Reinforcement Learning

Dynamic Programming; Monte Carlo Methods

Subramanian Ramamoorthy
School of Informatics

27 January 2017

Recap: Key Quantities defining an MDP

- System dynamics are stochastic – represented by a probability distribution.
- Problem is defined as maximization of expected rewards
  - Recall that $E(X) = \Sigma x_i p(x_i)$ for finite-state systems

State Transition Dynamics:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$$

Expected Rewards:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

Note that:
$\mathcal{R}^a_s = \sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'}$

Recap: Decision Criterion

What is the criterion for optimization (i.e., learning)?

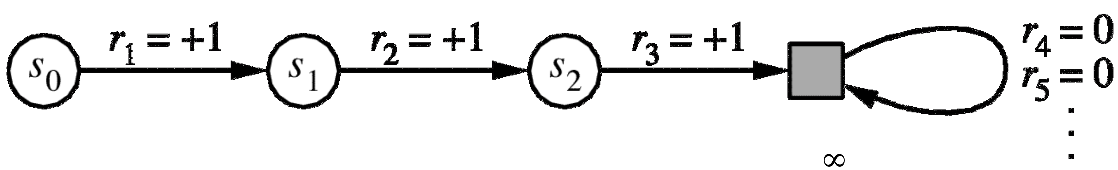
$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma, 0 \leq \gamma \leq 1,$ is the **discount rate**.

Effect of changing γ?

Notation for Episodic vs. Infinite

- In (discrete) episodic tasks, we could number the time steps of each episode starting from zero.
- We usually do not have to distinguish between episodes, so we write s_t instead of $s_{t,j}$ for the state at step *t* of episode *j*.
- Think of each episode as ending in an absorbing state that always produces reward of zero:



- We can cover all cases by writing $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$,

where γ can be 1 only if a zero reward absorbing state is always reached.

Three Aspects of the RL Problem

- **Planning**
  The MDP is known (states, actions, transitions, rewards).
  Find an optimal policy, π^*!

- **Learning**
  The MDP is unknown. You are allowed to interact with it.
  Find an optimal policy π^*!

- **Optimal learning**
  While interacting with the MDP, minimize the loss due to not using an optimal policy from the beginning.

Solving MDPs – Many Dimensions

- Which problem? (Planning, learning, optimal learning)
- Exact or approximate?
- Uses samples?
- Incremental?
- Uses value functions?
  - Yes: Value-function based methods
    - Planning: DP, Random Discretization Method, FVI, …
    - Learning: Q-learning, Actor-critic, …
  - No: Policy search methods
    - Planning: Monte-Carlo tree search, Likelihood ratio methods (policy gradient), Sample-path optimization (Pegasus), …
- Representation
  - Structured state:
    - Factored states, logical representation, …
  - Structured policy space:
    - Hierarchical methods

Value Functions

- Value functions are used to determine how good it is for the agent to be in a given state
  - Or, how good is it to perform an action from a given state?
- This is defined w.r.t. a specific policy, i.e., distribution $\pi(s,a)$
- State value function:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right\}$$

- Action (or State-Action) value function:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right\}$$

Value Functions

Note that there are multiple sources of (probabilistic) uncertainty:

- In state s, one is allowed to select different actions a
- The system may transition to different states s' from s
- Depending on the above, return (defined in terms of reward) is a random variable – which we seek to maximize in expectation

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\ &= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} \end{aligned}$$

Recursive Form of V – **Bellman Equation**

$$V^\pi(s) = E_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\}$$

We rewrite as follows:

Expand 1-step forward & rewrite expectation

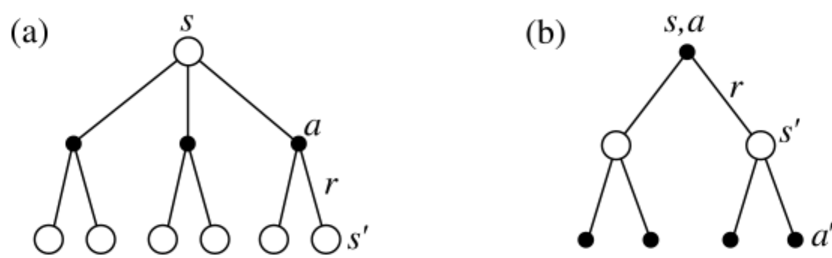
- First term: $\sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'}$
- Second term: $\sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \gamma E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\}$

$$\therefore V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\}]$$

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$$

Backup Diagrams

$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')]$$



Backup diagrams for (a) V^{π} and (b) Q^{π}.

- If you go all the way 'down', you can just read off the reward value
- The backup process (i.e., recursive equation above) allows you to compute the corresponding value at current state
  - taking transition probabilities into account

Bellman Equation for Q (State-Action Value Function)

$$
\begin{aligned}
Q^\pi(s,a) &= E_\pi\{R_t | s_t = s, a_t = a\} \\
&= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \\
&= E_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a\} \\
&= \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s'\}] \\
&= \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(s',a') E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s', a_{t+1} = a'\}] \\
&= \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a')]
\end{aligned}
$$

Optimal Value Function

- For finite MDPs, $\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s)\ \forall s \in \mathcal{S}$
- Let us denote the optimal policy π^*
- The corresponding optimal value functions are:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

- From this, $Q^*(s,a) = E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\right\}$

- Will there always be a well defined π^* ?

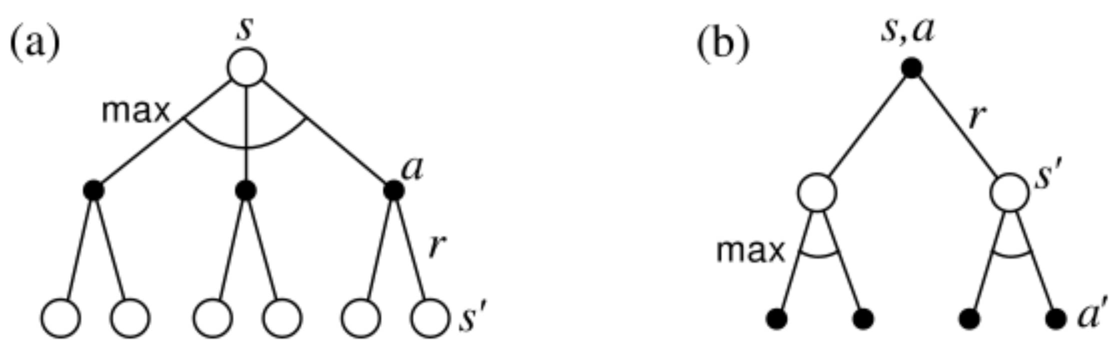
  Theorem [Blackwell, 1962] – For every MDP with finite state/action space, there exists an optimal deterministic stationary plan

Recursive Form of V^*

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s,a) \\
&= \max_a E_{\pi^*}\{R_t | s_t = s, a_t = a\} \\
&= \max_a E_{\pi^*}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \\
&= \max_a E_{\pi^*}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s', a_{t+1} = a'\} \\
&= \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_{t+1} = s', a_{t+1} = a'\} \\
&= \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^*(s')]
\end{aligned}
$$

Backup for Q^*

$$
\begin{aligned}
Q^*(s,a) &= E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a') | s_t = s, a_t = a\} \\
&= \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')]
\end{aligned}
$$



Backup diagrams for (a) V^* and (b) Q^*

What is Dynamic Programming?

Given a known model of the environment as an MDP (transition dynamics and reward probabilities),

DP is a collection of algorithms for computing optimal policies (via Optimal Value Functions)

Policy Evaluation

How to compute $V(s)$ for an arbitrary policy π? (*Prediction* problem)

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \qquad \forall s \in \mathcal{S}$$

For a given MDP, this yields a system of simultaneous equations

– as many unknowns as states

– Solve using linear algebraic computation

Solve iteratively, with a sequence of value functions, $V_0, V_1, V_2, \ldots : \mathcal{S} \mapsto \Re$

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \; \forall s \in \mathcal{S}$$

$V_k = V^\pi$ is a fixed-point for these updates, as $k \to \infty$
- *Iterative* policy evaluation.

Computationally…

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \; \underline{\forall s \in \mathcal{S}}$$

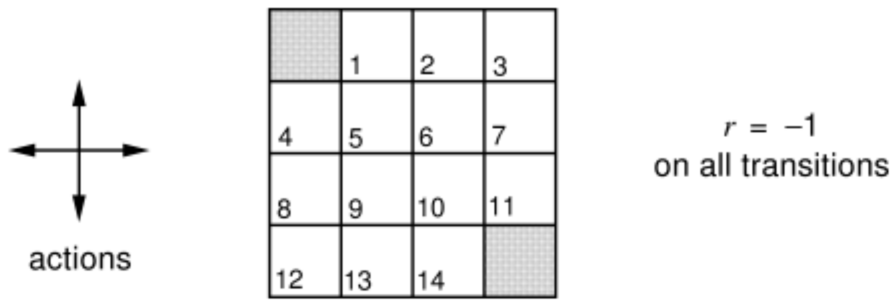
We could achieve this in a number of different ways:

- Maintain two arrays, computing iterations over one and copying results to the other
- In-place: Overwrite as new backed-up values become available
- It can be shown that this algorithm will also converge to optimality (somewhat faster, even)
  - Backups *sweep* through the space
  - Sweep order has significant influence on convergence rates

Iterative Policy Evaluation

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Grid-World Example



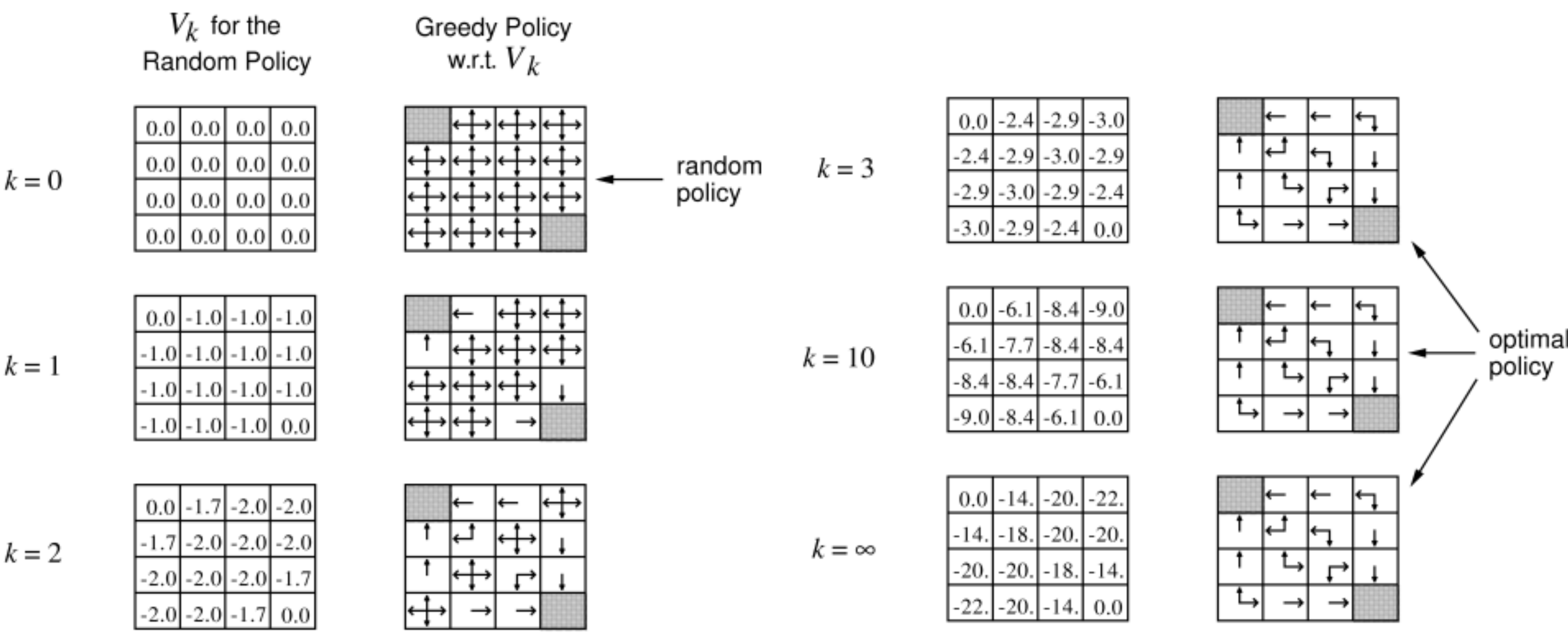
Four possible actions: $\mathcal{A} = \{$ up, down, right, left$\}$

- the actions change state deterministically (but, not allowed to go off grid)

Encoded in transition probabilities, e.g., $\mathcal{P}_{5,6}^{\text{right}} = 1$, $\mathcal{P}_{5,10}^{\text{right}} = 0$, $\mathcal{P}_{7,7}^{\text{right}} = 1$

Undiscounted, episodic task with reward -1 everywhere except goal states.

Iterative Policy Evaluation in Grid World



Note: *The value function can be searched greedily to find long-term optimal actions*

Policy Improvement

Does it make sense to deviate from $\pi(s)$ at any state (following the policy everywhere else)?

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]$$

If π and π' are any two deterministic policies such that $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$ then $V^{\pi'}(s) \geq V^\pi(s)$

If the inequality $Q^\pi(s, \pi'(s)) > V^\pi(s)$ is strict for any state, there must be at least that many states for which $V^{\pi'}(s) > V^\pi(s)$

- Policy Improvement Theorem [Howard/Blackwell]

Key Idea Behind Policy Improvement

$$
\begin{aligned}
V^{\pi}(s) &\leq Q^{\pi}(s, \pi'(s)) \\
&= E_{\pi'}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\} \\
&\leq E_{\pi'}\{r_{t+1} + \gamma Q^{\pi}(s_{t+1}, \pi'(s_{t+1})) \mid s_t = s\} \\
&= E_{\pi'}\{r_{t+1} + \gamma E_{\pi'}\{r_{t+2} + \gamma V^{\pi}(s_{t+2})\} \mid s_t = s\} \\
&= E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 V^{\pi}(s_{t+2}) \mid s_t = s\right\} \\
&\leq E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^{\pi}(s_{t+3}) \mid s_t = s\right\} \\
&\vdots \\
&\leq E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \mid s_t = s\right\} \\
&= V^{\pi'}(s).
\end{aligned}
$$

Computing Better Policies

Starting with an arbitrary policy, we'd like to approach truly optimal policies. So, we compute new policies using the following,

$$\begin{aligned}\pi'(s) &= \arg\max_a Q^\pi(s,a) \\ &= \arg\max_a E\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a\right\} \\ &= \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]\end{aligned}$$

Are we restricted to deterministic policies? No.

With stochastic policies, $Q^\pi(s, \pi'(s)) = \sum_a \pi'(s,a) Q^\pi(s,a)$

Policy Iteration

We can combine policy evaluation and improvement to obtain a sequence of monotonically improving policies and value functions

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} V^{\pi_2} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

– Each policy is guaranteed to be a strict improvement over previous one (unless it is already optimal)

  [Policy Improvement Theorem]

– As a finite MDP admits finitely many policies, this *eventually* converges to an optimal policy

Policy Iteration Algorithm

1. Initialise
   - $\pi =$ arbitrary deterministic policy
   - $V =$ arbitrary value function
   - $\theta =$ small positive number

2. Policy Evaluation
   - For each state
   - New $V = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$ where $a = \pi(s)$
   - Repeat until no V changes by more than θ

3. Policy Improvement
   - Get $b = \pi(s)$
   - New $\pi = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
   - If policy changed, i.e. new $\pi(s) \neq b$ for some s, goto 2

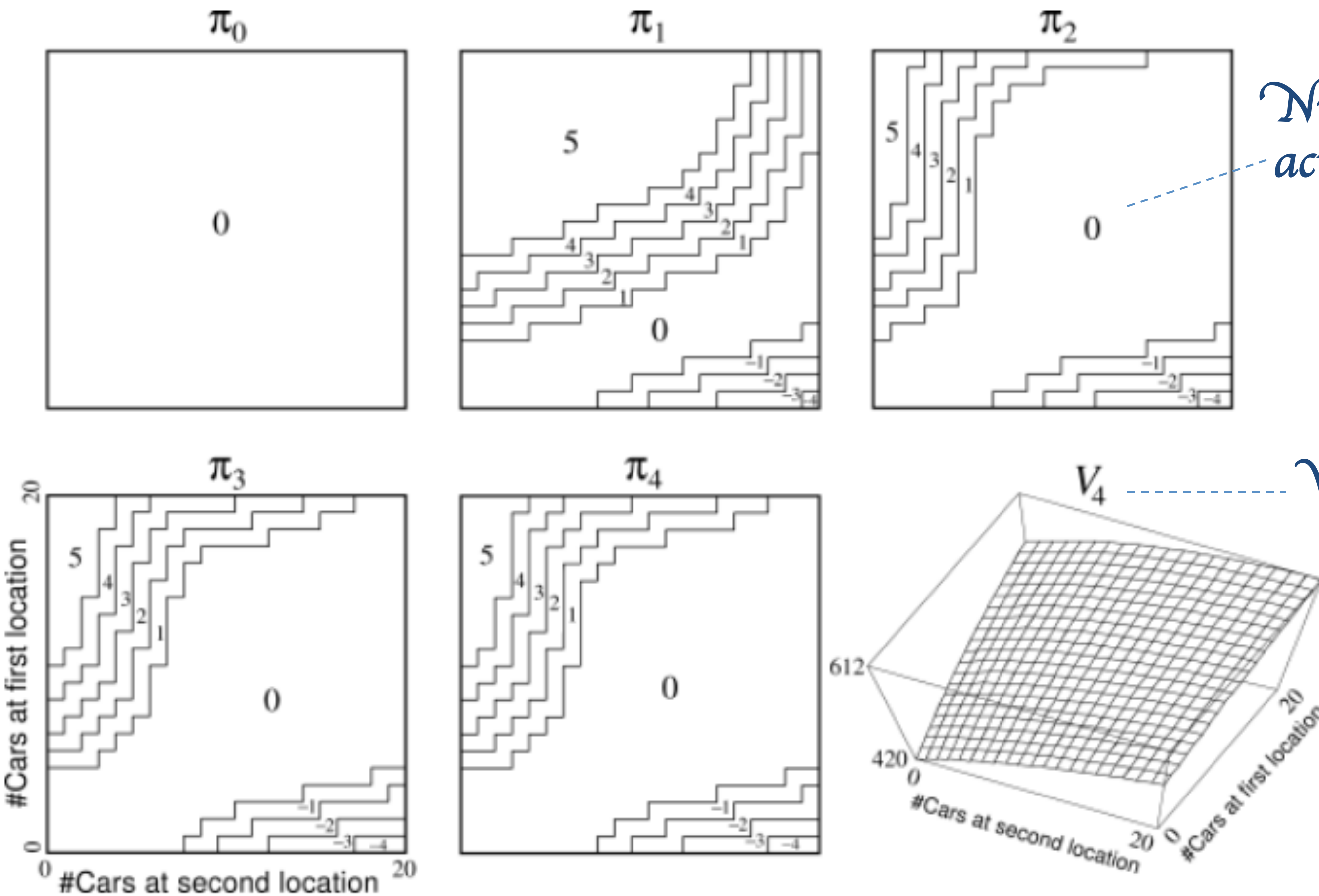
Example: Jack's Car Rental

- £10 for each car rented (must be available when request received)
- Two locations, maximum of 20 cars at each
- Cars returned and requested randomly
  - Poisson distribution, n returns/requests with probability $\frac{\lambda^n}{n!}e^{-\lambda}$
  - Location 1: Average requests = 3, Average returns = 3
  - Location 2: Average requests = 4, Average Returns = 2
- Can move up to 5 cars between locations overnight (costs £2 each)

Problem setup:

- States, actions, rewards?
- Transition probabilities?

Solution: Jack's Car Rental



Numbers indicate action: #cars to move

Value

Points to Ponder: Jack's Car Rental

- Suppose first car moved is free but all others transfers cost £2
  - From Location 1 to Location 2 (not other direction!)
  - Because an employee would anyway go in that direction, by bus

- Suppose only 10 cars can be parked for free at each location
  - More than 10 incur fixed cost £4 for using an extra parking lot

… typical examples of '*real-world nonlinearities*'

Value Iteration

Each step in Policy Iteration needs Policy Evaluation (upto convergence)
- can we avoid this computational overhead?

Just update the values for *one* iteration and then improve the policy.

Update rule:

$$V = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$$

... use Bellman equation as update rule

So we sweep through the state space once (and don't wait for V to stop changing, as in policy evaluation), then improve the policy, then repeat.

This update combines the one-iteration update of V plus the policy improvement (greedification wrt V) in one step.

Value Iteration Algorithm

1. Initialise
   - $V,\ \pi =$ arbitrary
2. Repeat
   - For each state
   - Update $V(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
   - Until no V changes by more than some small amount
3. Policy is
   - $\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

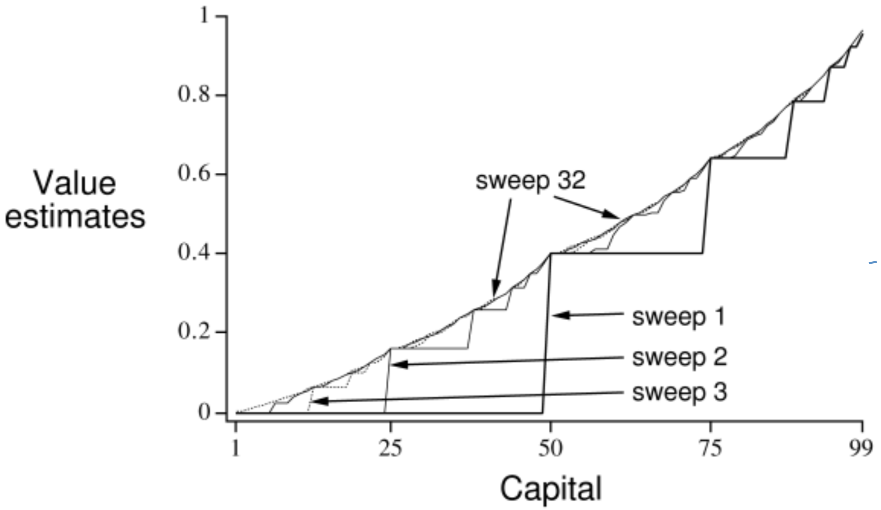
Example: Gambler's Problem

- Gambler can repeatedly bet on a coin flip
- Heads: wins stake; Tails: loses his money
- Initial capital $\in$ {\$1, \$2, … , \$99}
- Gambler has won if he reaches \$100 and has lost if he goes bankrupt (\$0)
- Unfair coin: p(H) = 0.4, p(T) = 0.6

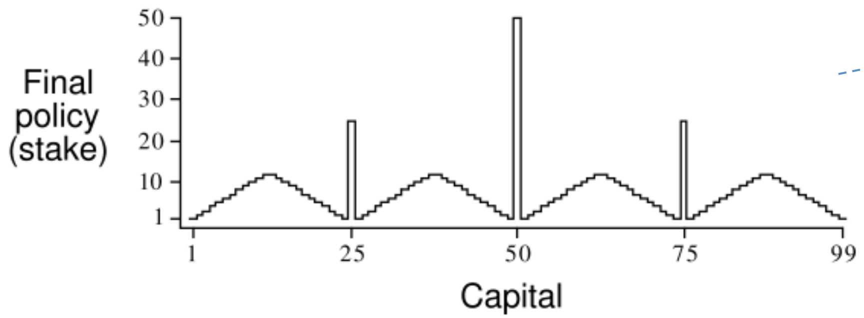
Problem formulation:

- States, Actions, Rewards?
- State transitions?

Solution to Gambler's Problem

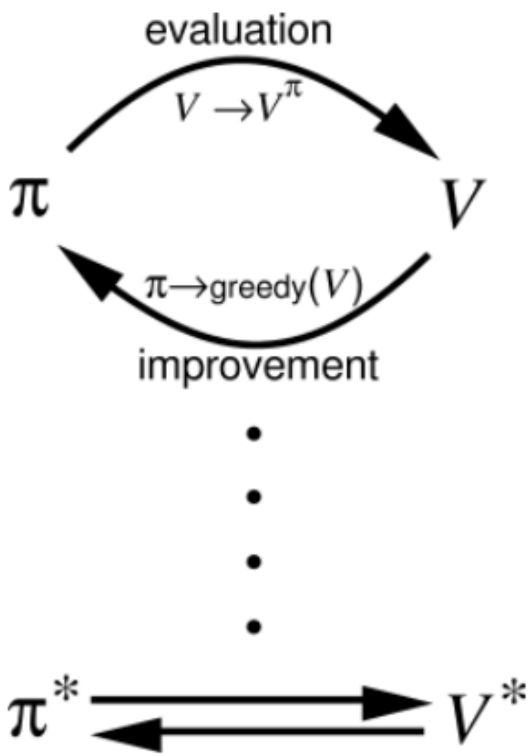


Based on successive sweeps of value iteration

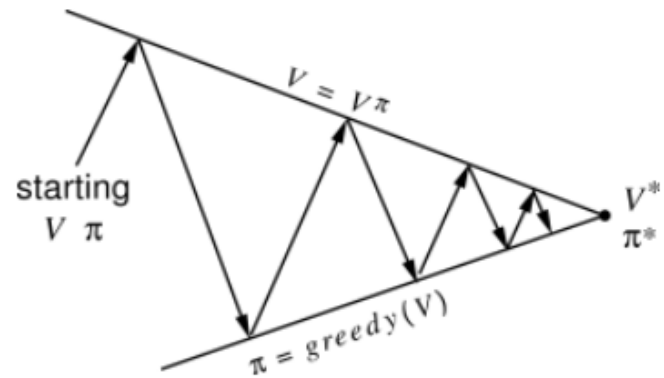


Why does it look so strange?

Generalized Policy Iteration



Caricature of the process:



Builds on the notion of interleaving evaluation and improvement – but allows the granularity to be flexible

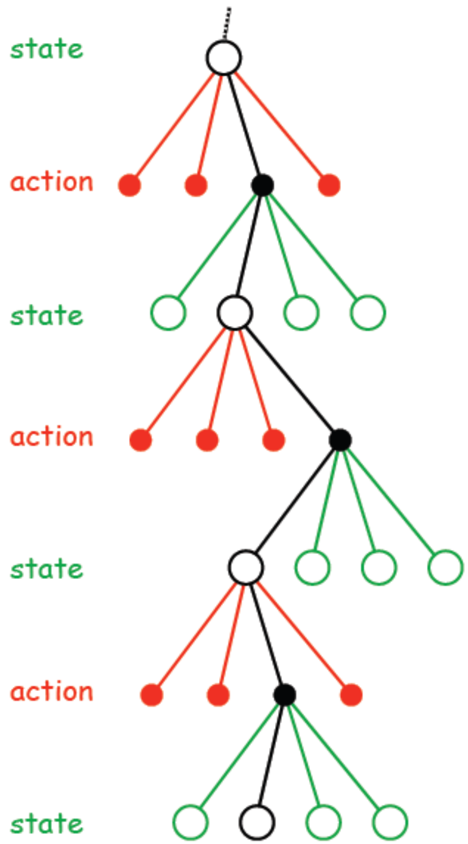
Monte Carlo Methods

- Learn value functions
- Discover optimal policies
- Do not assume knowledge of model as in DP, i.e., $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$

- Learn from experience: Sample sequences of states, actions and rewards (s, a, r)
  - In simulated or real (e.g., physical robotic) worlds
  - Clearly, simulator *is* a model but not a *full* one as in a prob. distribution

- Eventually attain optimal behaviour (same as with DP)

Learning in MDPs

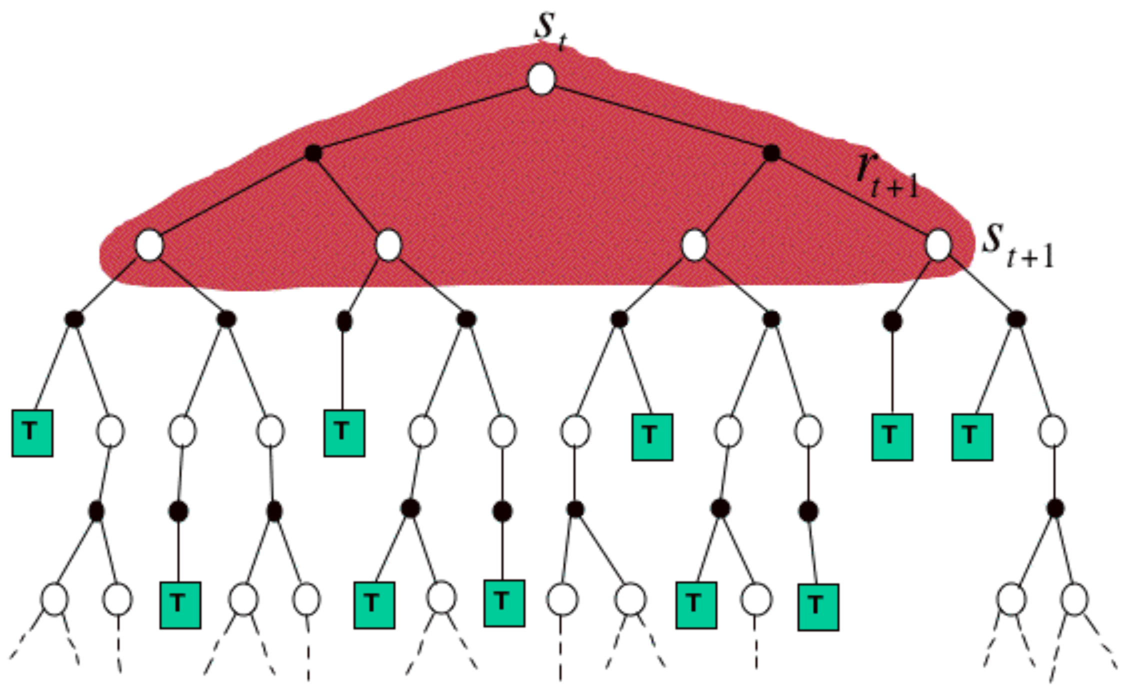


- You are learning from a long stream of experience:
  $$s_0 a_0 r_0 s_1 a_1 r_1 \ldots s_k a_k r_k \ldots$$
  … up to some terminal state

- Direct methods:
  Approximate value function (V/Q) straight away - without computing $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$

Pictorial: What does DP Do?

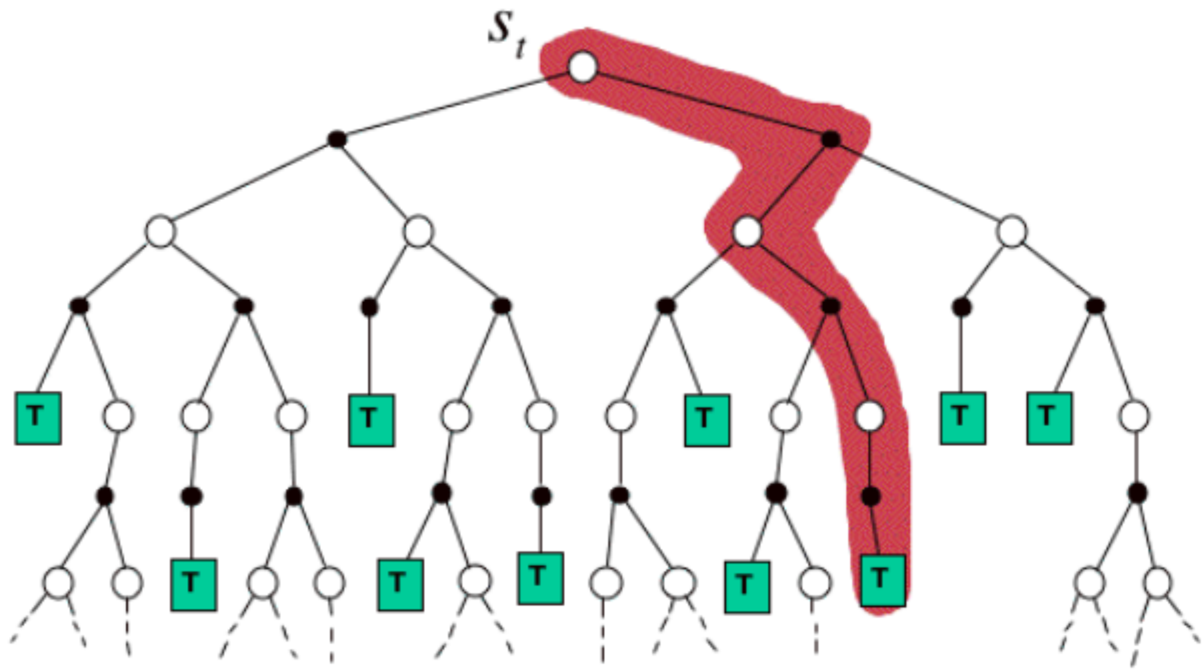
$$V(s_t) \leftarrow E_\pi \{ r_{t+1} + \gamma V(s_t) \}$$



Pictorial: What does Simple MC Do?

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual return following state s_t.



Monte Carlo Policy Evaluation

- Goal: Approximate a value function $V^{\pi}(s)$
- Given: Some number of episodes under π which contain s
- Maintain average returns after visits to s

What is the effect of π?
What if it is deterministic?

- First visit vs. Every visit MC:
  - Consider a reward process $R(t) = r_t + \gamma r_{t+1} + ...$ and define the first visit time, $\tau = \min\{t | x = x_i\}$ and a set, $\Gamma = \{t | x = x_i\}$
  - First visit MC averages $\{R^i(\tau)\}, i = 1, ..., n$
    whereas every visit MC averages over $\{R^i(t_j)\}, i = 1, ..., n, t_j \in \Gamma$

First-visit Monte Carlo Policy Evaluation

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

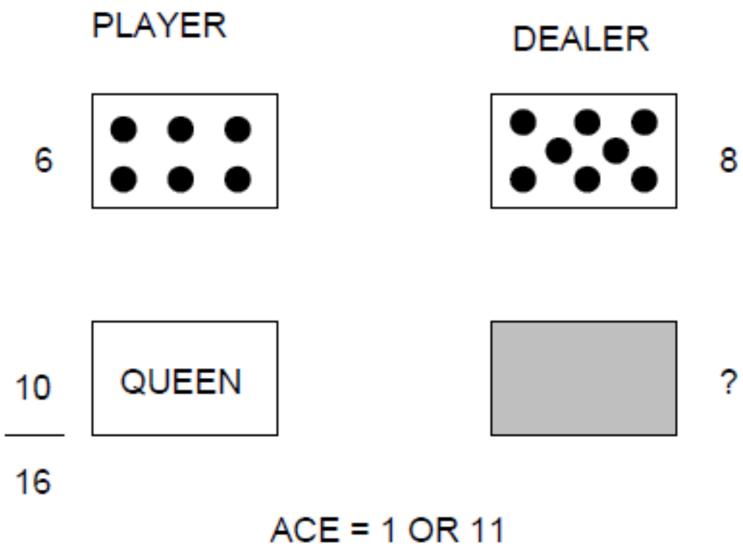
Example: Blackjack

- **Goal**: Achieve a card sum greater than dealer without exceeding 21
- Player's options: Hit (take another card) or Stick (pass)
  - If player crosses 21 - loss
- Dealer follows simple rule:

  Stick if ≥ 17, else Hit
- **Result**:

  Closest to 21 wins

  Equally close is a draw

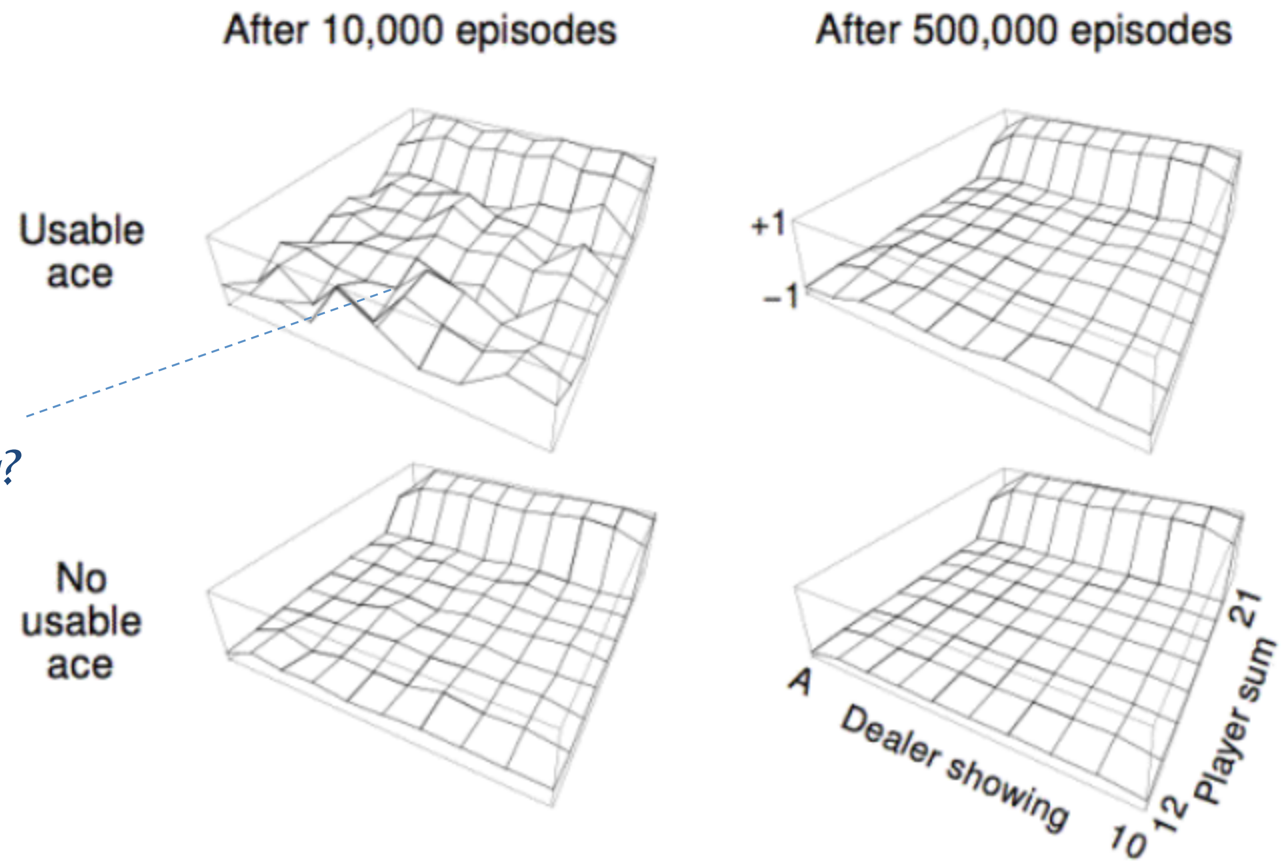


Example: Blackjack

- **Goal**: Achieve a card sum greater than dealer without exceeding 21
- State space: (200 states)
  - Current sum (12 – 21)
  - Dealer's showing card (ace - 10)
  - Do I have a usable ace (can be used as 11 without overshoot)?
- Reward: +1 for win, 0 for loss, -1 for a loss
- Action space: *stick* (no more cards), *hit* (receive another card)
- Policy: *stick* if sum is 20 or 21, else *hit*

  Note: This is an (arbitrary) policy π with which algorithm works

Solution ($V^\pi(s)$) : Blackjack



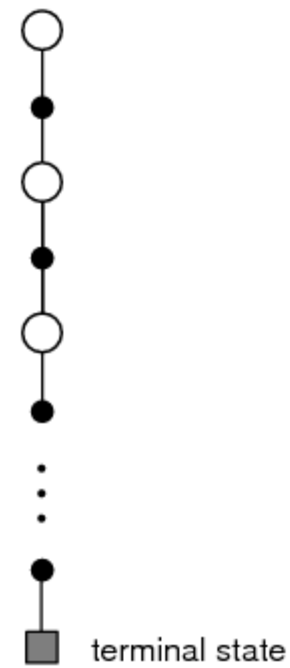
Why is this more choppy?

Remarks on Blackjack Example

- Why does the value function jump up for the last two rows in the rear?
  - When sums correspond to 20 or 21, policy is to *stick*; this is a good choice in this region of state space
- Why does it drop off for the whole last row on the left?
  - Dealer is showing an ace, which gives him extra flexibility (two chances to get close to 21)
- Why are the foremost values higher on upper plots than lower plots?
  - Player has usable ace (more flexibility)

Backup in MC

- Does the concept of backup diagram make sense for MC methods?
- As in figure, MC does not sample all transitions
  - Root node to be updated as before
  - Transitions are dictated by policy
  - Acquire samples along a sample path
  - Clear path from eventual reward to states along the way (credit assignment easier)
- Estimates are different states are independent
  - Computational complexity **not** a function of state dimensionality



Monte Carlo Estimation of Action Values

- Model is not available, so we do not know how states and actions interact
  - We want Q^*

- We can try to approximate $Q^{\pi}(s,a)$ using Monte Carlo method
  - Asymptotic convergence if every state-action pair is visited

- Explore many different starting state-action pairs: Equal chance of starting from any given state
  - Not entirely practical, but simple to understand

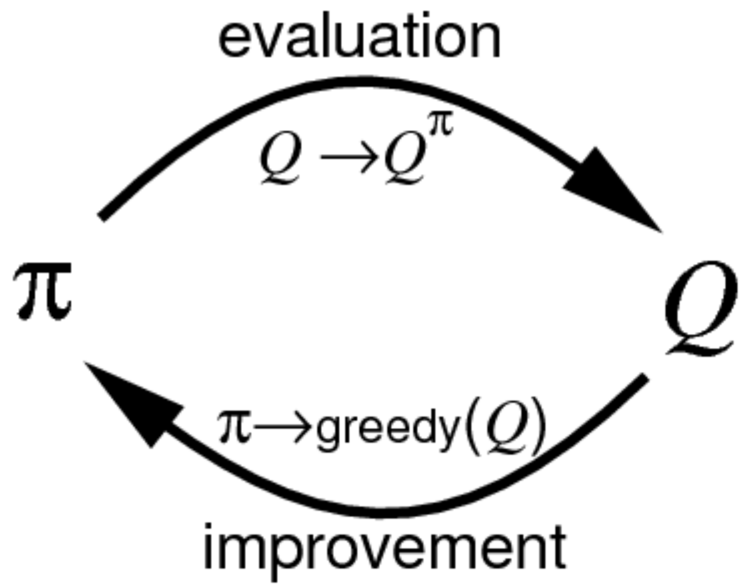
Monte Carlo Control

- Policy Evaluation:

  Monte Carlo method

- Policy Improvement:

  Greedify with respect to state-value of action-value function



Convergence of MC Control

- Policy improvement still works if evaluation is done with MC:

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s,a)) \\ &= \max_a Q^{\pi_k}(s,a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \\ &= V^{\pi_k}(s). \end{aligned}$$

- $\pi_{k+1} \geq \pi_k$ by the policy improvement theorem
- Assumption: exploring starts and infinite number of episodes for MC policy evaluation (i.e., value function has stabilized)
- Things to do (as in DP):
  - update only to given tolerance
  - interleave evaluation/improvement

Monte Carlo Exploring Starts

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list

Fixed point is optimal policy π^*

Repeat forever:
  (a) Generate an episode using exploring starts and π
  (b) For each pair s, a appearing in the episode:
    $R \leftarrow$ return following the first occurrence of s, a
    Append R to $Returns(s,a)$
    $Q(s,a) \leftarrow$ average$(Returns(s,a))$
  (c) For each s in the episode:
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Blackjack Example – Optimal Policy

Exploring starts

Initial policy as described before