2(s)|$$

Since $BV^\star = V^\star$ (the contraction property also implies existence and uniqueness of this fixed point), we have:

$$\max_{s \in \mathcal{S}} \left| B\,\hat{V}(s) - V^\star(s) \right| \leq \gamma \max_{s \in \mathcal{S}} \left| \hat{V}(s) - V^\star(s) \right| \implies \hat{V} \to V^\star$$

Convergence of value iteration

Proof of contraction property:

$$
\begin{aligned}
&|BV_1(s) - BV_2(s)| \\
&= \gamma \left| \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V_1(s') - \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V_2(s') \right| \\
&\leq \gamma \max_{a \in \mathcal{A}} \left| \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V_1(s') - \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V_2(s') \right| \\
&= \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\, |V_1(s') - V_2(s')| \quad \leq \gamma \max_{a} \sum_{s'} P(s'|s,a) \max_{s} |V_1(s) - V_2(s)| \\
&\leq \gamma \max_{s \in \mathcal{S}} |V_1(s) - V_2(s)|
\end{aligned}
$$

where third line follows from property that

$$
|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)|
$$

and final line because $P(s'|s,a)$ are non-negative and sum to one

Value iteration convergence rate

How many iterations will it take to find optimal policy?

Assume rewards in $[0, R_{\max}]$, then

$$V^\star(s) \leq \sum_{t=1}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma}$$

Then letting V^k be value after kth iteration

$$\max_{s \in \mathcal{S}} |V^k(s) - V^\star(s)| \leq \frac{\gamma^k R_{\max}}{1-\gamma}$$

i.e., we have linear convergence to optimal value function

But, time to find optimal policy depends on separation between value of optimal and second suboptimal policy, difficult to bound

Stopping Condition

- Want to stop when we can guarantee the value function is near optimal.
- Key property:

$$\text{If } \|V^k - V^{k-1}\| \leq \varepsilon \text{ then } \|V^k - V^*\| \leq \varepsilon\gamma / (1-\gamma)$$

- Continue iteration until $\|V^k - V^{k-1}\| \leq \varepsilon$
  - Select small enough ε for desired error guarantee

How to Act

- Given a V^k that closely approximates V^*, what should we use as our policy?
- Use *greedy* policy: (one step lookahead)

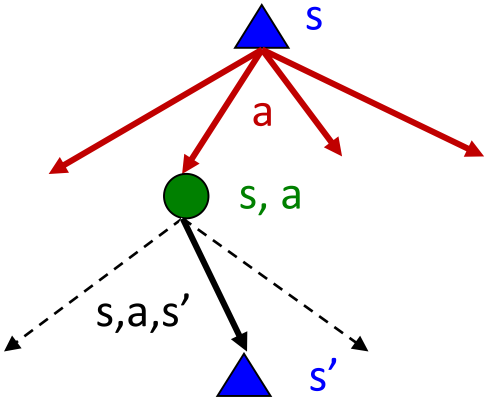
$$greedy[V^k](s) = \underset{a}{\arg\max} \sum_{s'} T(s,a,s') \cdot V^k(s')$$

- This selects the action that looks best if we assume that we get value V^k in one step
- How good is this policy?

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\,\hat{V}(s'), \quad \forall s \in \mathcal{S}$$



- Problem 1: It's slow – $O(S^2A)$ per iteration

- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

Approach #2: Policy Iteration

- Another approach to computing the optimal policy

- Algorithm:
  1. Initialize policy π (e.g., randomly)
  2. Compute the value V^{π} of policy π
  3. Update policy π to be the greedy policy with respect to V^{π}

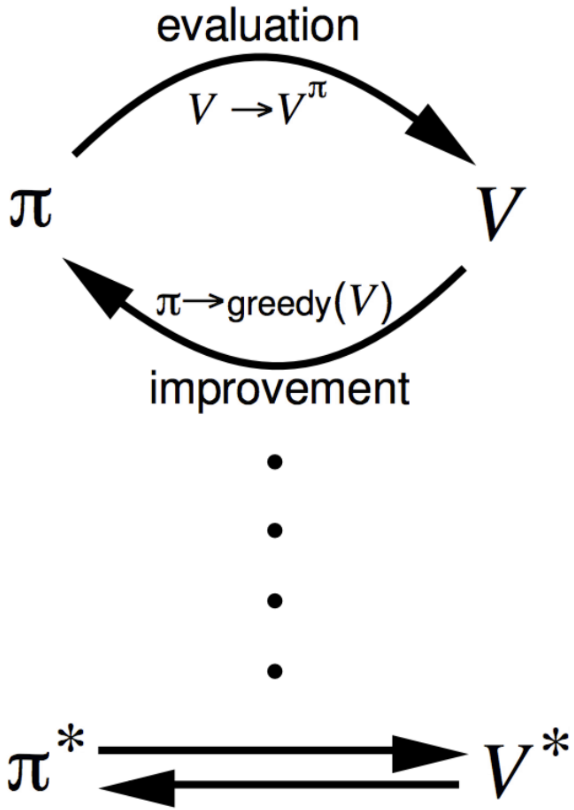
$$\pi(s) \leftarrow \operatorname*{argmax}_a \sum_{s'} P(s'|s,a) V^{\pi}(s')$$

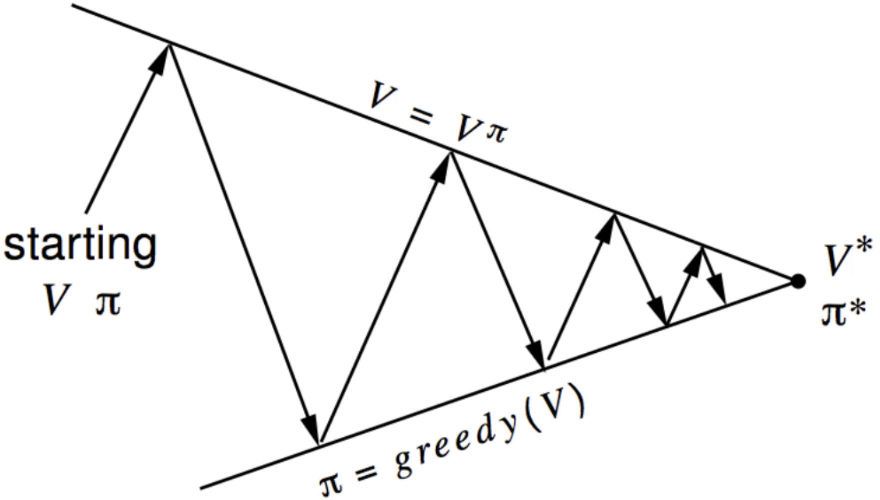
  4. If policy is changed in the last iteration, goto step 2

Policy Evaluation

Policy Improvement

Policy Iteration





Policy Evaluation

- Value equation for fixed policy

$$V_\pi(s) = R(s) + \underbrace{\gamma \sum_{s'} T(s,\pi(s),s') \cdot V_\pi(s')}_{\text{discounted expected value of following policy in the future}}$$

immediate reward

- Equation can be derived from original definition of infinite horizon discounted value

Exact Policy Evaluation via Linear Solver

- $\boldsymbol{V_\pi}$ and $\boldsymbol{R}$ are n-dimensional column vector (one element for each state)
- $\boldsymbol{T}$ is an n x n matrix s.t. $\mathrm{T}(\mathrm{i}, \mathrm{j}) = \mathrm{T}(\mathrm{s}_\mathrm{i}, \pi(s_i), s_j)$

$$V_\pi = R + \gamma \mathrm{T} V_\pi$$

$$\Downarrow$$

$$(I - \gamma \mathrm{T}) V_\pi = R$$

$$\Downarrow$$

$$V_\pi = (I - \gamma \mathrm{T})^{-1} R$$

Policy Evaluation via Value Iteration

- Initialize V(s) to anything, e.g., 0
- Do until change in $||V_{k+1} - V_k||_\infty$ is below desired threshold
  - for every state s, update:

$$V_\pi(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi(s), s') \cdot V_\pi(s')$$

Iterative policy evaluation is guaranteed to converge!

Policy Improvement

- Define $q_\pi(s,a) = \mathrm{E}_\pi[R(s) + \gamma V_\pi(s')] = R(s) + \gamma \sum_{s'} P(s'|s,a) V_\pi(s')$
- We can improve the policy by acting greedily

$$\pi'(s) = \operatorname*{argmax}_{a \in \mathcal{A}} q_\pi(s,a)$$

- This improve the value from any state s over one step

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s,a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned}
v_\pi(s) \leq q_\pi(s, \pi'(s)) &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s)
\end{aligned}$$

Policy Improvement (2)

- If the improvement stops (it will stop eventually)

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- Then the Bellman optimality equation has been satisfied

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

- Therefore $v_\pi(s) = v_*(s)$ for all state s

- So π is the optimal policy

Illustration of policy iteration

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

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

Illustration of policy iteration

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

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

V^π at one iteration

Illustration of policy iteration

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

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

V^π at two iterations

Illustration of policy iteration

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

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

V^π at three iterations (converged)

Grid-World Results

- Approximation of the value function
  - Policy iteration: exact value function after three iteratons
  - Value iteration: after 100 iteration, ||V – V*|| = 7.1 * 10^4
- Calculation of the optimal policy
  - Policy iteration: three iterations
  - Value iteration: 12 iterations
- Note: value iteration converges to the optimal policy long before it converges to the correct value in this MDP

Policy Iteration Complexity

- Each iteration runs in polynomial time in the number of states and actions
- There are at most $|A|^n$ policies and PI never repeats a policy
  - So at most an exponential number of iterations
  - Not a very good complexity bound
- Empirically O(n) iterations are required
  - **Challenge:** try to generate an MDP that requires more than that n iterations
- Still no polynomial bound on the number of PI iterations (open problem)!

Fast Policy Evaluation

- Complexity of policy evaluation by a linear solver: O(n^$^{2.373}$)
  - Prohibitive for large problems

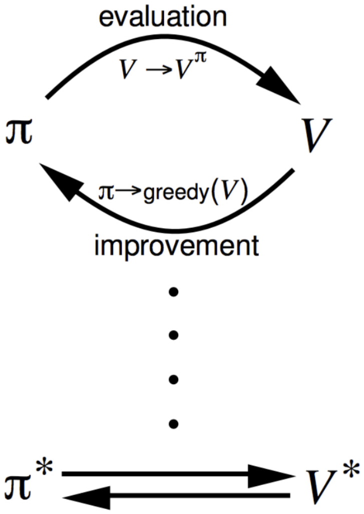
- Using Bellman update with repeating k times for policy evaluation (like value iteration)

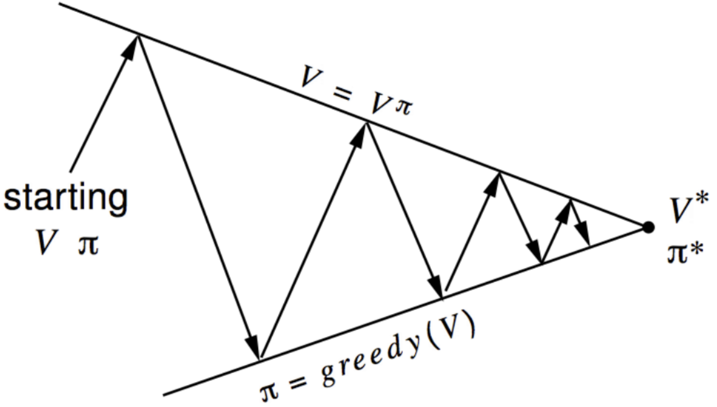
$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s'), \quad \forall s \in \mathcal{S}$$

- Called Modified Policy Iteration, which is much faster.

Generalized Policy Iteration

- Generalized Policy Iteration (GPI): any interleaving of policy evaluation and policy improvement
  - independent of their granularity and other details of the two processes





Policy Iteration vs Value Iteration

- PI requires fewer iterations than VI, but each iteration requires solving policy evaluation instead of just applying Bellman operator

- In practice, policy iteration is often faster
  - Especially, the transition function is structure (e.g., sparse)

- *Modified policy iteration* often perform better than PI and VI
  - Approximately solving policy evaluation using VI

Today’s Outline

- Value iteration
- Policy iteration
- **Asynchronized Dynamic Programming**
- Extensions of MDPs

Synchronous vs Asynchronous Dynamic Programming

- Synchronous DP methods described so far require
  - exhaustive sweeps of the entire state set
  - updates to V only after a full sweep
- Asynchronous DP backs up states individually, in any order
  - Repeat until convergence criterion is met:
    - Select a state and apply the appropriate backup
- Still need lots of computation, but does not get locked into hopelessly long sweeps
- Guaranteed to converge if all states continue to be selected
- Can you select states to backup intelligently?
  - YES: an agent’s experience can act as a guide.

Asynchronous Dynamic Programming

- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming
  - Prioritized sweeping
  - Real-time dynamic programming

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function

$$\hat{V}'(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s')$$

and then set $\hat{V}(s) \leftarrow \hat{V}'(s)$

- In-place value iteration or asynchronous value iteration only stores one copy of value function

Alternatively, can loop over states $s = 1, \ldots, |\mathcal{S}|$ (or randomize over states), and directly set

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s')$$

Prioritized Sweeping

- Use the magnitude of Bellman errors to guide state selection, e.g.

$$\left| \; R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s') \; - \; \hat{V}(s) \; \right|$$

- Backup the state with the largest remaining Bellman error
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

Real-Time Dynamic Programming

- Idea: only states that are relevant to the agent
- Use the agent's experience to guide the selection of states
- After each time-step S_t , A_t , R_{t+1}
- Backup the state S_t

$$\hat{V}'(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\,\hat{V}(s')$$

Full-Width Backups

- DP uses full-width backups
- For each backup (sync or async)
  - Every successor state and action is considered
  - Using knowledge of the MDP transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers Bellman's curse of dimensionality
  - Number of states $n = |S|$ grows exponentially with number of state variables
- Even one backup can be too expensive

Sample Backups

- In subsequent lectures we will consider sample backups
- Using sample rewards and sample transitions <S, A, R, S’>
- Instead of reward function R and transition dynamics P
- Advantages:
  - Model-free: no advance knowledge of MDP required
  - Breaks the curse of dimensionality through sampling
  - Cost of backup is constant, independent of $n = |S|$

Approximate Dynamic Programming

- Approximate the value function
- Using function approximation (e.g., neural net), $\hat{v}(s, \mathbf{w})$
- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration **k**,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
  - For each state $s \in \tilde{\mathcal{S}}$, estimate its target value using Bellman optimality equation

$$\hat{V}'(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a)\, \hat{V}(s')$$

  - Train next value function $\hat{v}(\cdot, \mathbf{w_{k+1}})$) using targets $\{\langle s, \tilde{v}_k(s) \rangle\}$

Today’s Outline

- Value iteration
- Policy iteration
- Asynchronized Dynamic Programming
- **Extensions of MDPs**

Extensions to MDPs

- Infinite and continuous MDPs
- Partially observable MDPs
- Undiscounted, average reward MDPs

Infinite MDPs

- Countably infinite state and/or action spaces
  - Straightforward
- Continuous state and/or action spaces
  - Closed form for linear quadratic model (LQR)
- Continuous time
  - Requires partial differential equations
  - Hamilton-Jacobi-Bellman (HJB) equation
  - Limiting case of Bellman equation as time-step $\rightarrow 0$

Partially observable MDPs

- A Partially Observable MDP is an MDP with hidden states.
  - It is a hidden Markov model with actions.

- An Partially Observable MDP is defined by:
  - A finite set of states $s \in S$
  - A finite set of actions $a \in A$
  - A finite set of observations O
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s'| s, a)
  - An observation function Z
    - $Z(o_{t+1}, s_{t+1}, a_t) = P(o_{t+1} \mid s_{t+1}, a_t)$
  - A reward function R(s)

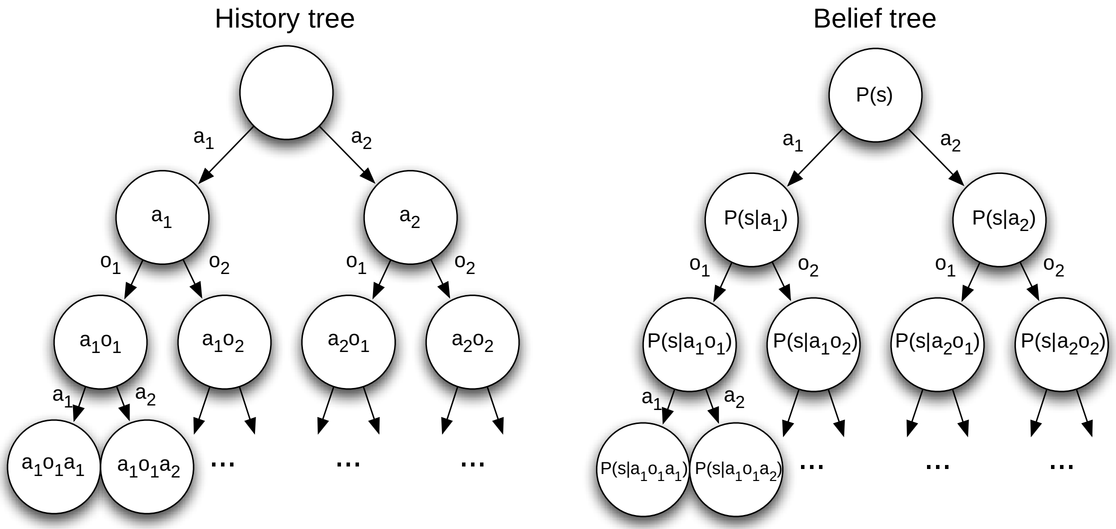
Belief States

- A history h_t is a sequence of actions, observations and rewards,
  - $h_t = < a_0, o_1, R_1, ..., a_{t-1}, o_t, R_t >$

- A belief state $b(h)$ is a probability distribution over states, conditioned on the history h
  - $b(h) = (P[S_t = s_1 \mid H_t = h) , ..., P[S_t = s_n \mid H_t = h])$

Reductions of POMDPs

- The history h_t satisfies the Markov property
- The belief state $b(h_t)$ satisfies the Markov property



- A POMDP can be reduced to an (infinite) history tree
- A POMDP can be reduced to an continuous MDP with belief states

Ergodic Markov Process

- An ergodic Markov process is
  - Recurrent: each state is visited an infinite number of times
  - Aperiodic: each state is visited without any systematic period

Theorem

An ergodic Markov process has a limiting stationary distribution $d^\pi(s)$ with the property

$$d^\pi(s) = \sum_{s' \in \mathcal{S}} d^\pi(s') \mathcal{P}_{s's}$$

Ergodic MDP

- An MDP is ergodic if the Markov chain induced by any policy is ergodic.

For any policy π, an ergodic MDP has an *average reward per time-step* ρ^π that is independent of start state.

$$\rho^\pi = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^{T} R_t\right]$$

Average Reward Value Function

- The value function of an undiscounted, ergodic MDP can be expressed in terms of average reward.
- $\tilde{v}_\pi(s)$ is the extra reward due to starting from state s,

$$\tilde{v}_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=1}^{\infty} (R_{t+k} - \rho^\pi) \mid S_t = s \right]$$

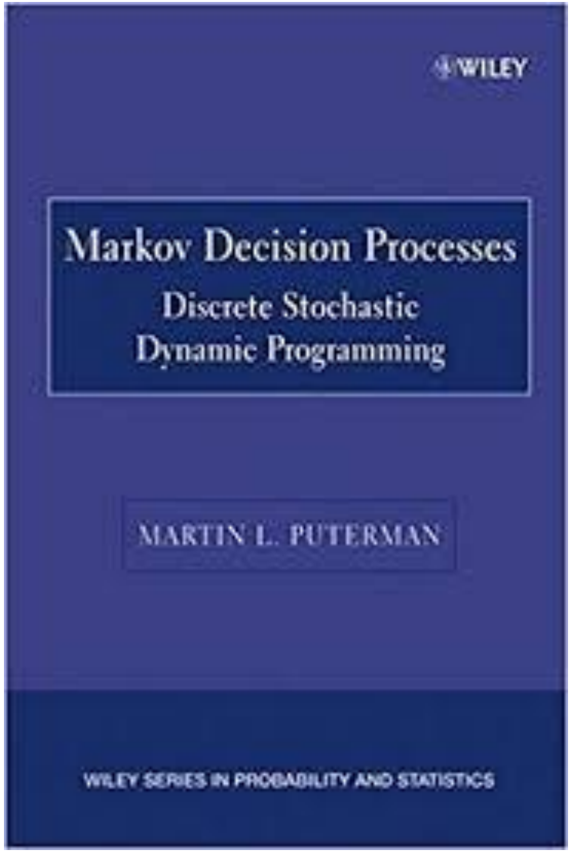
- There is a corresponding average reward Bellman equation

$$\begin{aligned} \tilde{v}_\pi(s) &= \mathbb{E}_\pi \left[(R_{t+1} - \rho^\pi) + \sum_{k=1}^{\infty} (R_{t+k+1} - \rho^\pi) \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[(R_{t+1} - \rho^\pi) + \tilde{v}_\pi(S_{t+1}) \mid S_t = s \right] \end{aligned}$$

Recap: things you should know

- What is an MDP?
- What is a policy?
  - Stationary and non-stationary
- What is a value function?
  - Finite-horizon and infinite horizon
- How to evaluate policies?
  - Finite-horizon and infinite horizon
  - Time/space complexity?
- How to optimize policies?
  - Finite-horizon and infinite horizon
  - Time/space complexity?
  - Why they are correct?

A Recommended MDP Book



Next Time: Reinforcement Learning!