Markov Decision Process

I-Chen Wu

- Sutton, R.S. and Barto, A.G., Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998. (Bible for RL)
  - http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html
  - Chapters 3-4
- David Silver, Online Course for Deep Reinforcement Learning.
  - http://www.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html
  - Chapters 2-3

Outline

- Introduction
- Markov Property
- Markov Process
- Markov Reward Process (MRP)
- Markov Decision Process (MDP)
- Partially Observable Markov Decision Process (POMDP)

The purpose of this chapter:

- Introduce all kinds of Markov processes

Introduction

- Markov decision processes formally describe an environment for reinforcement learning
  - where the environment is fully observable.
  - i.e. The current state completely characterizes the process
  - E.g., 2048.
- Almost all RL problems can be formalized as MDPs, e.g.
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state

Markov Property

- Markov Property:
  - "The future is independent of the past given the present"
  - Definition: A state S_t is Markov if and only if
    $$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t]$$
- Comments:
  - The state captures all relevant information from the history
  - Once the state is known, the history may be thrown away
  - i.e. The state is a sufficient statistic of the future
- But, what if the history does matter?
  - Simply let S_t carry all information of history, $H_t = (S_1, \ldots, S_{t-1})$.
    - E.g., the castling rule for chess.
  - Then, it satisfies Markov Property.

Markov Process

- A Markov process is a memoryless random process,
  - i.e. a sequence of random states S_1, S_2, ... with the Markov property.

Definition:

- A Markov Process (or Markov Chain) is a tuple $<\mathcal{S}, \mathcal{P}>$
  - $\mathcal{S}$ is a (finite) set of states
  - $\mathcal{P}$ is a state transition probability matrix (part of the environment), $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$

State Transition Matrix

- For a Markov state s and successor state s', the state transition probability is defined by
  $$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$
- State transition matrix $\mathcal{P}$: (assume n states)

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \dots & & \dots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

  – Each row of matrix sums to 1.

Markov Reward Process (MRP)

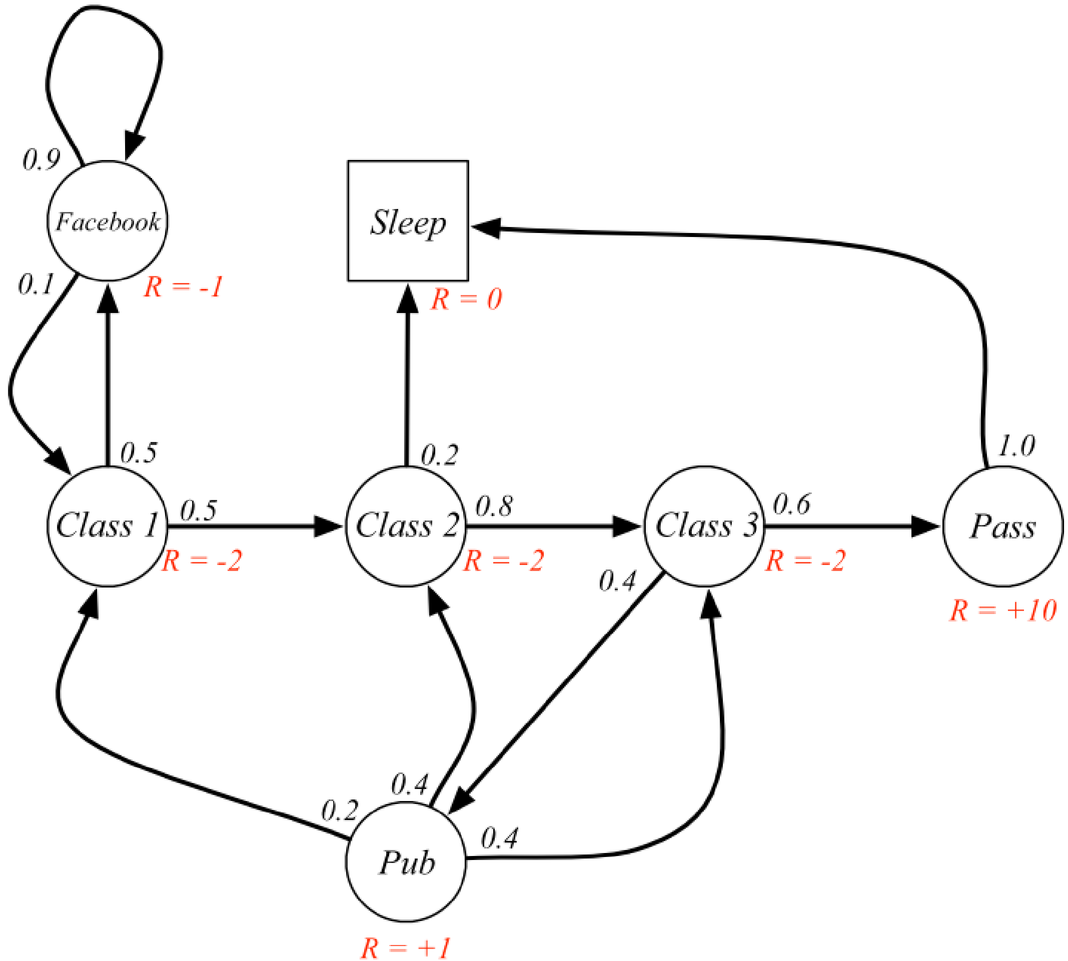
- A Markov reward process is a Markov chain with values.

Definition:

- A Markov Reward Process is a tuple $<\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma>$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{P}$ is a state transition probability matrix (part of the environment),
    $\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
  - $\mathcal{R}$ is a reward function,
    $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - γ is a discount factor $\gamma \in [0, 1]$.

-

Example: Student MRP



Return

Definition

- The return G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Notes:

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R is diminishing
  - $\gamma^k R$, after $k + 1$ time-steps.
- This values immediate reward above delayed reward.
- Discount:
  - γ close to 0 leads to ”myopic” evaluation
  - γ close to 1 leads to ”far-sighted” evaluation

Value Function

- The value function $v(s)$ gives the long-term value of s
- Definition
  - The state value function $v(s)$ of an MRP is the expected return starting from state s
  - $v(s) = \mathbb{E}[G_t \mid S_t = s]$

Bellman Equation for MRPs

- The value function can be decomposed into two parts:
  - immediate reward R_{t+1}
  - discounted value of successor state $\gamma\, v(S_{t+1})$
- $$\begin{aligned} v(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma\, G_{t+1} \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma\, v(S_{t+1}) \mid S_t = s] \end{aligned}$$
- For a transition (s, r, s'), we have

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}\, v(s')$$

Bellman Equation in Matrix Form

- The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

  - where v is a column vector with one entry per state.

$$\begin{bmatrix} v(1) \\ \dots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \dots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \dots & & \dots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \dots \\ v(n) \end{bmatrix}$$

Solving the Bellman Equation

- The Bellman equation is a linear equation
- It can be solved directly:

$$v = \mathcal{R} + \gamma \mathcal{P} v$$
$$v = (1 - \gamma \mathcal{P})^{-1} \mathcal{R}$$

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Markov Decision Processes (MDP)

- A Markov Decision Process is a tuple $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma>$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{P}$ is a state transition probability matrix (part of the environment),
    $\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
  - $\mathcal{R}$ is a reward function,
    $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
  - γ is a discount factor $\gamma \in [0, 1]$.

Example: Recycling Robot

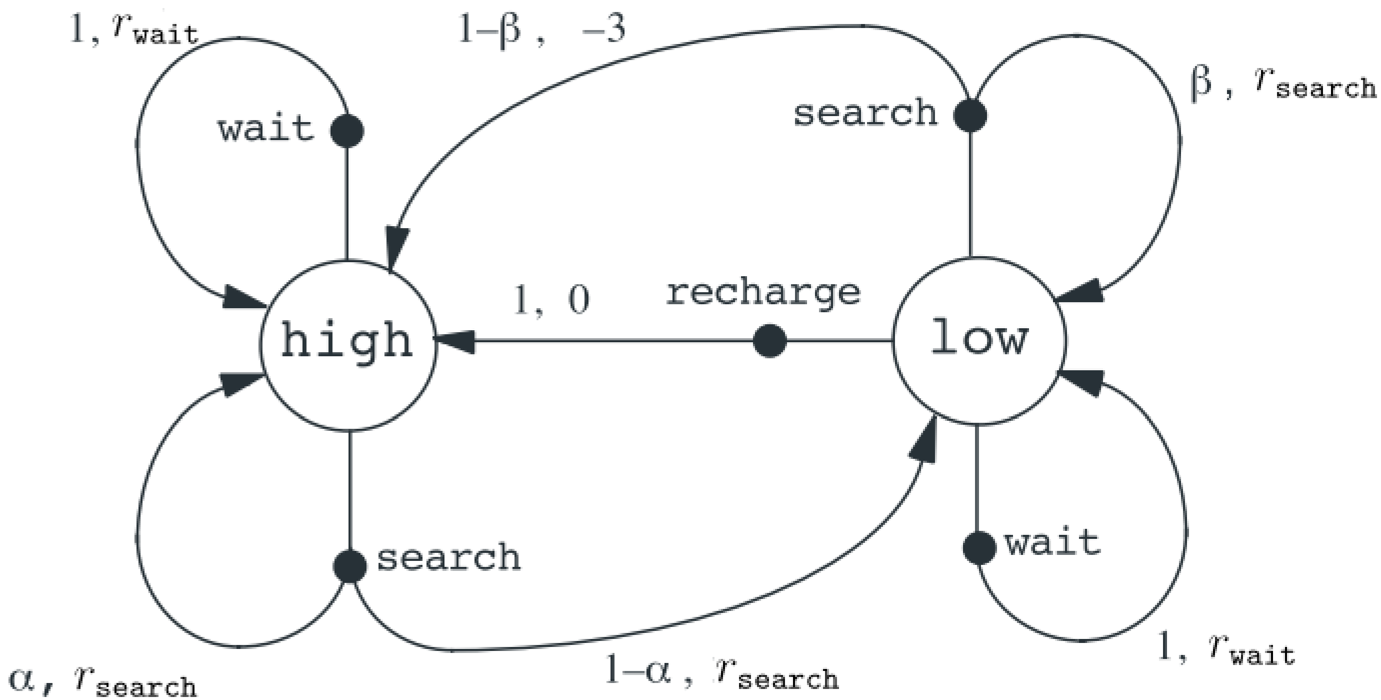


Figure 3.3: Transition graph for the recycling robot example.

Example: Recycling Robot

- Transition and Rewards:

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	$0.$

Policies

- A policy is the agent's behavior
  - It is a map from state to action
  - A policy fully defines the behaviour of an agent
  - MDP policies depend on the current state (not the history)
    - i.e. Policies are stationary (time-independent),
      $A_t \sim \pi(\cdot\,|S_t), \forall t > 0$
- Policy types:
  - Deterministic policy: $a = \pi(s_i)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|\ S_t = s]$
    - Sometimes, written in $\pi(s, a)$.
- Examples:
  - In 2048: Up/down/left/right
  - In robotics: angle/force/…

Policy and MRP

- Given an MDP $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma>$ and a policy π
- The state sequence $S_1, S_2, \ldots$ is a Markov process $<\mathcal{S}, \mathcal{P}^{\pi}>$
- The state and reward sequence $S_1, R_2, S_2, R_3, \ldots$ becomes a Markov reward process (MRP) $<\mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma>$
  - $\mathcal{P}^{\pi}$ is a state transition probability matrix (part of the environment),

$$\mathcal{P}^{\pi}_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^{a}_{ss'}$$

  - $\mathcal{R}^{\pi}$ is a reward function,

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^{a}_{s}$$

- So, the property of MRP can be applied.

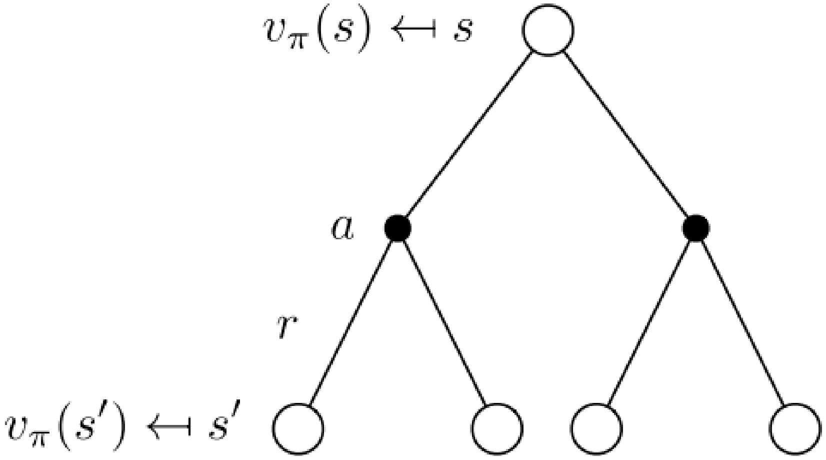
Value Function

- A value function is a prediction of future reward
  - Used to evaluate the goodness/badness of states
    - therefore to select between actions.
  - Return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$
- Types of value functions under policy π:
  - State value function: the expected return from s.
    $$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] = \mathbb{E}_\pi[G_t \mid S_t = s]$$
  - Q-Value function: the expected return from s taking action a.
    $$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$
- Examples:
  - In 2048, the expected score from a board S_t.

Bellman Expectation Equation for π

- State value function:

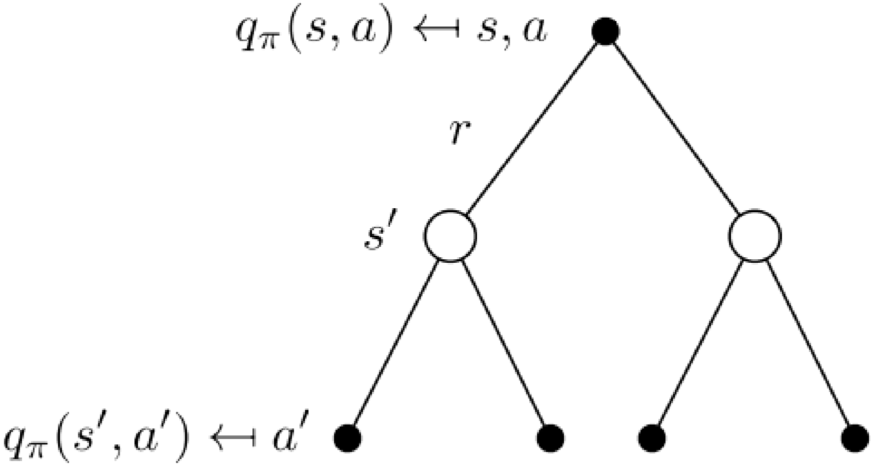
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$



Bellman Expectation Equation for π

- Q value

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$



Bellman Expectation Equation in Matrix

- The Bellman expectation equation can be expressed concisely using the induced MRP.
- So, it can be solved directly:

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$
$$v_\pi = (1 - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

Optimal Value Function

- The optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

- The optimal action-value function $q_*(s,a)$ is the maximum action-value function over all policies

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

- Notes:
  - The optimal value function specifies the best possible performance in the MDP.
  - An MDP is "solved" when we know the optimal value fn.

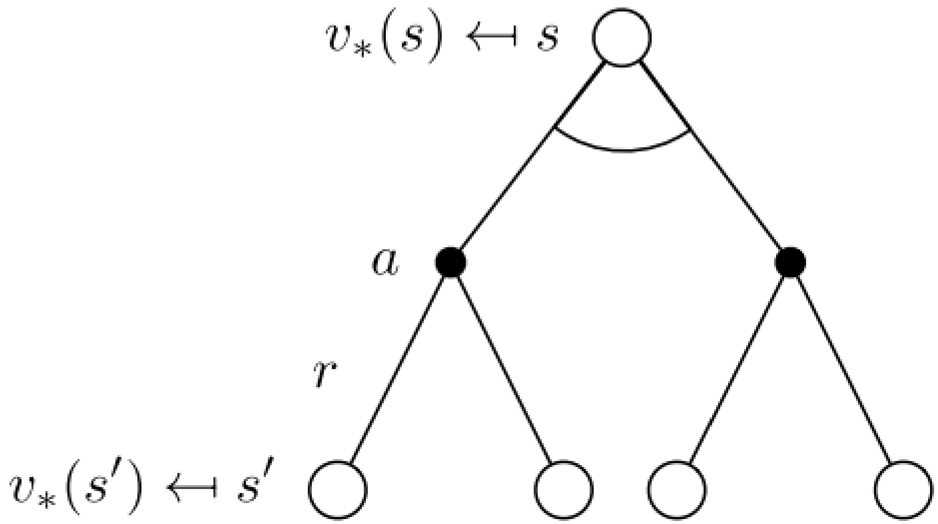
Optimal Policy

- Define a partial ordering over policies
  $\pi \geq \pi', v_*(s) = \max_\pi v_\pi(s)$ if $v_\pi(s) \geq v_{\pi'}(s), \forall s$
- Theorem:
  - There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$.
  - All optimal policies achieve the optimal value function,
  - All optimal policies achieve the optimal action-value function,

Finding an Optimal Policy

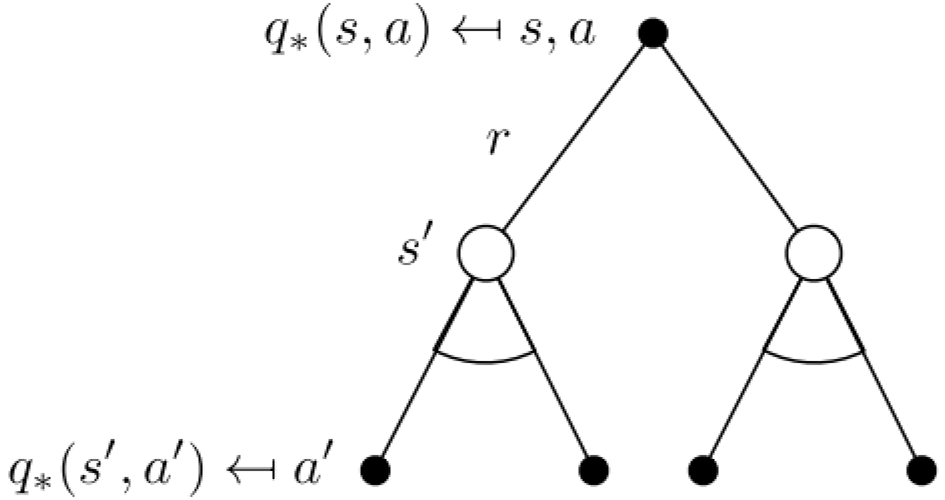
- An optimal policy can be found by maximizing over $q_*(s,a)$,
  - $\pi(a|s) = 1$, if $a = argmax_a\, q_*(s,a)$
  - $\pi(a|s) = 0$, otherwise.
- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s,a)$, we immediately have the optimal policy
- What about state value function $v_*(s)$?
  - Similar, but we need to know model, $\mathcal{P}_{ss'}^{a}$. → not model free.

Bellman Optimality Equation for V*



$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \, v_*(s') \right)$$

Bellman Optimality Equation for Q*



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}} q_\pi(s,a')$$

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa
- Extensions to MDPs
  - Infinite and continuous MDPs
  - Partially observable MDPs
  - Undiscounted, average reward MDPs

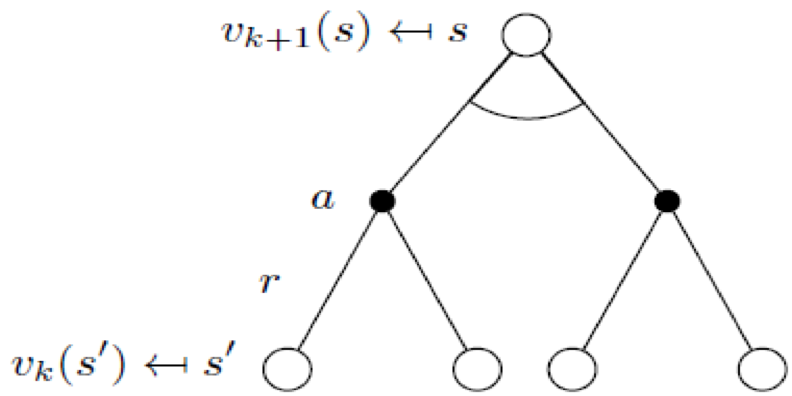
Partially Observable Markov Decision Processes (POMDP)

- A POMDP is a tuple $<\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma>$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{O}$ is a finite set of observations
  - $\mathcal{P}$ is a state transition probability matrix,
    $\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' |\ S_t = s, A_t = a]$
  - $\mathcal{R}$ is a reward function,
    $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - $\mathcal{Z}$ is an observation function,
    $\mathcal{Z}_{s'o}^{a} = \mathbb{P}[O_{t+1} = o |\ S_{t+1} = s', A_t = a]$
  - γ is a discount factor $\gamma \in [0, 1]$.
- Examples:
  - Like Mahjong (麻將) in games, hidden scene in robotics.
- POMDP vs. MDP
  - A MDP can be trivially mapped onto a POMDP
  - A POMDP can be mapped onto an MDP:

Example: Mahjong

- Partially observable:

Value Iteration



$$v_{n+1}(s) = \max_{a \in A} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \, v_n(s') \right)$$

or:

$$V^{(n+1)}(s) = \max_{a \in \mathcal{A}} \left(\mathbb{E}_{s'|s,a} \left[r + \gamma V^{(n)}(s') \right] \right)$$

Operator View

- Value iteration update is viewed as:
  - A function $\mathcal{T}: \mathcal{S} \rightarrow \mathcal{S}$.
  - Called backup operator.

$$[\mathcal{T}V](s) = \max_{a \in \mathcal{A}} \left(\mathbb{E}_{s'|s,a}[r + \gamma V(s')] \right)$$

Algorithm Value Iteration

Initialize V (0) arbitrarily.

for n = 0, 1, 2, . . . until termination condition do

$$V^{(n+1)} = \mathcal{T}V^{(n)}$$

end

Contraction Mapping View

- Backup operator $\mathcal{T}$ is a contraction with modulus γ under ∞-norm
$$||\mathcal{T}V - \mathcal{T}W||_\infty \leq \gamma \, ||V - W||_\infty$$
- By contraction-mapping principle, it has a fixed point V^*
  - by iterating
$$V, \mathcal{T}V, \mathcal{T}^2V, \ldots \rightarrow V^*$$

Policy Evaluation

- Problem: how to evaluate fixed policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \,|\, S_t = s\,] = \mathbb{E}_\pi[R_{t+1} + \gamma V^\pi(S_{t+1}) \,|\, S_t = s]$$

- Backwards recursion involves a backup operation

$$V^{(k+1)} = \mathcal{T}^\pi V^{(k)}$$

  - $\mathcal{T}^\pi$ is defined as:

$$[\mathcal{T}^\pi V](s) = \mathbb{E}_{s'|s,a=\pi(s)}[r + \gamma V(s')\,]$$

- $\mathcal{T}^\pi$ is also a contraction with modulus γ, sequence

$$V, \mathcal{T}^\pi V, (\mathcal{T}^\pi)^2 V, (\mathcal{T}^\pi)^3 V, \ldots \rightarrow V^\pi$$

- $V = \mathcal{T}^\pi V$ is a linear equation that we can solve directly.

Policy Iteration: Overview

- Alternate between
  - Evaluate policy $\pi \Rightarrow V^{\pi}$
  - Set new policy to be greedy policy for V^{π}

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, \mathbb{E}_{s'|s,a}[R_{t+1} + \gamma V^{\pi}(s')\,]$$

- Guaranteed to converge to optimal policy and value function in a finite number of iterations, when $\gamma < 1$
- Value function converges faster than in value iteration

Algorithm Policy Iteration

Initialize $\pi^{(0)}$ arbitrarily.

for $n = 1, 2, \ldots$ until termination condition do

$V^{(n+1)} = \text{Solve}\ [V = \mathcal{T}^{\pi^{(n-1)}} V]$

$\pi^{(n)} = \mathcal{G} V^{(n)}$ $\mathcal{G}$: a greedy mapping function.

end

Modified Policy Iteration

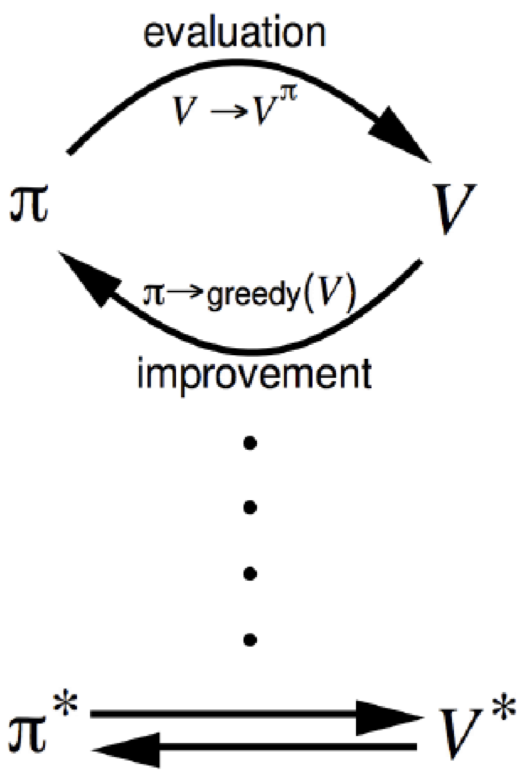
- Update π to be the greedy policy, then value function with k backups (k-step lookahead)

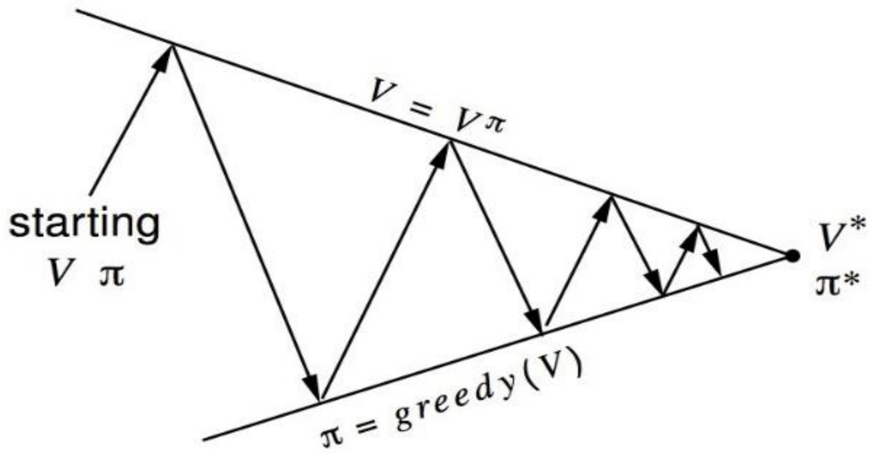
Algorithm Modified Policy Iteration
Initialize $V^{(0)}$ arbitrarily.
for n = 1, 2, . . . until termination condition do
$\pi^{(n+1)} = \mathcal{G}V^{(n)}$
$V^{(n+1)} = \left(\mathcal{T}^{\pi^{n+1}}\right)^k V^{(n)}$, for integer $k \geq 1$.
end

- $k = 1$: value iteration
- $k = \infty$: policy iteration

Policy Iteration in Different View

- Policy evaluation → Estimate v_π
  - Iterative policy evaluation
- Policy improvement → Generate $\pi' \geq \pi$
  - Greedy policy improvement





Policy Improvement

- Definition of policy improvement
  - Let π and π' be any pair of deterministic policies
    - For all $s \in S$, "$\pi(s)$ performs better than $\pi'(s)$"
    - or $\pi \geq \pi'$.
- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- Notes:
  - In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
  - In general, need more iterations of improvement / evaluation
  - But this process of policy iteration always converges to π^*

Proof of Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can improve the policy by acting greedily

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

- This improves the value from any state s over one step,

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

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

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

Convergence of Policy Improvement

- If improvements stop,
  $$q_\pi\big(s, \pi'(s)\big) = \max_{a \in A} q_\pi(s, a) \geq q_\pi\big(s, \pi(s)\big) = v_\pi(s)$$
- Then the Bellman optimality equation has been satisfied
  $$v_\pi(s) = \max_{a \in A} q_\pi(s, a)$$
- This implies $v_\pi(s) = v_*(s)$ for all $s \in S$
- The above proves that π will converge to an optimal policy.