

First Steps in Reinforcement Learning

Policy Iteration and Monte Carlo Learning
Volker Krüger

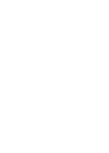


LUND
UNIVERSITY

Overview

This and the next lecture:

- How can the state value and the action value functions be computed?
 - **What do we need to know for doing this?**



LUND
UNIVERSITY

Overview

This and the next lecture:

- How can the state value and the action value functions be computed?
 - **What do we need to know for doing this?**
- How can we find an optimal policy?



Overview

This and the next lecture:

- How can the state value and the action value functions be computed?
 - **What do we need to know for doing this?**
- How can we find an optimal policy?
- Start with a first simple solution: **Iterative Policy Evaluation** and **Policy Iteration**



Overview

This and the next lecture:

- How can the state value and the action value functions be computed?
 - **What do we need to know for doing this?**
- How can we find an optimal policy?
- Start with a first simple solution: **Iterative Policy Evaluation** and **Policy Iteration**
- Then: first steps in **Model-free Prediction and Control**:
 - Monte Carlo Learning, TD Learning, SARSA



Overview

This and the next lecture:

- How can the state value and the action value functions be computed?
 - **What do we need to know for doing this?**
- How can we find an optimal policy?
- Start with a first simple solution: **Iterative Policy Evaluation** and **Policy Iteration**
- Then: first steps in **Model-free Prediction and Control**:
 - Monte Carlo Learning, TD Learning, SARSA
- On-policy control vs off-policy control
 - Q-Learning



Plan for Today

- Revisit MDPs
- Iterative Policy Evaluation
- Policy Iteration, Modified Policy Iteration, Generalised Policy Integration
- Value Iteration
- Monte Carlo Learning,
 - On-Policy Monte Carlo Control



Revisiting Markov Decision Processes



LUND
UNIVERSITY

Markov Decision Process

- A Markov Decision Process is a Markov Reward Process with decisions on actions

Definition: A **Markov Decision Process** is a tupel (S, A, P, R, γ)

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix $P_{ss'}^a = P(S_{t+1} = s' | S_t = s, A_t = a)$
- R is a reward function $R_s^a = E(R_{t+1} | S_t = s, A_t = a)$
- γ is a discount factor $\gamma \in [0,1]$



Value Functions

Definition: The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s and following policy π

$$v_\pi(s) = E_\pi \{ G_t | S_t = s \} = E_\pi \{ R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \}$$



Value Functions

Definition: The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s and following policy π

$$v_\pi(s) = E_\pi \{ G_t | S_t = s \} = E_\pi \{ R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \}$$

Definition: The action-value function $q_\pi(s, a)$ of an MDP is the expected return starting from state s by taking action a and following policy π

$$q_\pi(s, a) = E_\pi \{ G_t | S_t = s, A_t = a \} = E_\pi \{ R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a \}$$

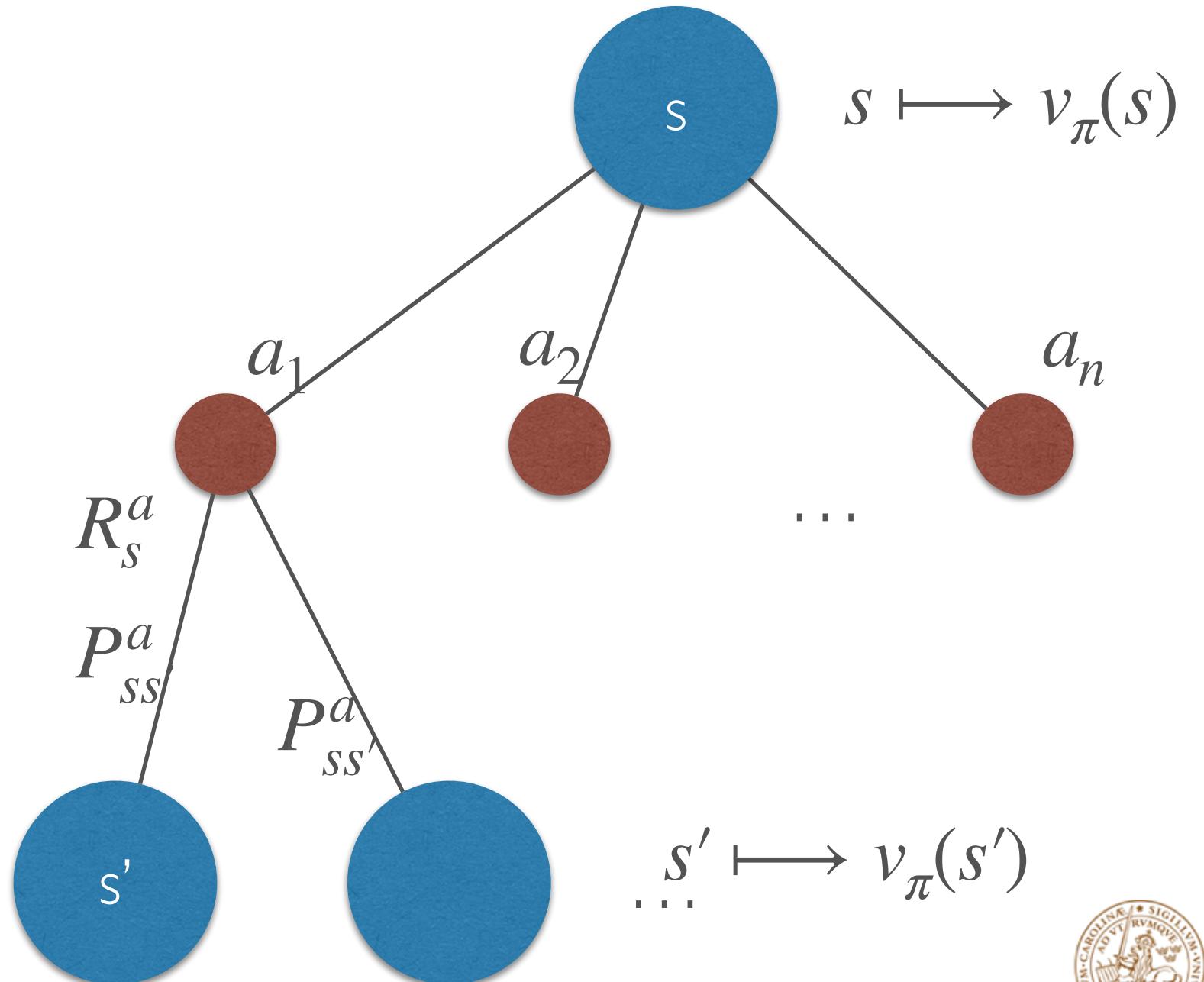


Bellman Expectation Equation

- From state s , we can take different actions $a_i \in \mathbf{A}$ chosen according to $\pi(a | s)$

- Based on s and a we get a reward R_s^a

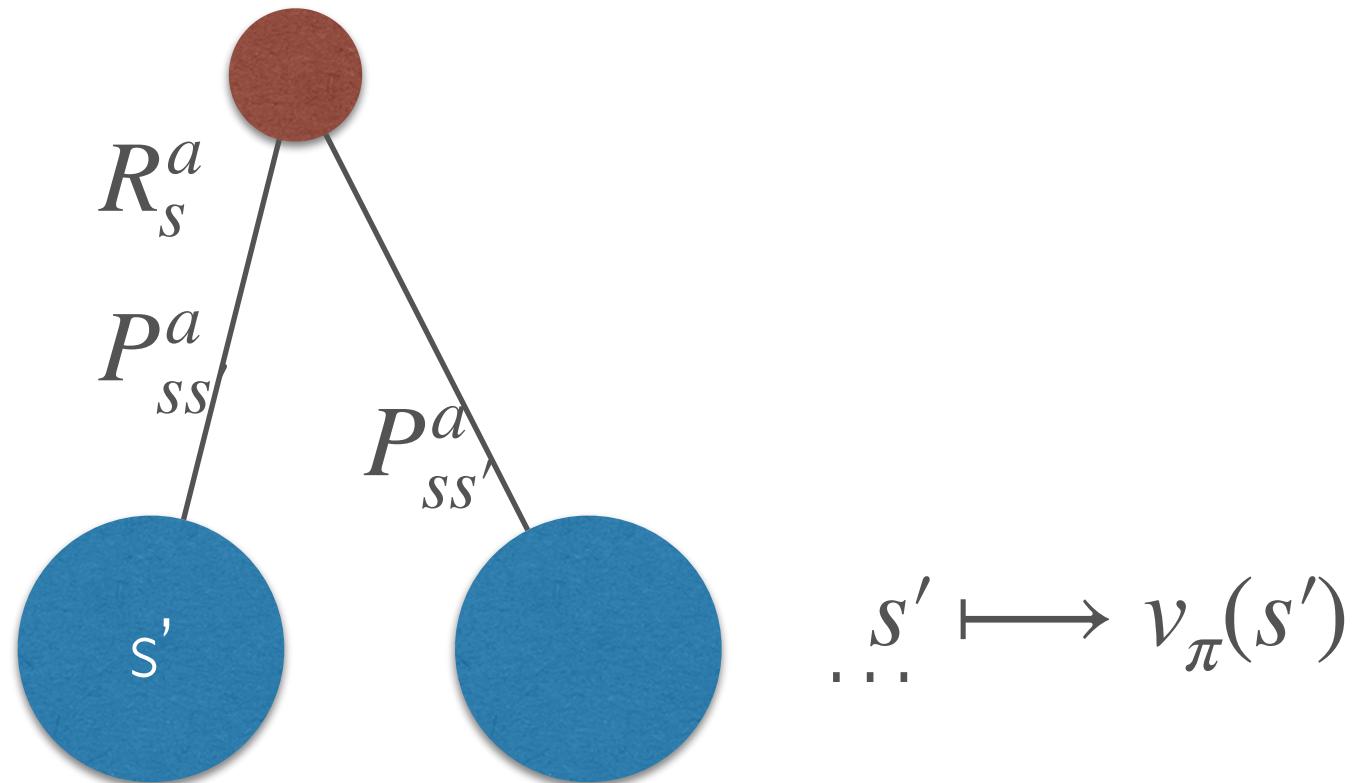
$$\nu_{\pi}(s) = \sum_{a \in \mathbf{A}} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a \nu_{\pi}(s') \right)$$



Bellman Expectation Equation

- From state s , we can take different actions $a_i \in \mathbf{A}$ chosen according to $\pi(a | s)$
- Based on s and a we get a reward R_s^a

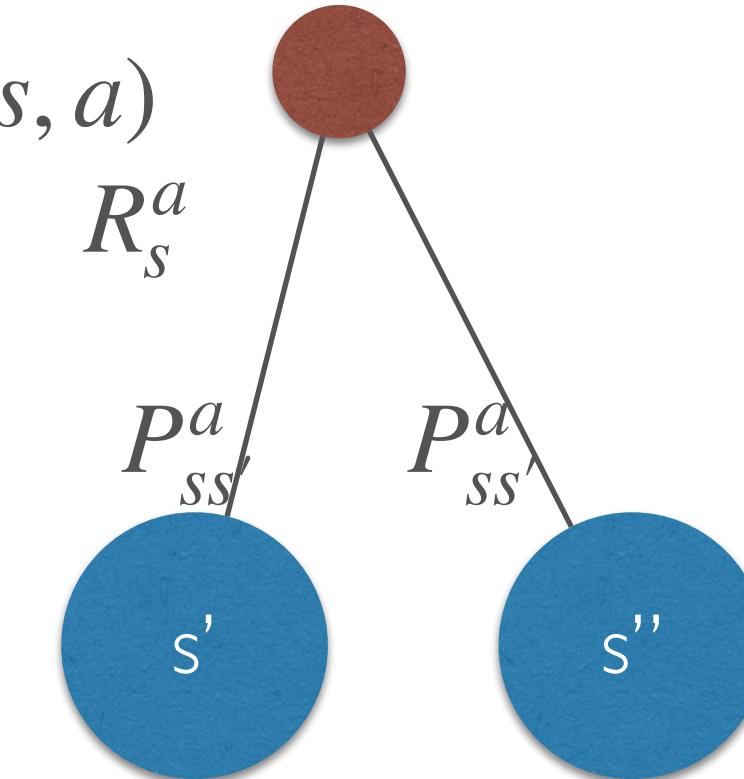
$$v_\pi(s) = \sum_{a \in \mathbf{A}} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$$



Bellman Expectation Equation

$$s, a \mapsto q_{\pi}(s, a)$$

$$R_s^a$$



$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' | s') q_{\pi}(s', a')$$

$$s', a_i \mapsto q_{\pi}(s', a_i)$$

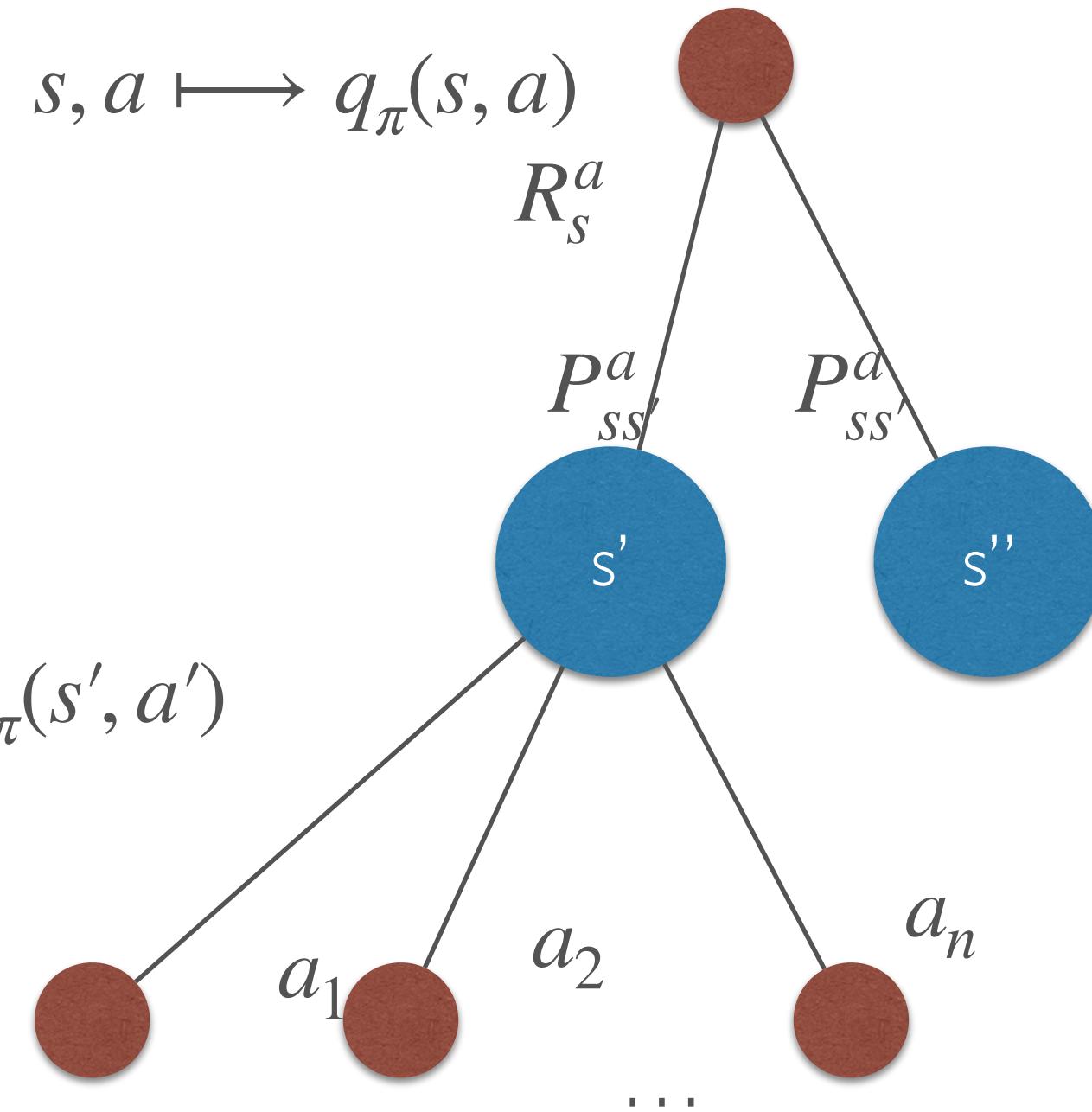


LUND
UNIVERSITY

Bellman Expectation Equation

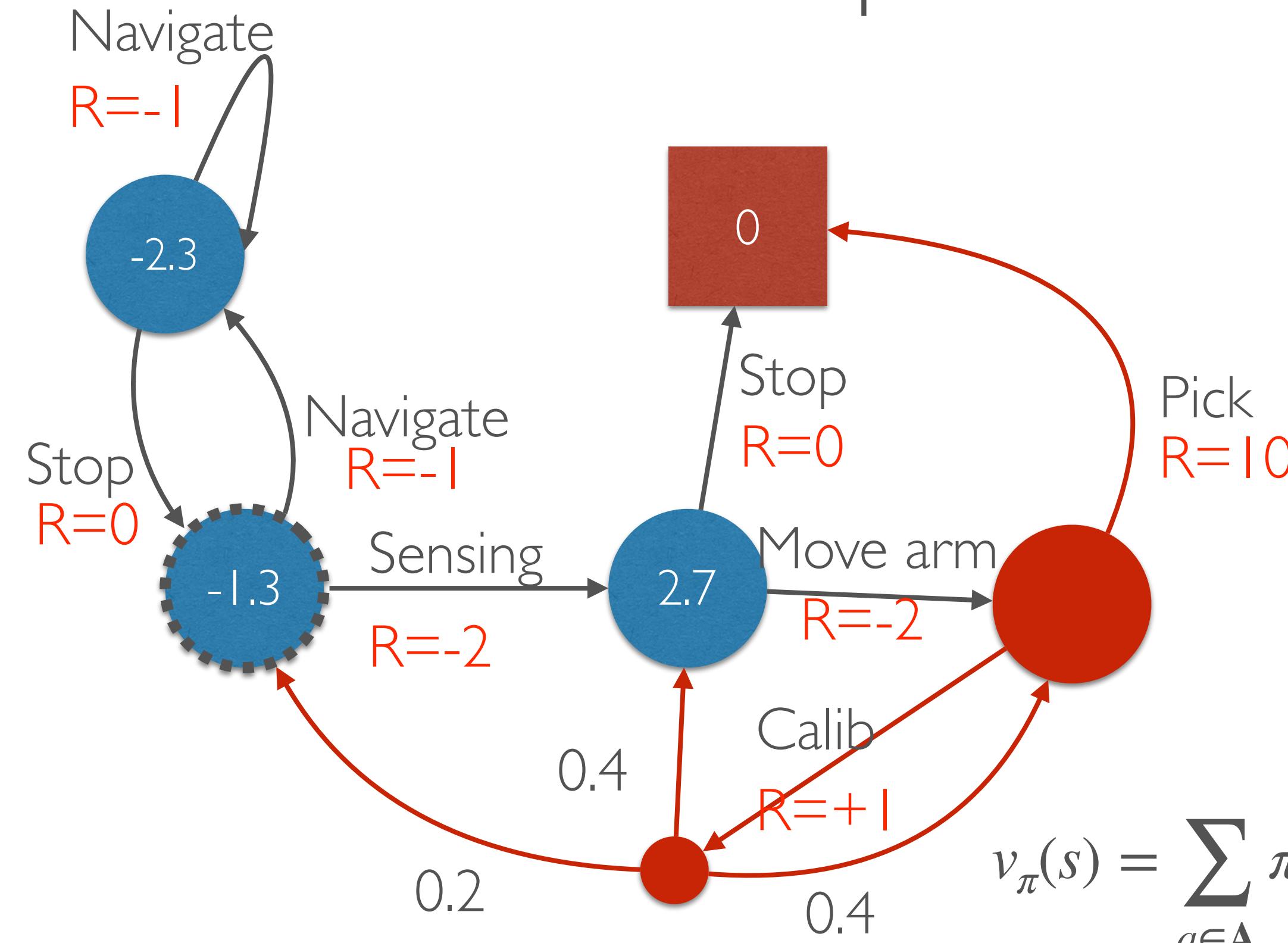
$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' | s') q_{\pi}(s', a')$$

$$s', a_i \longmapsto q_{\pi}(s', a_i)$$



Bellman expectation Equation

$$v_{\pi}(s) \text{ for } \pi(s, a) = 0.5, \gamma = 1$$

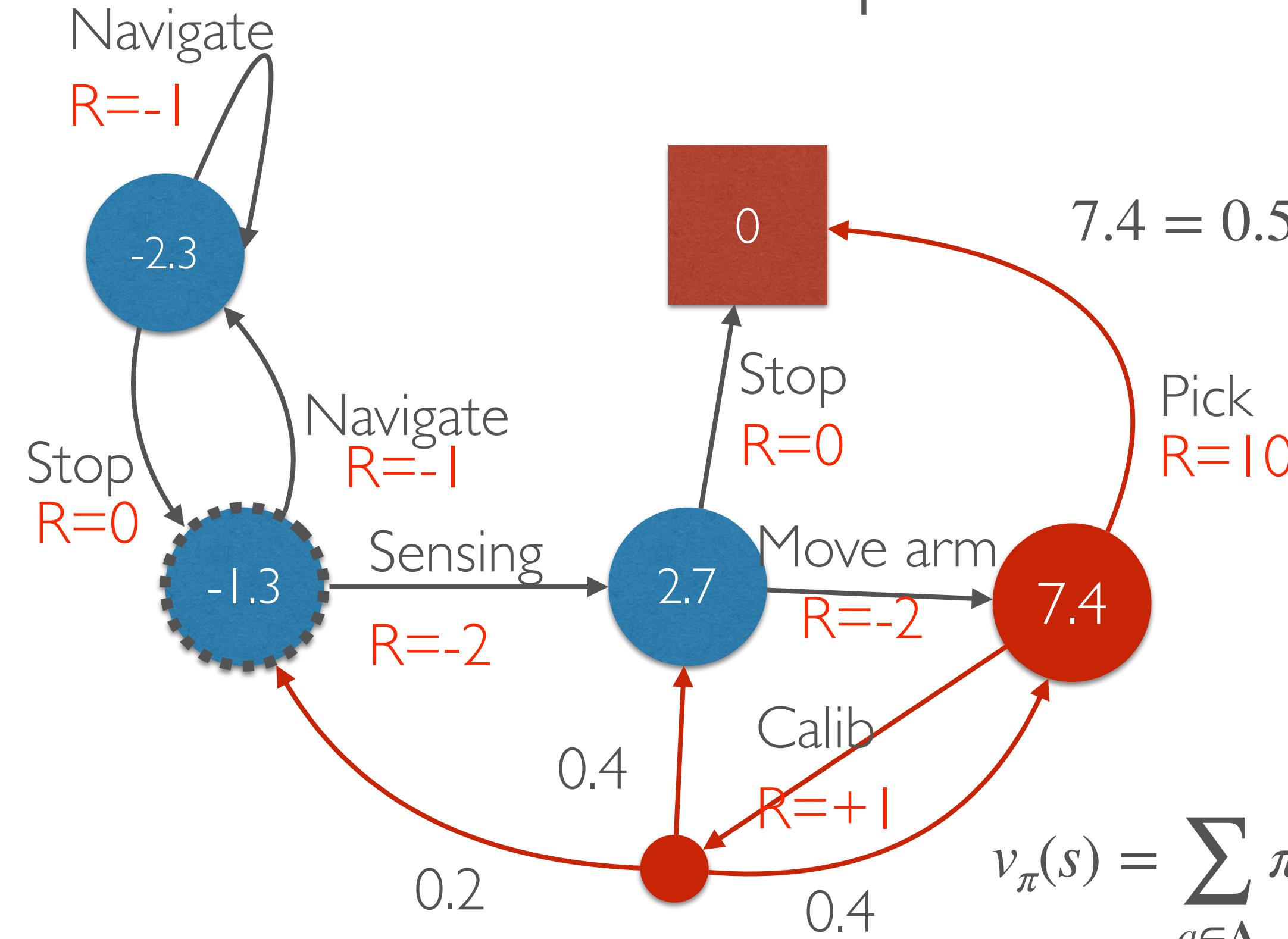


$$v_{\pi}(s) = \sum_{a \in A} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s') \right)$$

Bellman expectation Equation

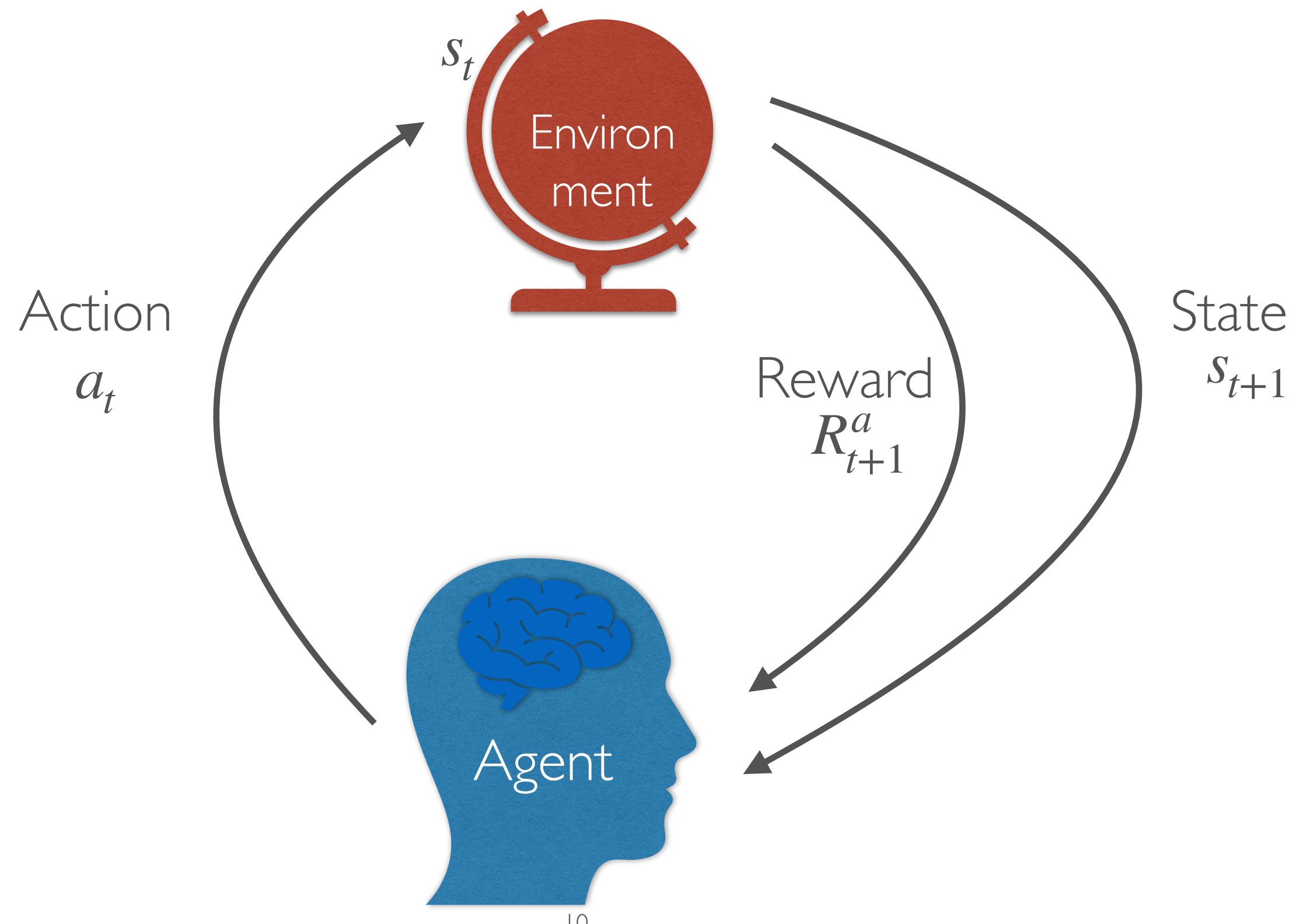
$$v_{\pi}(s) \text{ for } \pi(s, a) = 0.5, \gamma = 1$$

$$7.4 = 0.5 \cdot (1 + 0.2 \cdot (-1.3) + 0.4 \cdot 2.7 + 0.4 \cdot 7.4) + 0.5 \cdot 10$$



$$v_{\pi}(s) = \sum_{a \in A} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s') \right)$$

Markov Decision Process



Jack's Car Rental



- States: Two locations, maximum 20 cars at each location
- Actions: Move up to 5 cars between locations overnight
- Reward: 10€ for each car rented if available
- Transitions: Cars returned and requested randomly
 - Poisson distribution, n returns/requests with probability $\frac{\lambda^n}{n!} e^{-\lambda}$
 - 1st location: average requests=3, average returns=3
 - 2nd location: average requests=4, average returns =2



Next steps

- We have given a fully defined MDP ($\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma$)
- We have also given a policy π
- We want to know the state values $v_\pi(s)$ of each state s for the policy.
 - This is called **Prediction** of how good the plan / policy is.
- Then, we want to know π^* , i.e., the optimal policy for the MDP
 - This is called **Control**.
- **Planning** in an MDP: How do we achieve a goal within the MDP most efficiently?



Iterative Policy Evaluation

Prediction: Finding v_π



LUND
UNIVERSITY

The trick: Dynamic programming

- Optimization approach for complex problems that can be broken down into sub-problems.
- Approach:
 - Solve sub-problem
 - Combine solution from subproblem to solve the complete problem
- DP can be used to solve an MDP
- requires knowledge of the full MDP

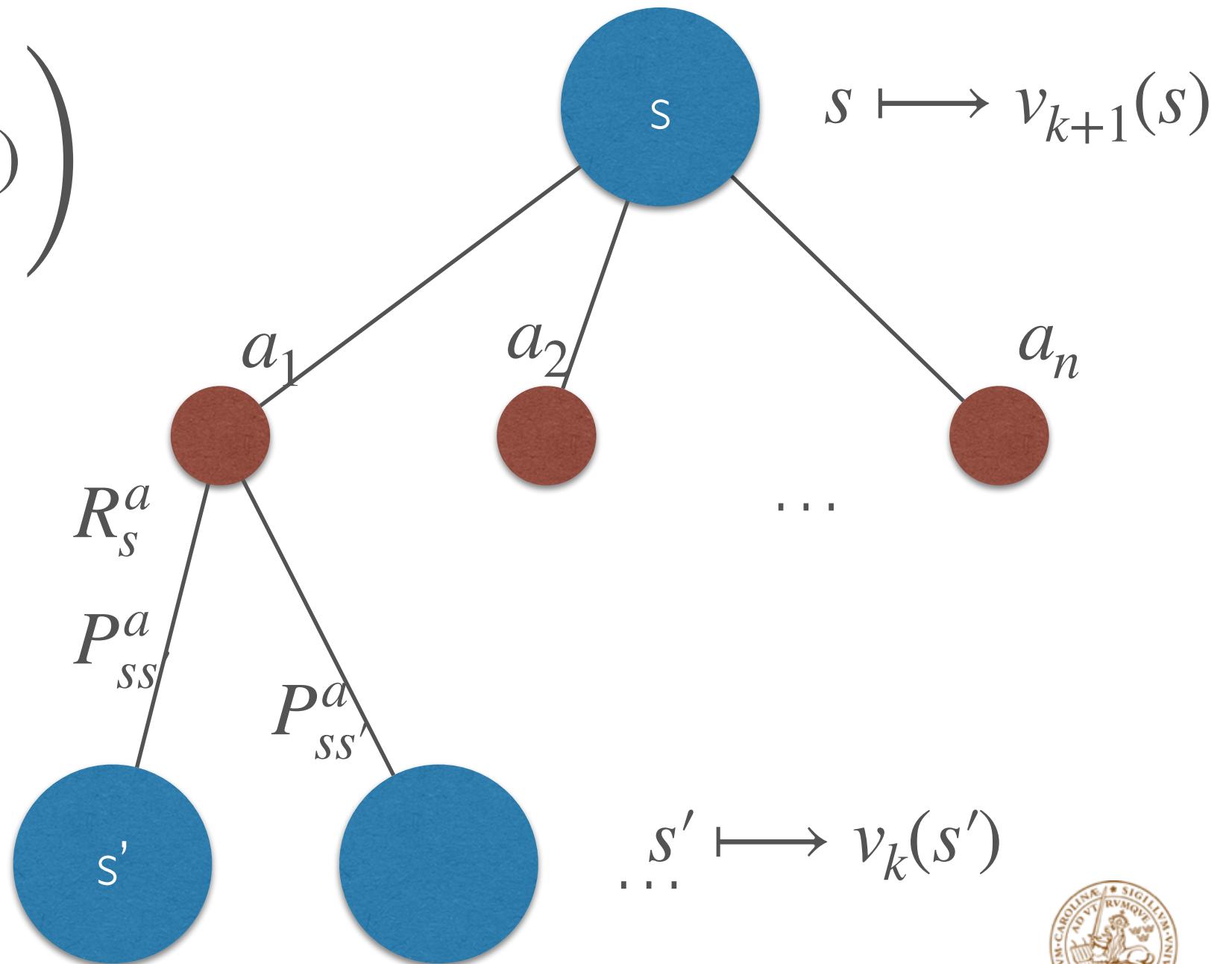


Iterative Policy Evaluation

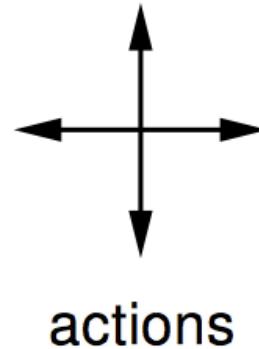
- Goal: evaluate a given policy π
- Solution: Iterative application of the Bellman expectation equation $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_\pi$
- Initialise all state value to 0: $v_1(s) = 0$ for all $s \in S$
- At each iteration $k + 1$
 - For all states $s \in S$
 - Update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s

$$v_{k+1}(s) = \sum_{a \in A} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

$$v_{k+1}(s) = \sum_{a \in A} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$



Example: Random Policy in the Small Gridworld



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$$r = -1 \quad \gamma = 1$$

for all transition

- Non-terminal states 1...14
- terminal state in grey
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n, \cdot) = \pi(e, \cdot) = \pi(s, \cdot) = \pi(w, \cdot)$$

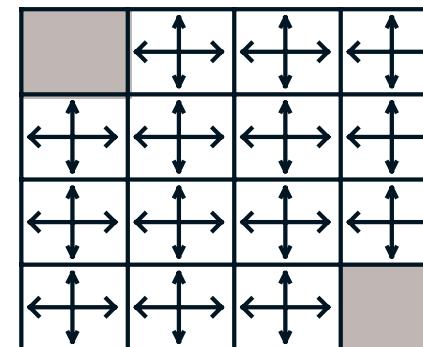
Example: Random Policy in the Small Gridworld

v_k for the
Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Greedy Policy
w.r.t. v_k



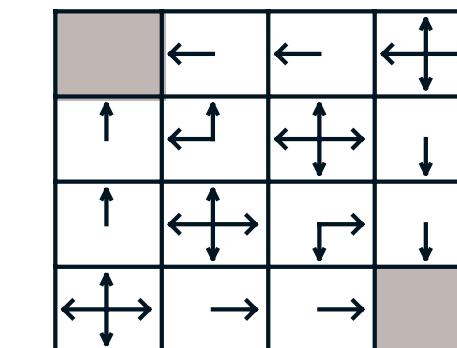
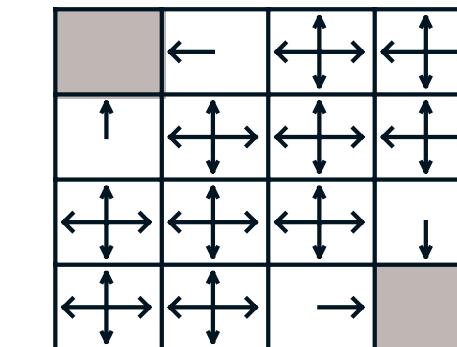
random
policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Example: Random Policy in the Small Gridworld

$k = 3$

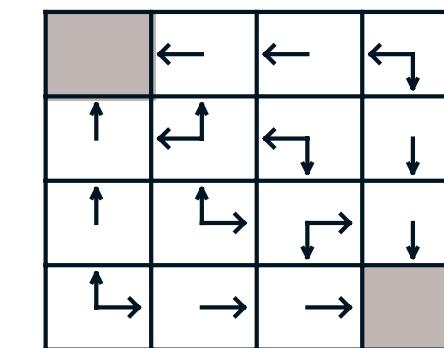
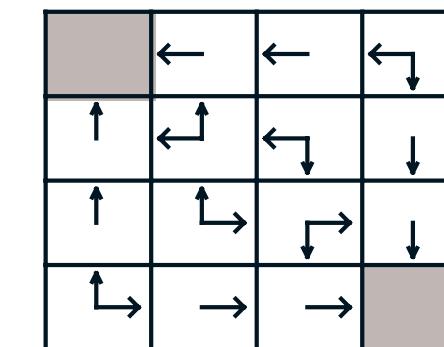
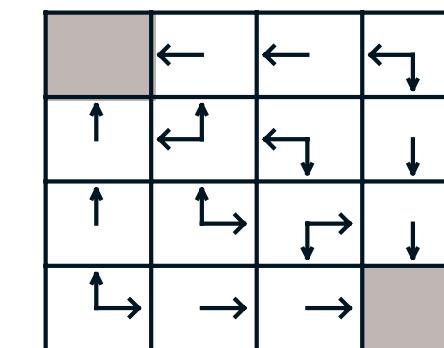
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



optimal
policy

How To Improve the Policy

Policy Iteration



LUND
UNIVERSITY

How to improve the Policy

- Given a policy.
 - Evaluate the policy

$$v_{\pi} = E \left\{ R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s \right\} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$



LUND
UNIVERSITY

How to improve the Policy

- Given a policy.
 - Evaluate the policy

$$v_{\pi} = E \left\{ R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s \right\} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s') \right)$$

- Improve the policy by acting greedily with respect to v_{π}

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



LUND
UNIVERSITY

How to improve the Policy

- Given a policy.
 - Evaluate the policy

$$v_{\pi} = E \{ R_{t+1} + \gamma R_{t+2} + \dots | S_t = s \} = \sum_{a \in A} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

- Improve the policy by acting greedily with respect to v_{π}

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

$$\pi'(s) = \arg \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$



How to improve the Policy

- Given a policy.
 - Evaluate the policy

$$v_{\pi} = E \left\{ R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s \right\} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

- Improve the policy by acting greedily with respect to v_{π}

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

$$\pi'(s) = \arg \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

- In the Gridworld the improved policy was optimal $\pi' = \pi^*$



How to improve the Policy

- Given a policy.
 - Evaluate the policy

$$v_{\pi} = E \left\{ R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s \right\} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

- Improve the policy by acting greedily with respect to v_{π}

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

$$\pi'(s) = \arg \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

- In the Gridworld the improved policy was optimal $\pi' = \pi^*$
- In general, more iterations are needed



How to improve the Policy

- Given a policy.
 - Evaluate the policy

$$v_{\pi} = E \left\{ R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s \right\} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

- Improve the policy by acting greedily with respect to v_{π}

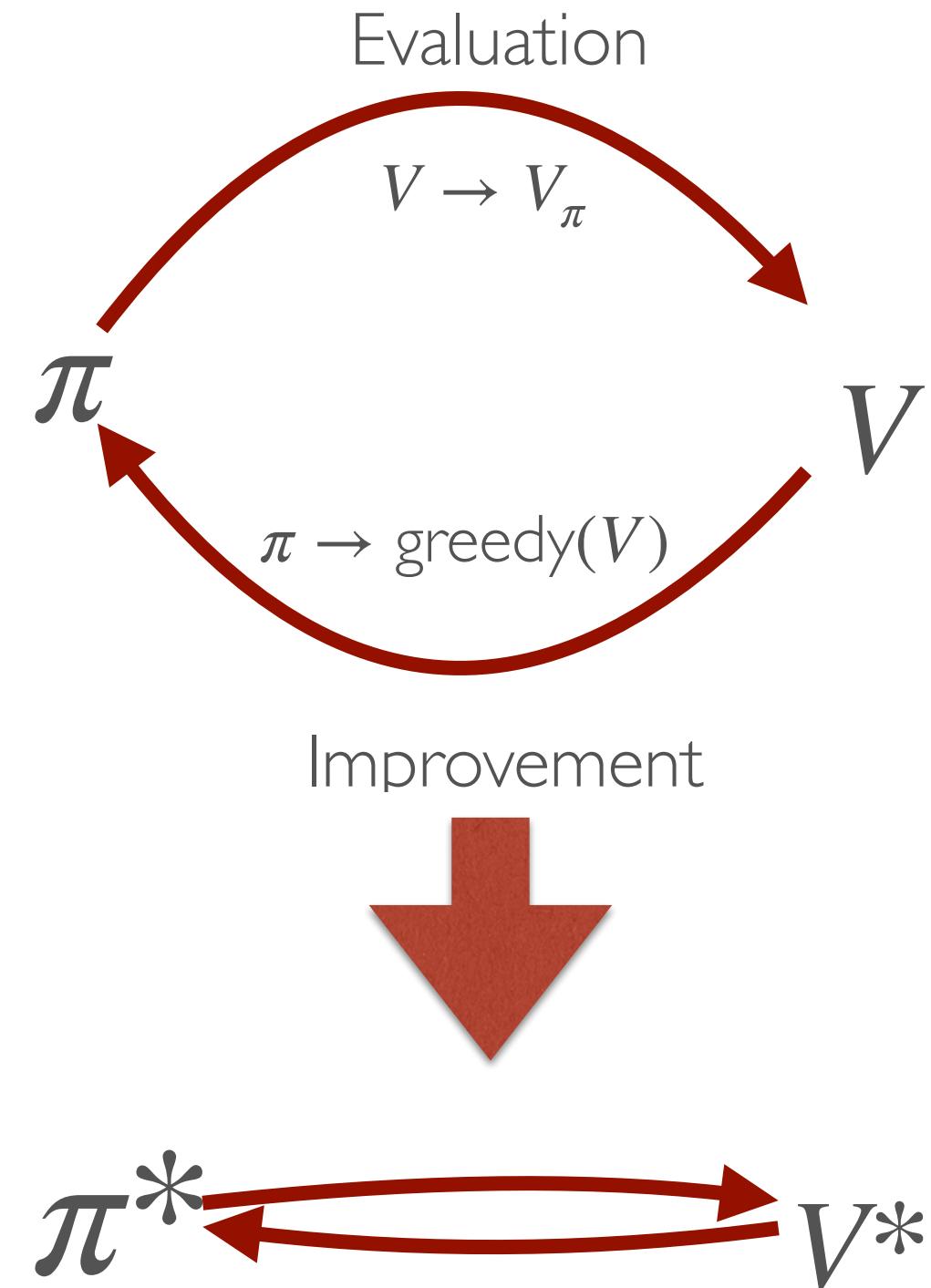
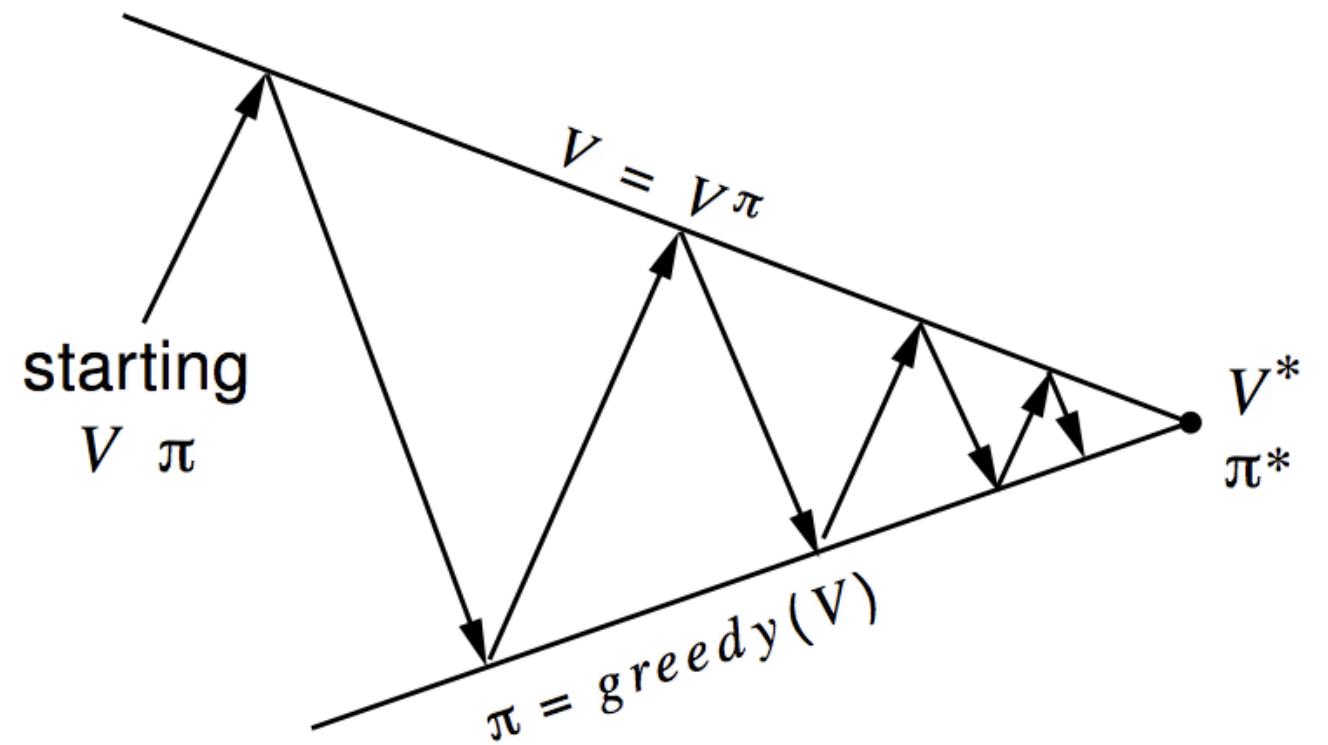
$$\pi' = \text{greedy}(v_{\pi})$$

$$\pi'(s) = \arg \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

- In the Gridworld the improved policy was optimal $\pi' = \pi^*$
- In general, more iterations are needed
- But this process of **policy iteration** always converges to π^*



Policy Iteration



- **Policy Evaluation:** Estimate v_π
 - Iterative policy evaluation
- **Policy improvement:** Generate $\pi' \geq \pi$
 - Greedy policy improvement

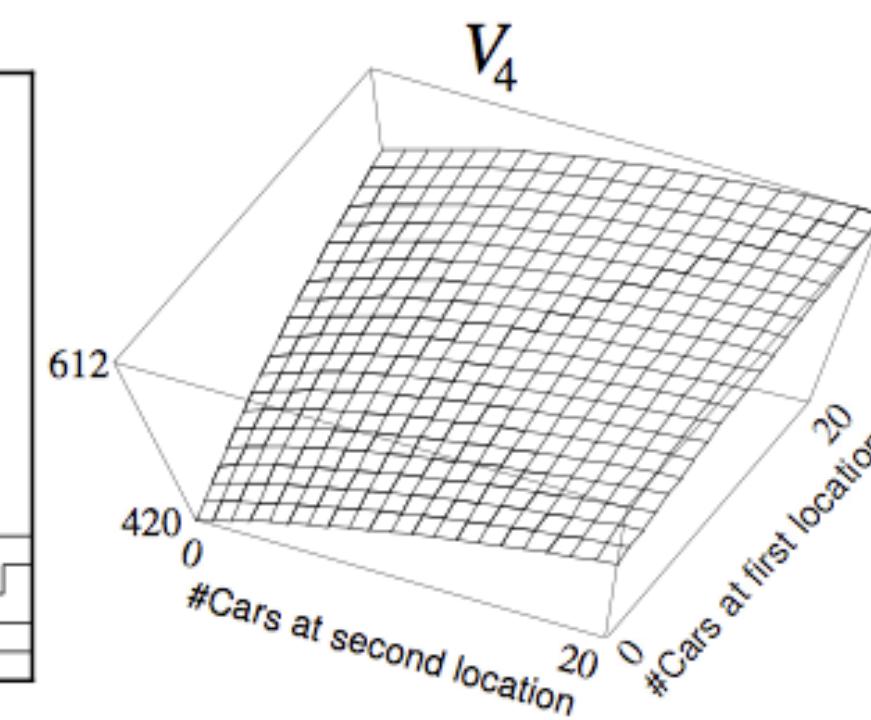
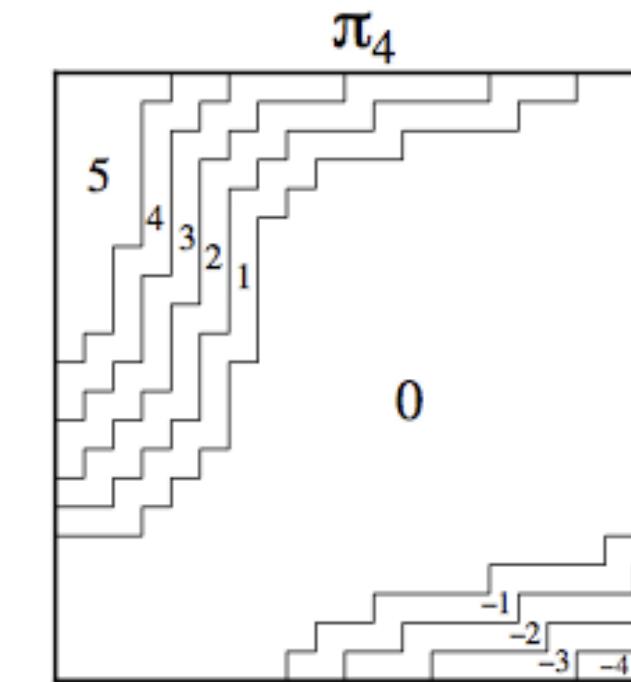
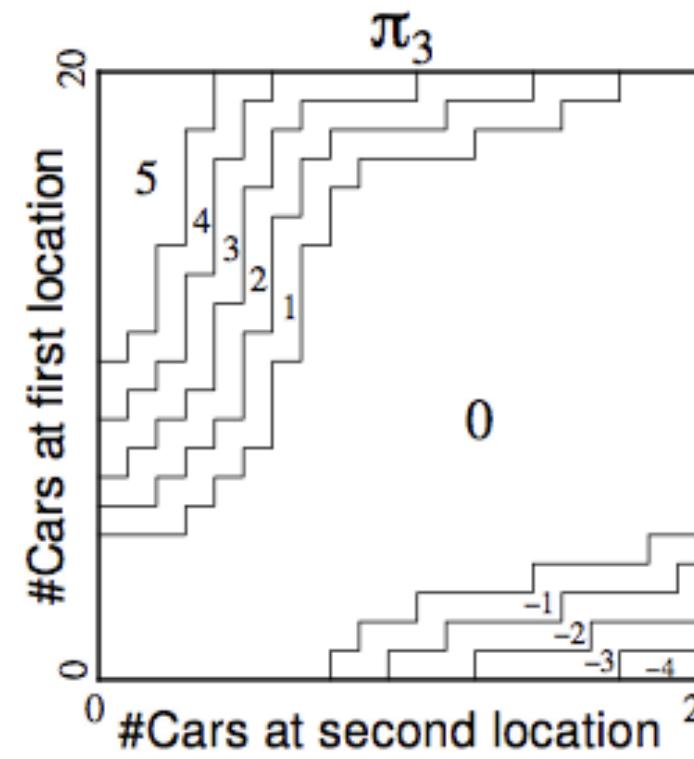
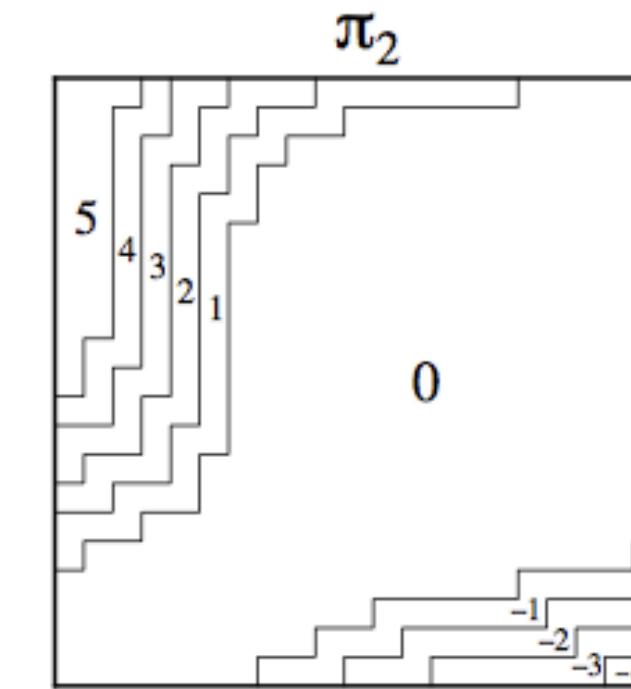
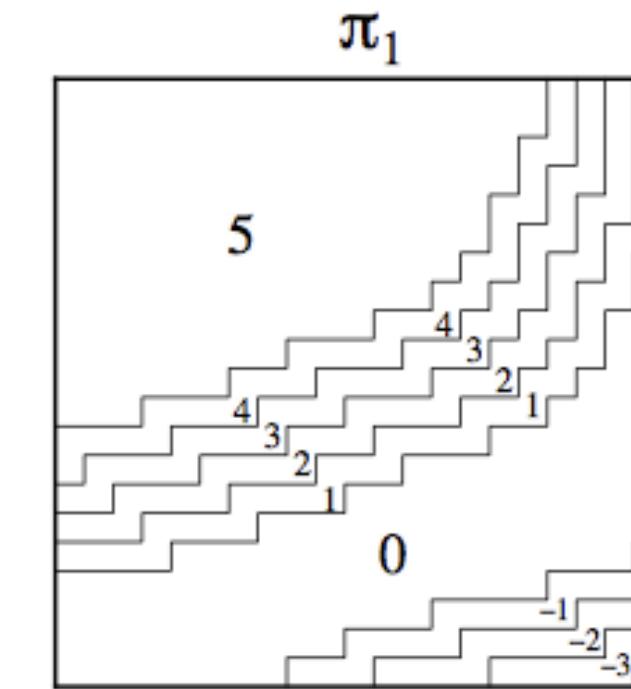
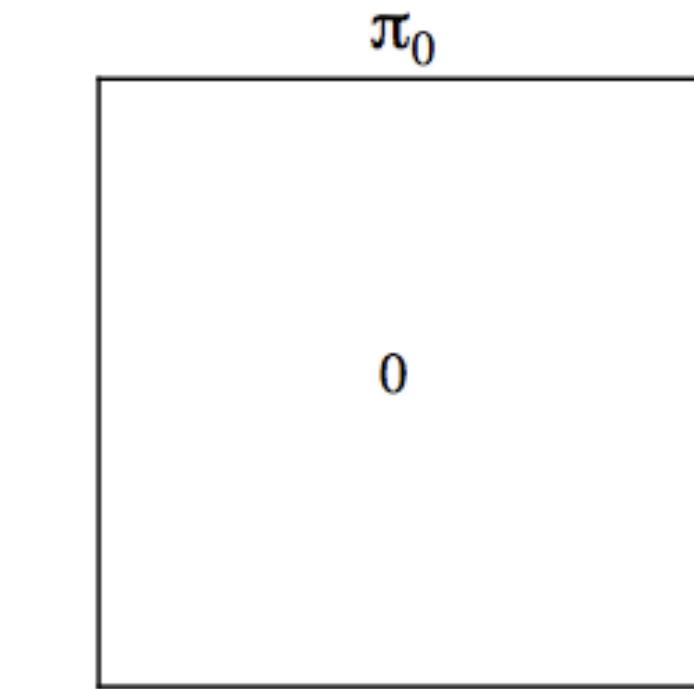
Jack's Car Rental



- States: Two locations, maximum 20 cars at each location
- Actions: Move up to 5 cars between locations overnight
- Reward: 10€ for each car rented if available
- Transitions: Cars returned and requested randomly
 - Poisson distribution, n returns/requests with probability $\frac{\lambda^n}{n!} e^{-\lambda}$
 - 1st location: average requests=3, average returns=3
 - 2nd location: average requests=4, average returns =2



Policy Iteration in Jack's Car Rental



Policy Improvement

Consider a deterministic policy $a = \pi(s)$

We can *improve* the policy by acting greedily $\pi'(s) = \arg \max_{a \in A} q_\pi(s, a)$

This improves the value from any state s over one step

Blackboard



- If improvement stops

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

- Bellman optimality equation has been satisfied

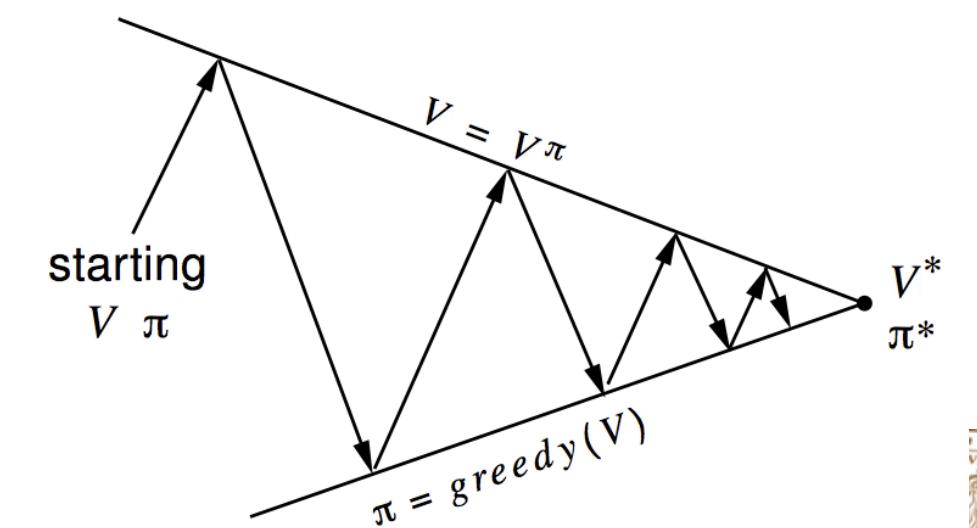
$$v_{\pi}(s) = \max_{a \in A} q_{\pi}(s, a)$$

- Thus, $v_{\pi}(s) = v^*(s)$ for all $s \in S$
- and π is the optimal policy.



Some amazing facts

- Policy evaluation always converges to v_π
- Policy Iteration always converges to π^*
- synchronous approach used above.
- Asynchronous approach converges as well if all states are selected
- How do we know we have reached v_π ?
- How do we know we have reached π^* ?
 - How many repetitions k ?
 - Simply stop after $k = 1$ and update after each iteration
 - This is equivalent to **value iteration**



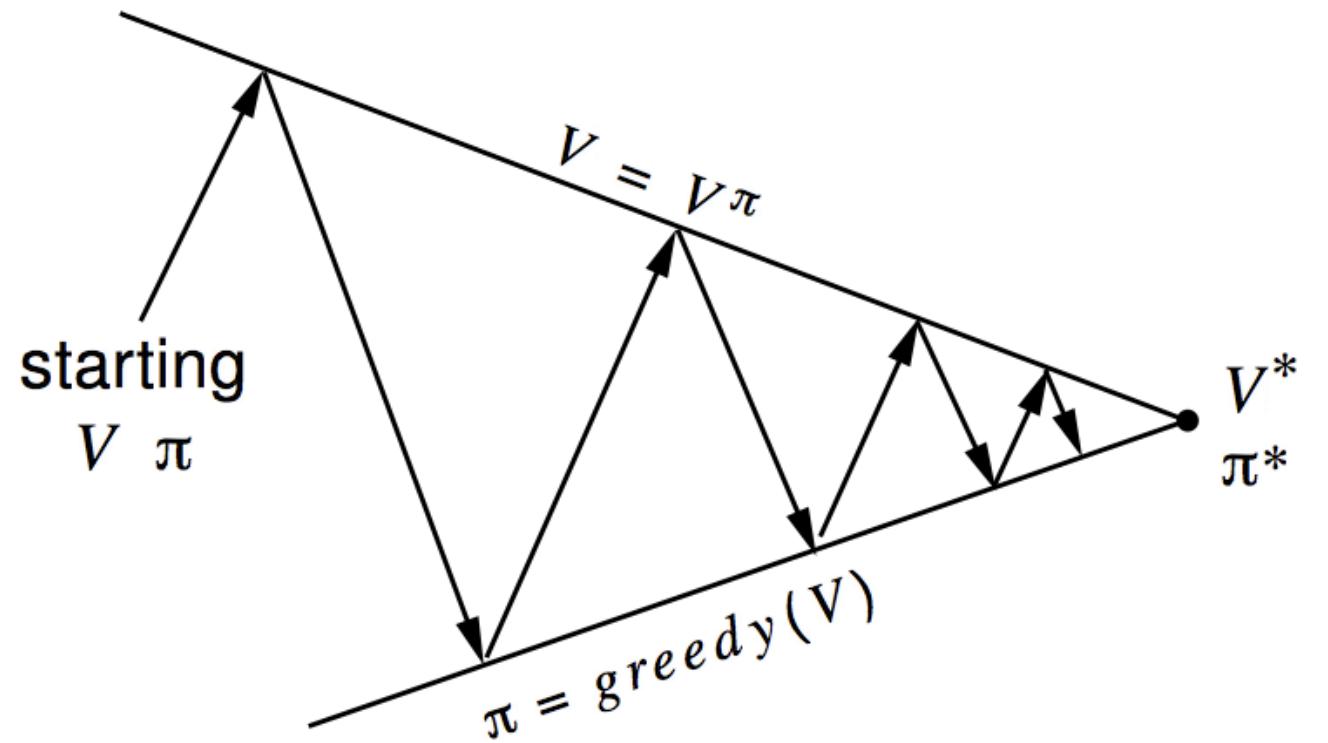
Value Iteration

- Goal: find the optimal policy π
- Solution: Iterative application of the Bellman expectation equation $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Initialise all state value to 0: $v_1(s) = 0$ for all $s \in S$
- At each iteration $k + 1$
 - For all states $s \in S$
 - Update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s
- Note: there is no(!!) explicit policy here.
- Intermediate value functions might not correspond to any policy.

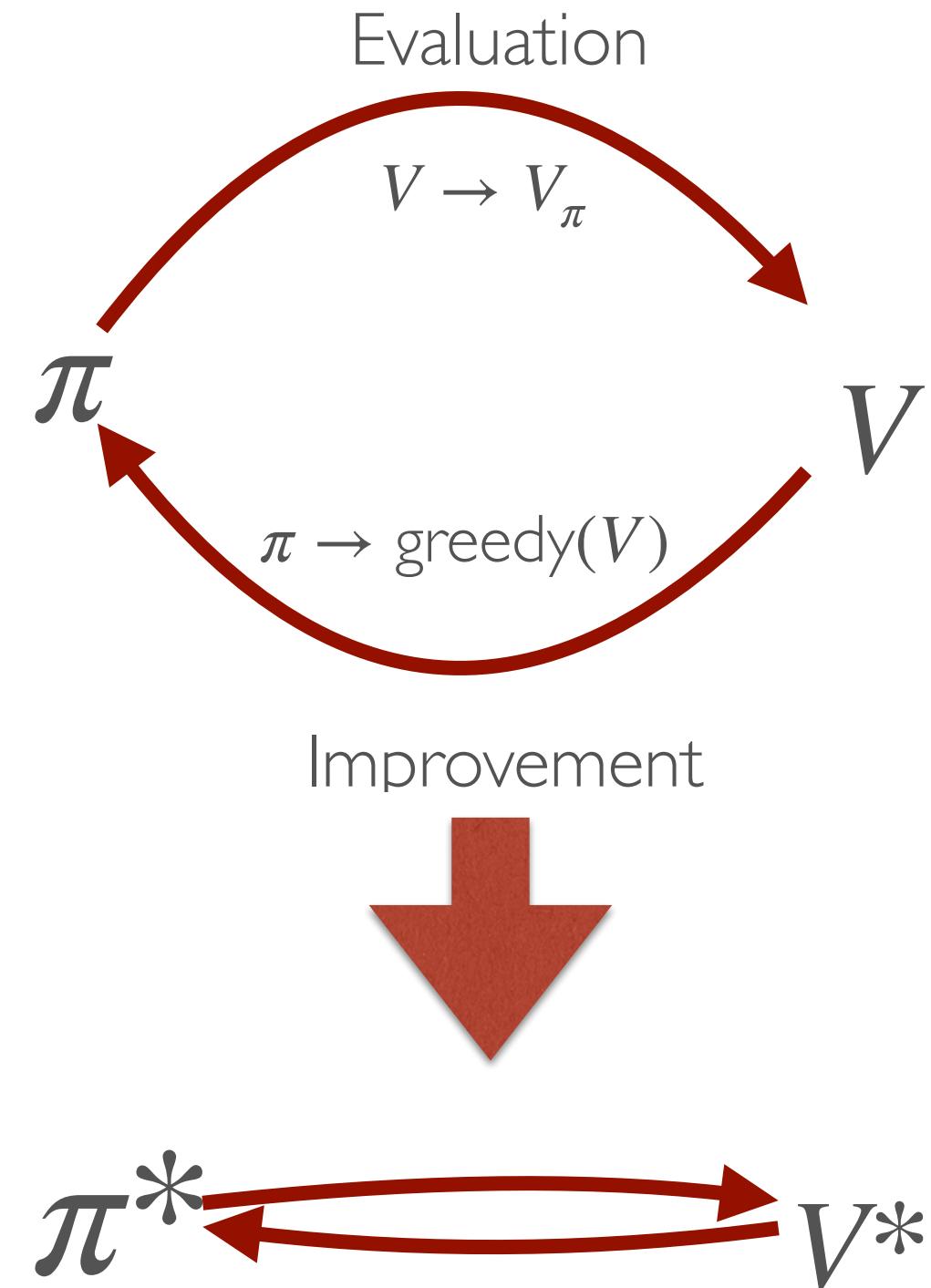
$$v_{k+1}(s) = \max_a \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$



Generalized Policy Iteration



- **Policy Evaluation:** Estimate v_π
 - **Any** policy evaluation approach
- **Policy improvement:** Generate $\pi' \geq \pi$
 - **Any** policy improvement approach



Summary so far

- Identify the optimal policy, state value function and action value function
- **Policy Iteration** and **Value Iteration** for finding the optimal policy.
 - Very simple approach, but still finds π^*
- The MDP needs to be known completely
 - The transition probabilities $P_{s,s'}^a$, are usually not known
 - It's not obvious how one can extend this to continuous actions and states
 - Everything is mathematically exact, expressible and analysable and best for small problems.



Model-Free Prediction and Control

Monte Carlo Learning



LUND
UNIVERSITY

Example Blackjack

- States (200 of them):
 - Current sum (12-21)
 - Dealer's showing card (ace-10)
 - Do I have a “useable” ace? (yes-no)
- Action **stick**: Stop receiving cards (and terminate)
- Action **twist**: Take another card (no replacement)
- Reward for **stick**:
 - +1 if sum of cards > sum of dealer cards
 - 0 if sum of cards = sum of dealer cards
 - -1 if sum of cards < sum of dealer cards
- Reward for **twist**:
 - -1 if sum of cards > 21 (and terminate)
 - 0 otherwise
- Transitions: automatically twist if sum of cards < 12



Plan

- Transition function $P_{s,s'}^a$? Rewards R_s^a ?
 - Two common cases
 1. The MDP is *unknown* but the experience is available: *Let's try things out and see what happens...*
 2. The MDP is known but too complicated to use.
- **Goal:**
 - **Estimate** the value function for an *unknown* MDP
 - **Optimise** the value function and the policy of an *unknown* MDP



LUND
UNIVERSITY

Learning from Episodes

- The goal: estimate the value function of an *unknown* MDP
- The trick: Use experiences!!

We learn v_π from episodes of experience under the policy π .

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$



Monte Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience / observations
- MC uses simplest possible idea: value = sample mean of the return



LUND
UNIVERSITY

Reminder: Expected Return

- Remember:
 - the **return** $G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
 - The **value function** is the expected return: $v_{\pi}(s) = E_{\pi} \{ G_t | S_t = s \}$
- The goal is to estimate the expected return from the observed episodes

$$E^1 : S_1^1, A_1^1, R_2^1, \dots, S_k^1$$

$$E^2 : S_1^2, A_1^2, R_2^2, \dots, S_k^2$$

...

$$E^n : S_1^n, A_1^n, R_2^n, \dots, S_k^n$$

Monte Carlo Policy Evaluation

Counting and Averaging



LUND
UNIVERSITY

Monte Carlo Policy Evaluation

Counting and Averaging

- For every episode E^i



LUND
UNIVERSITY

Monte Carlo Policy Evaluation

Counting and Averaging

- For every episode E^i
 - **First** times step t where state s is visited



LUND
UNIVERSITY

Monte Carlo Policy Evaluation

Counting and Averaging

- For every episode E^i
 - **First** times step t where state s is visited
 - increment counter $N(s) \leftarrow N(s) + 1$



Monte Carlo Policy Evaluation

Counting and Averaging

- For every episode E^i
 - **First** times step t where state s is visited
 - increment counter $N(s) \leftarrow N(s) + 1$
 - Increment total return $S(s) \leftarrow S(s) + G_t$



Monte Carlo Policy Evaluation

Counting and Averaging

- For every episode E^i
 - **First** times step t where state s is visited
 - increment counter $N(s) \leftarrow N(s) + 1$
 - Increment total return $S(s) \leftarrow S(s) + G_t$
 - The estimated mean return: $V(s) \leftarrow S(s)/N(s)$



Monte Carlo Policy Evaluation

Counting and Averaging

- For every episode E^i
 - **First** times step t where state s is visited
 - increment counter $N(s) \leftarrow N(s) + 1$
 - Increment total return $S(s) \leftarrow S(s) + G_t$
 - The estimated mean return: $V(s) \leftarrow S(s)/N(s)$
- Law of large numbers: $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$



Monte Carlo Policy Evaluation

Counting and Averaging

- For every episode E^i
 - **Every** times step t that state s is visited in episode E^i
 - increment counter $N(s) \leftarrow N(s) + 1$
 - Increment total return $S(s) \leftarrow S(s) + G_t$
 - The estimated mean return: $V(s) \leftarrow S(s)/N(s)$
- Law of large numbers: $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$



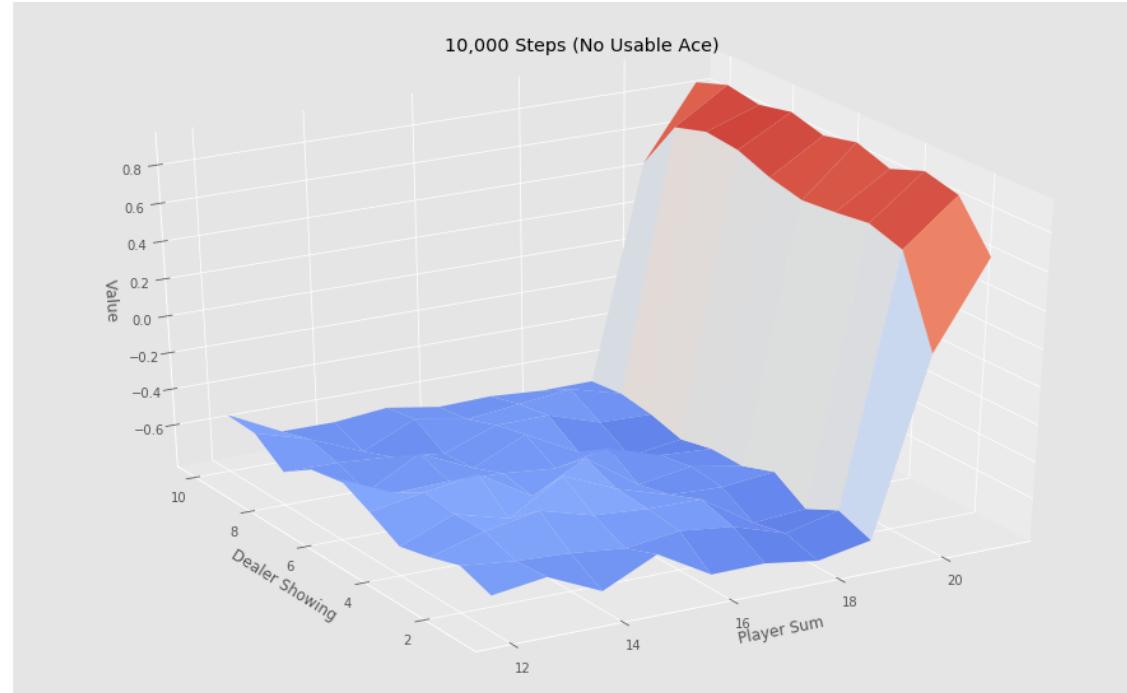
Example Blackjack

- States (200 of them):
 - Current sum (12-21)
 - Dealer's showing card (ace-10)
 - Do I have a “useable” ace? (yes-no)
- Action **stick**: Stop receiving cards (and terminate)
- Action **twist**: Take another card (no replacement)
- Reward for **stick**:
 - +1 if sum of cards > sum of dealer cards
 - 0 if sum of cards = sum of dealer cards
 - -1 if sum of cards < sum of dealer cards
- Reward for **twist**:
 - -1 if sum of cards > 21 (and terminate)
 - 0 otherwise
- Transitions: automatically twist if sum of cards < 12
- Policy: **stick** if sum of cards ≥ 20 , otherwise **twist**

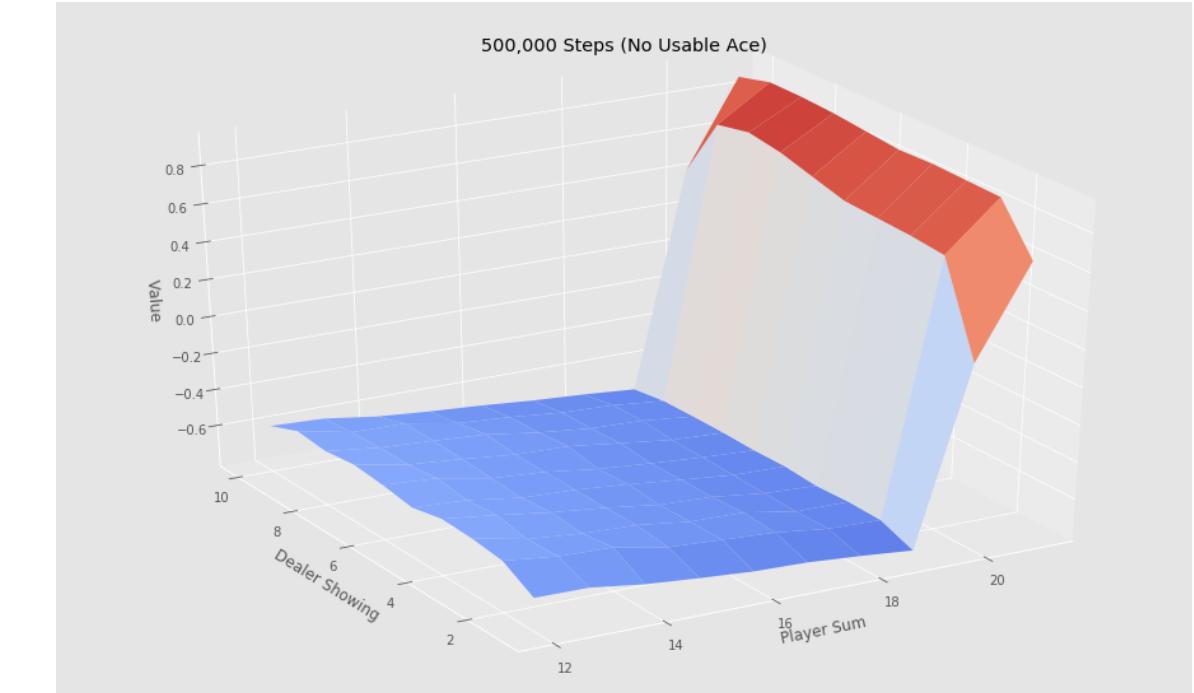


Blackjack Value FUNCTION WITH MONTE CARLO

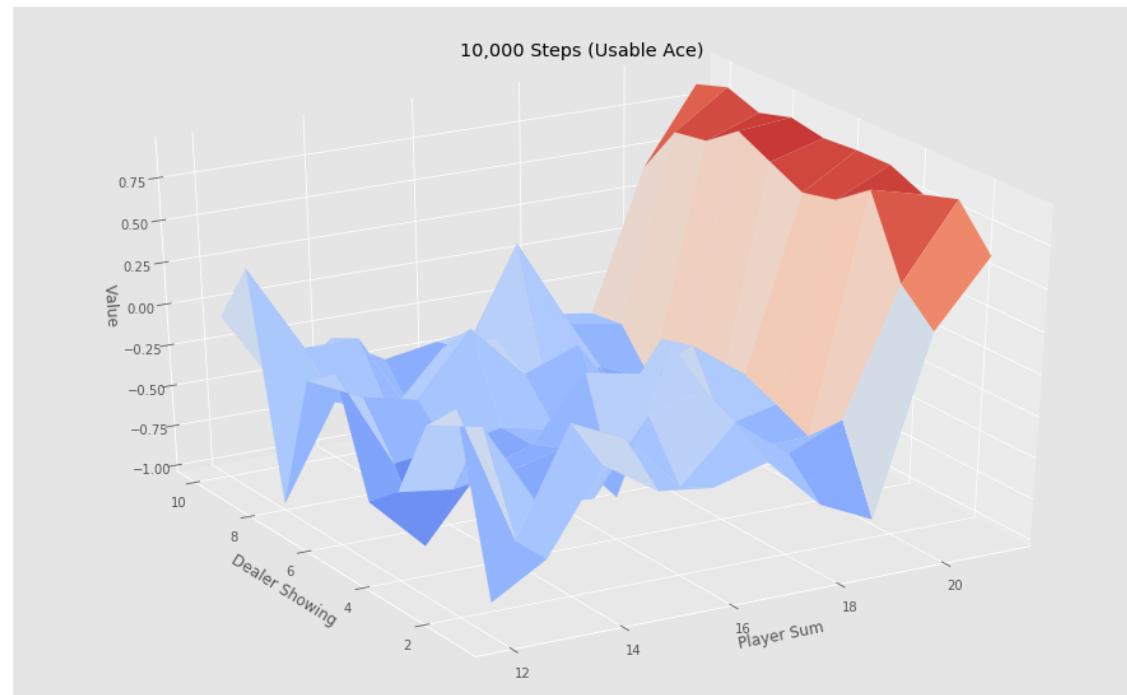
50.000 episodes



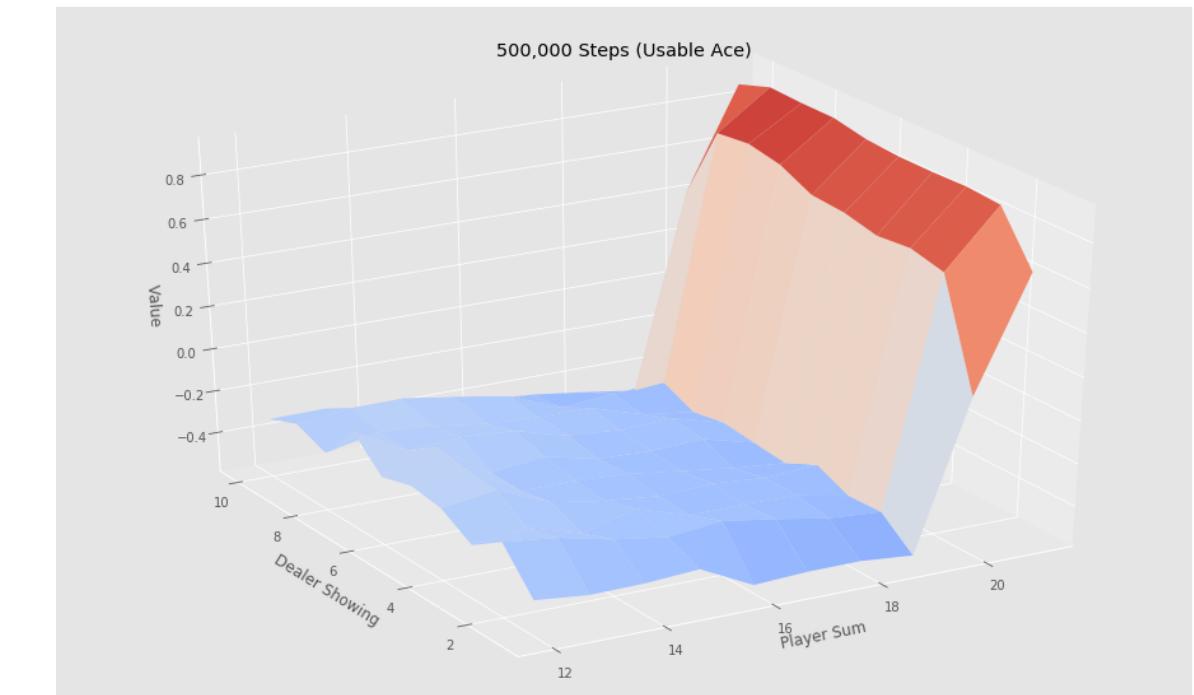
500.000 episodes



50.000 episodes



500.000 episodes



Policy: **stick** if sum of cards ≥ 20 , otherwise **twist**

Recursive computation of mean

- The mean μ_k can be computer recursively from the sequence x_1, x_2, \dots

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} (x_k + \sum_{j=1}^{k-1} x_j) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} \underbrace{(x_k - \mu_{k-1})}_{\text{error-term}}\end{aligned}$$



Incremental Monte-Carlo Updates

- Update each $V(s)$ after each episode E^i :
 - For every state S_t with return G_t
 - increment counter $N(s) \leftarrow N(s) + 1$
 - Increment total return $S(s) \leftarrow S(s) + G_t$
 - The estimated mean return: $V(s) \leftarrow S(s)/N(s)$



Incremental Monte-Carlo Updates

- Update each $V(s)$ after each episode E^i :
 - For every state S_t with return G_t
 - increment counter $N(s) \leftarrow N(s) + 1$
 - ~~Increment total return $S(s) \leftarrow S(s) + G_t$~~
 - The estimated mean return: ~~$V(s) \leftarrow S(s)/N(s)$~~



Incremental Monte-Carlo Updates

- Update each $V(s)$ after each episode E^i :
 - For every state S_t with return G_t
 - increment counter $N(s) \leftarrow N(s) + 1$
 - ~~Increment total return $S(s) \leftarrow S(s) + G_t$~~
 - The estimated mean return: ~~$V(s) \leftarrow S(s)/N(s)$~~

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$



Incremental Monte-Carlo Updates

- Update each $V(s)$ after each episode E^i :
 - For every state S_t with return G_t
 - increment counter $N(s) \leftarrow N(s) + 1$
 - ~~Increment total return $S(s) \leftarrow S(s) + G_t$~~
 - The estimated mean return: ~~$V(s) \leftarrow S(s)/N(s)$~~

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



Optimizing the Policy

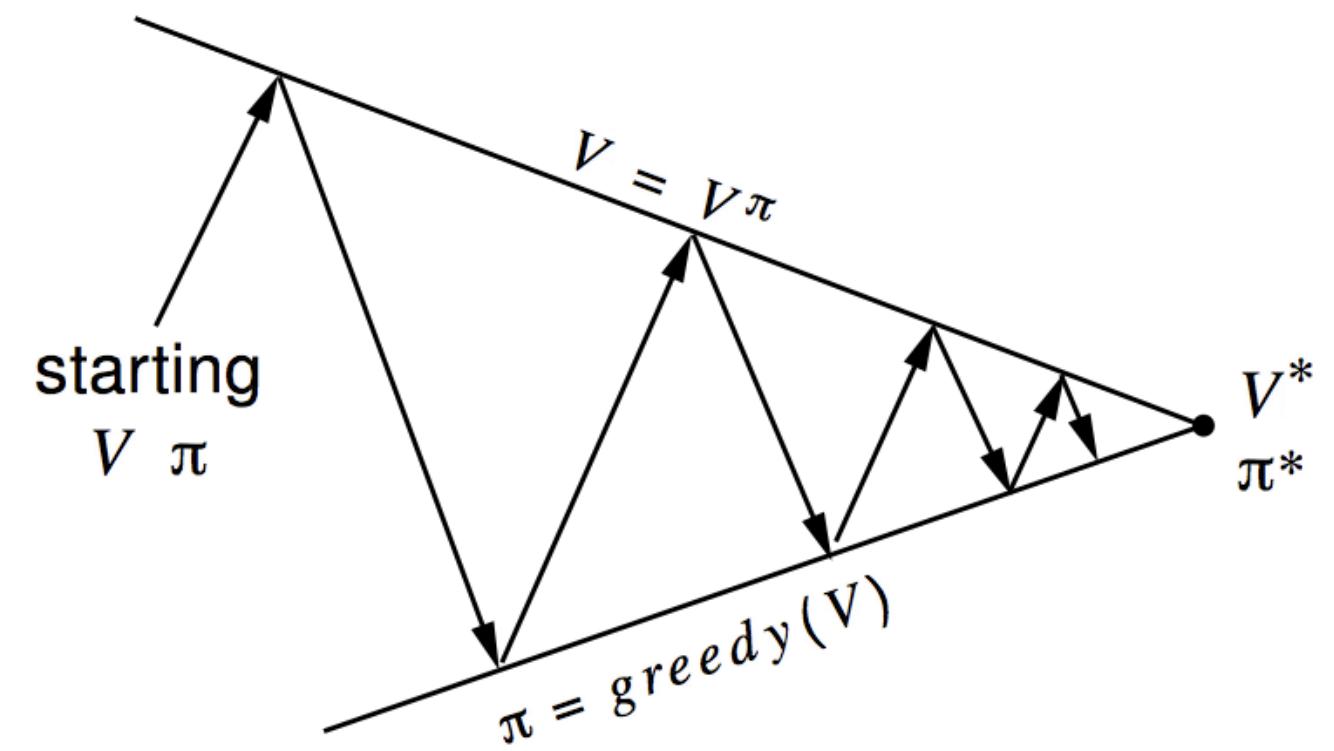
Monte Carlo Control



LUND
UNIVERSITY

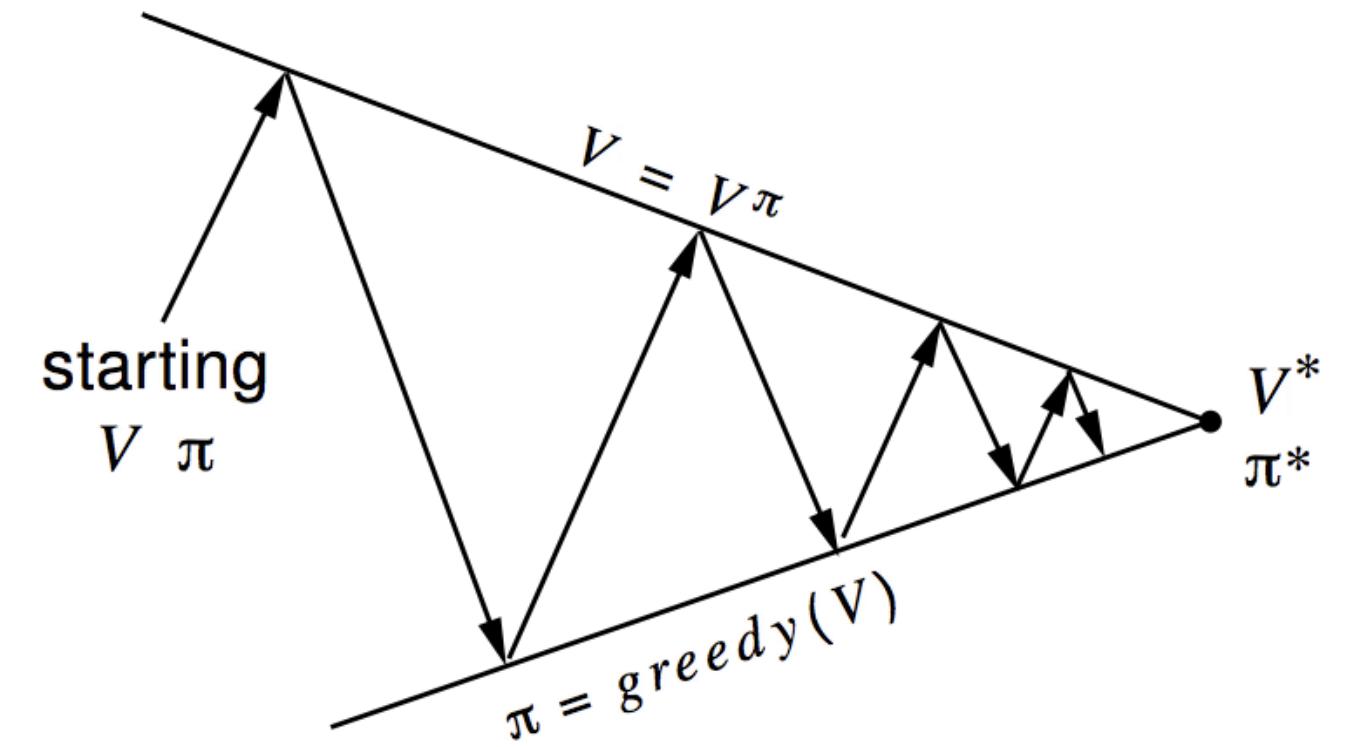
Policy Iteration

- **Policy Evaluation:** Estimate v_π
 - Iterative Policy Evaluation
- **Policy improvement:** Generate $\pi' \geq \pi$
 - Greedy policy improvement



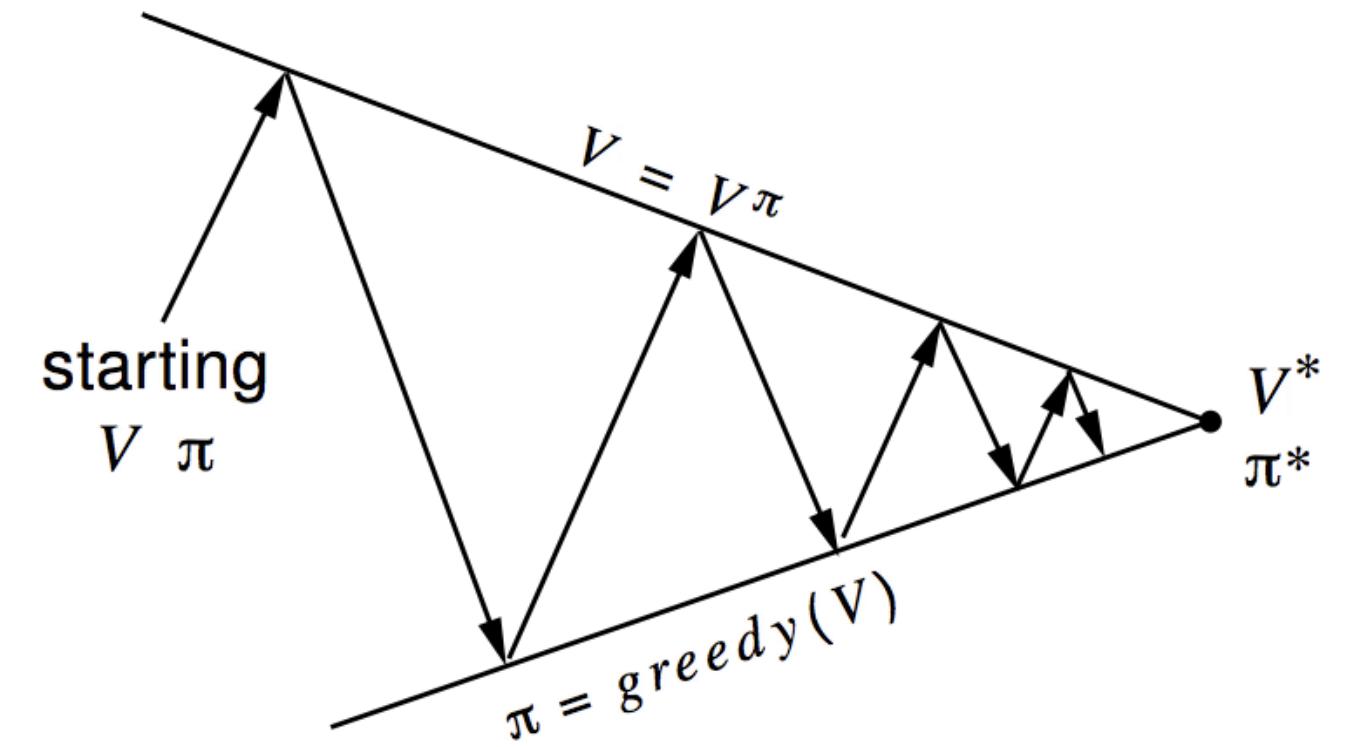
Policy Iteration

- **Policy Evaluation:** Estimate v_π
 - Iterative Policy Evaluation
 - Monte Carlo Policy Evaluation
- **Policy improvement:** Generate $\pi' \geq \pi$
 - Greedy policy improvement



Policy Iteration

- **Policy Evaluation:** Estimate v_π
 - Iterative Policy Evaluation
 - Monte Carlo Policy Evaluation
- **Policy improvement:** Generate $\pi' \geq \pi$
 - Greedy policy improvement **???**



Computing the Policy

- Greedy policy improvement over $V(s)$ requires knowledge of the MDP:

$$\pi'(s) = \arg \max_{a \in A} (R_s^a + P_{ss'}^a v_\pi(s'))$$



Computing the Policy

- Greedy policy improvement over $V(s)$ requires knowledge of the MDP:

$$\pi'(s) = \arg \max_{a \in A} (R_s^a + P_{ss'}^a v_\pi(s'))$$



Computing the Policy

- Greedy policy improvement over $V(s)$ requires knowledge of the MDP:

$$\pi'(s) = \arg \max_{a \in A} \left(R_s^a + P_{ss'}^a v_\pi(s') \right)$$

- Greedy improvement over $Q(s, a)$ is model-free.

$$\pi'(s) = \arg \max_{a \in A} Q(s, a)$$



Computing the Policy

- Greedy policy improvement over $V(s)$ requires knowledge of the MDP:

$$\pi'(s) = \arg \max_{a \in A} (R_s^a + P_{ss'}^a v_\pi(s'))$$

- Greedy improvement over $Q(s, a)$ is model-free.

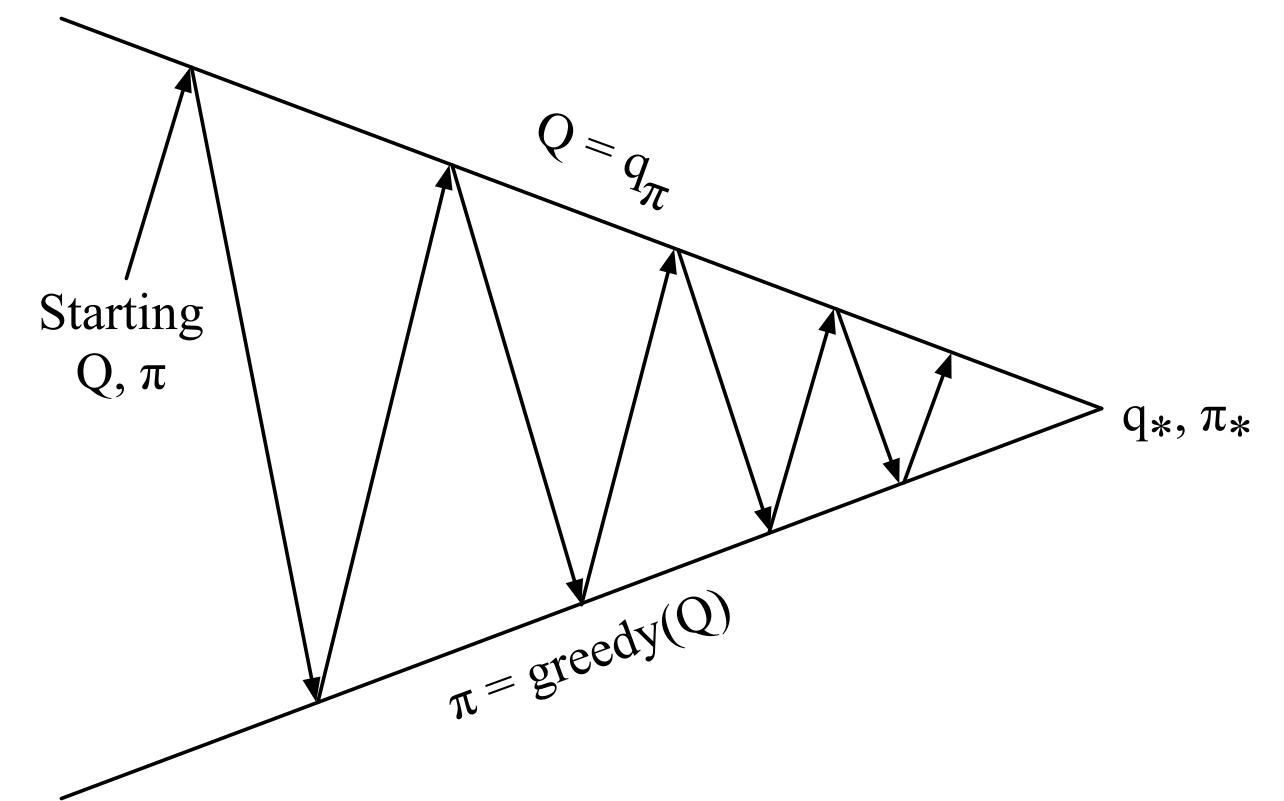
$$\pi'(s) = \arg \max_{a \in A} Q(s, a)$$

- What we need is a statistic of state transitions s given an action a

$S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_k$

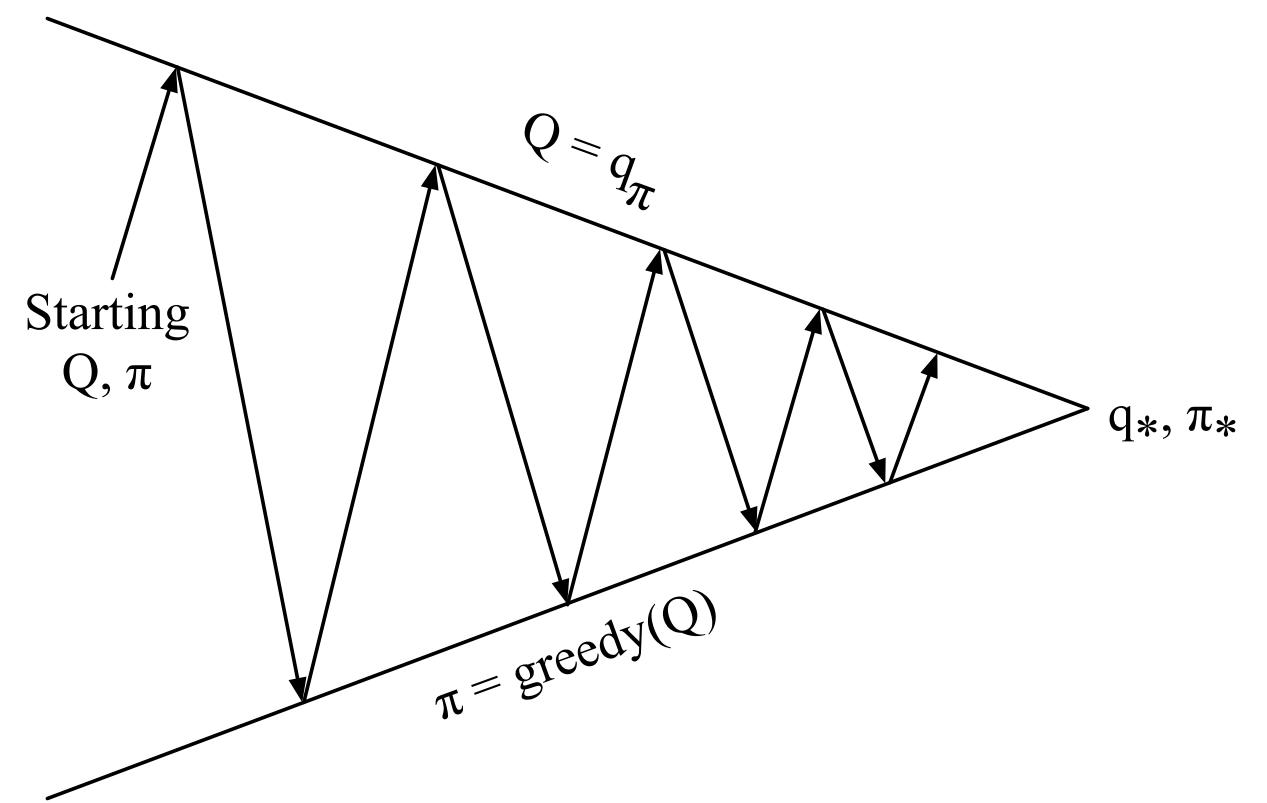
Generalized Policy Iteration for action value function

- Policy Evaluation:
 - Monte Carlos Policy Evaluation of $Q = q_\pi$
- Policy Improvement:
 - Greedy policy improvement **???**



Generalized Policy Iteration for action value function

- Policy Evaluation:
 - Monte Carlos Policy Evaluation of $Q = q_\pi$
- Policy Improvement:
 - Greedy policy improvement **???**



We learn on the job: The policy we can find when using greedy improvement will depend on the policy we have now.
We might miss certain state-action pairs.

Example of a greedy action



- Two restaurants you can choose from
 - The top one was closed and we got no food.
 - $V(\text{top}) = 0$
 - You go to the bottom one and get a reward +1
 - $V(\text{bottom}) = +1$
 - You go to the bottom one and get a reward +2
 - $V(\text{bottom}) = +2$
 - You go to the bottom one and get a reward +1
 - $V(\text{bottom}) = +1$
 - ...
 - Are you sure you chose the right restaurant?



ϵ -greedy exploration

- Try out something new for a change!! **exploration vs exploitation**
- Simplest idea to keep exploring: **ϵ -greedy**
 - All m actions are tried out with above zero probability:
 - With probability $1 - \epsilon$ you try the greedy action
 - with probability ϵ you choose an action at random

$$\pi'(a | s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \arg \max_{a \in A} Q(s, a) \geq 0 \\ \epsilon/m & \text{otherwise} \end{cases}$$



ϵ -greedy Policy Improvement

Theorem: For any ϵ -greedy policy π , the ϵ -greedy policy π' with respect to q_π improves the value function: $v_{\pi'}(s) \geq v_\pi(s)$

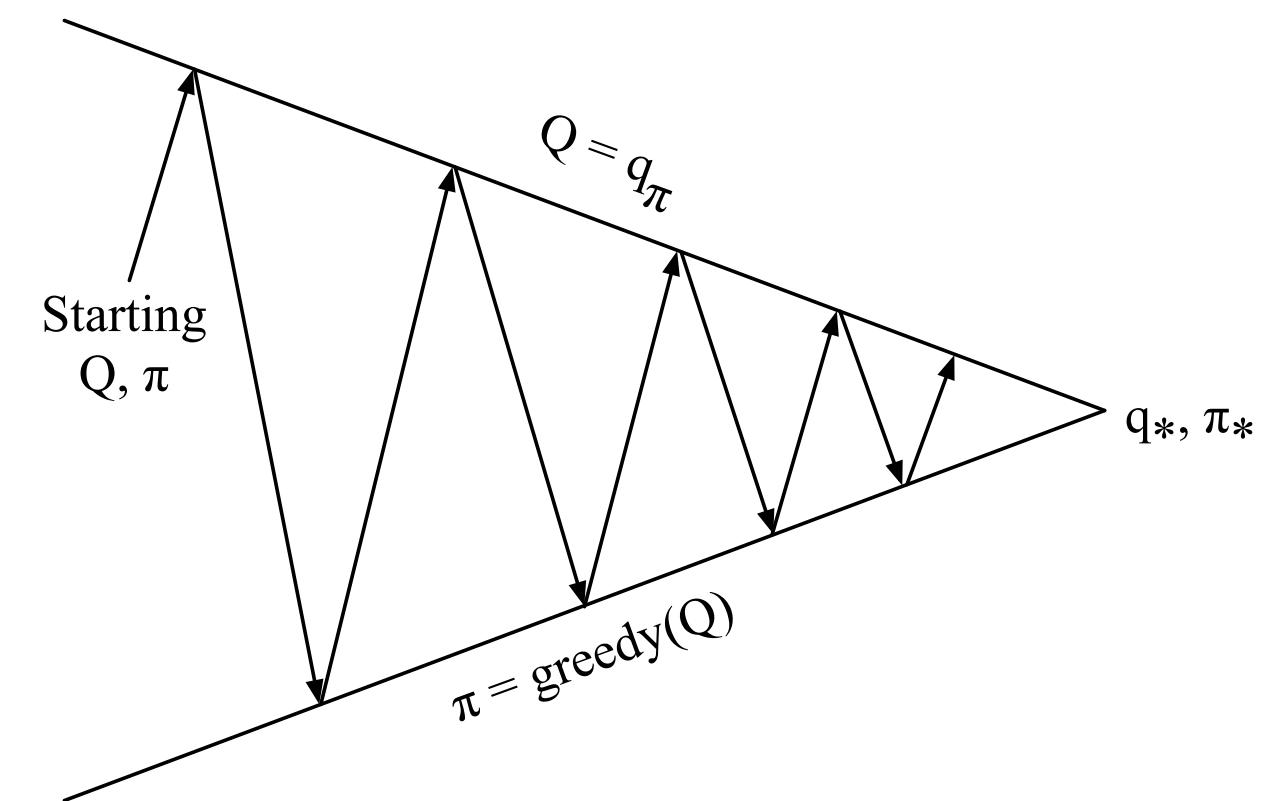
blackboard



LUND
UNIVERSITY

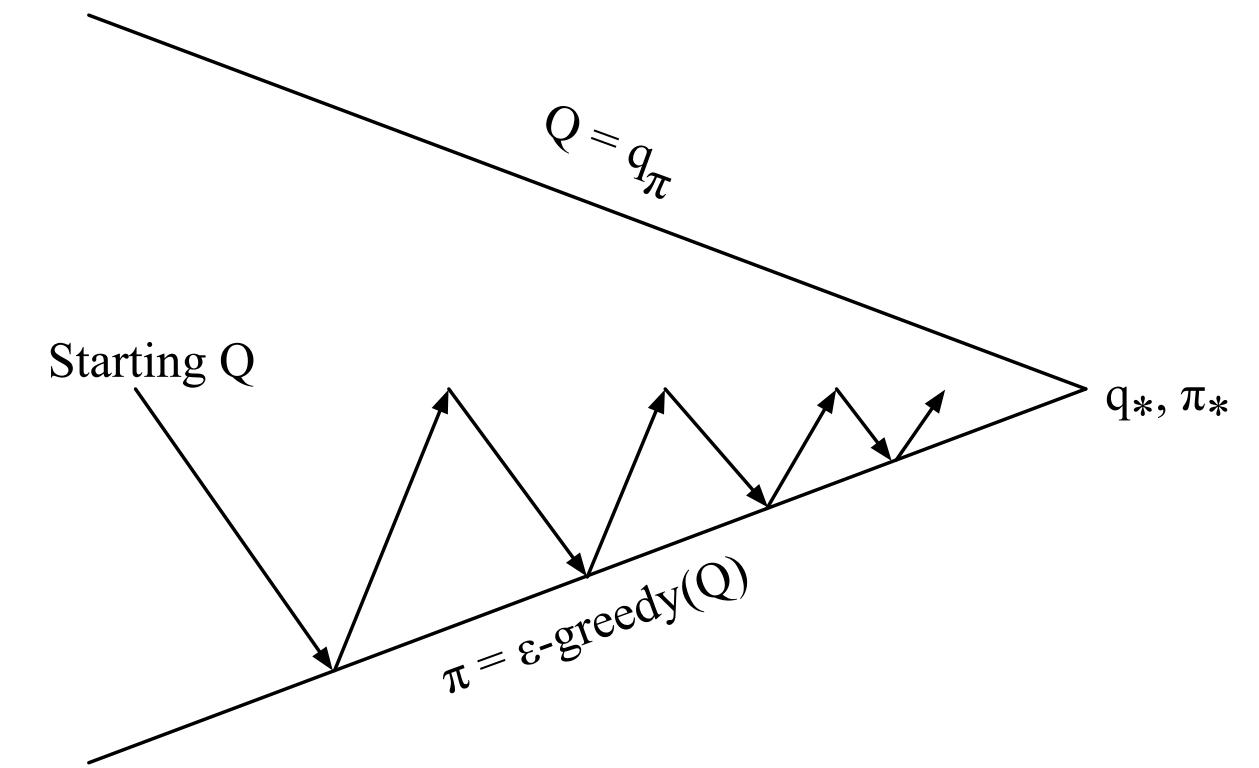
Monte Carlo Control

- Policy Evaluation:
 - Monte Carlo Policy Evaluation of $Q = q_\pi$
- Policy Improvement:
 - ϵ -greedy policy improvement



Monte Carlo Control more efficient

- For every episode:
 - Policy Evaluation:
 - Monte Carlos Policy Evaluation of $Q \approx q_\pi$
 - Policy Improvement:
 - ϵ -Greedy policy improvement



GLIE Monte Carlo Control

- **Greedy in the Limit with Infinite Exploration** (GLIE)
- Two principles:
 1. keep exploring
 2. make sure that the policy eventually becomes greedy!
 - needed for the Bellman optimality equation.
- One possibility: ϵ -greedy policy with slow decay of ϵ : e.g., $\epsilon_k = \frac{1}{k}$



GLIE Monte Carlo Control

- Generate a new episode E^k using your policy $\pi : S_1, A_1, R_2, S_2, A_2, R_3, \dots, R_T$
 - For each state S_t and action A_t in the episode

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve the policy based on the new $Q(S, A)$ $\epsilon \leftarrow 1/k$

GLIE Monte Carlos converges to the optimal $q^*(s, a)$



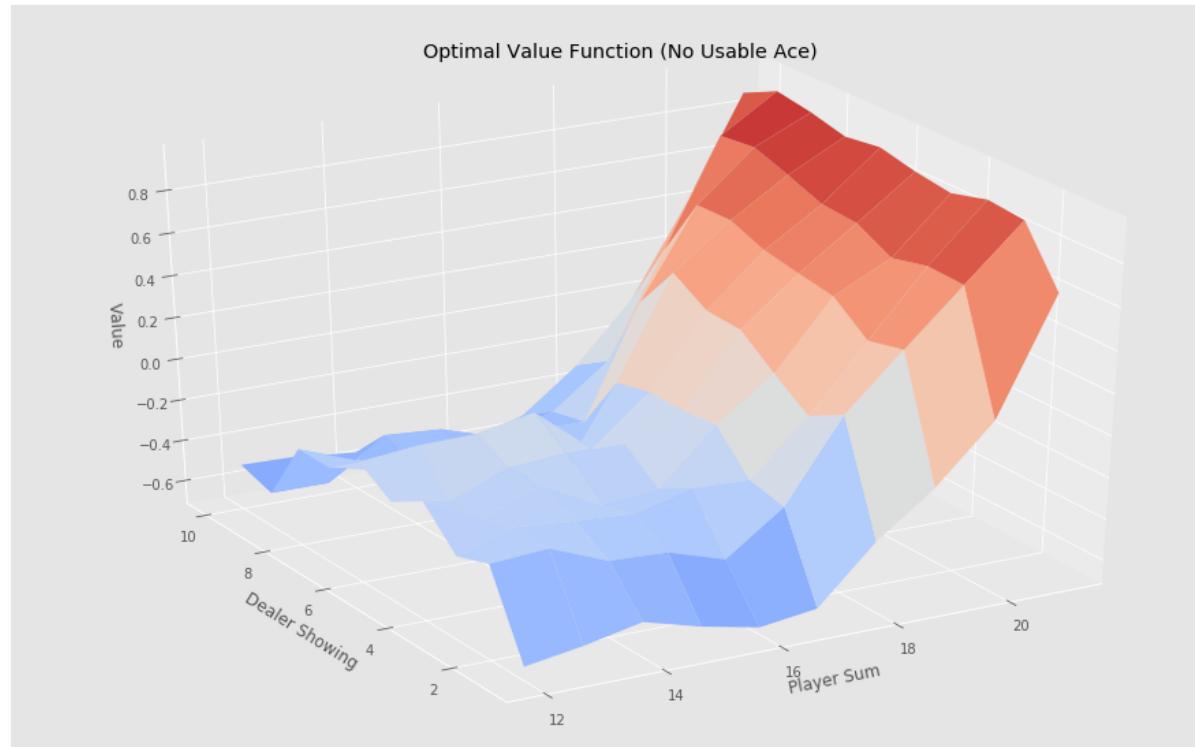
Back to Black Jack



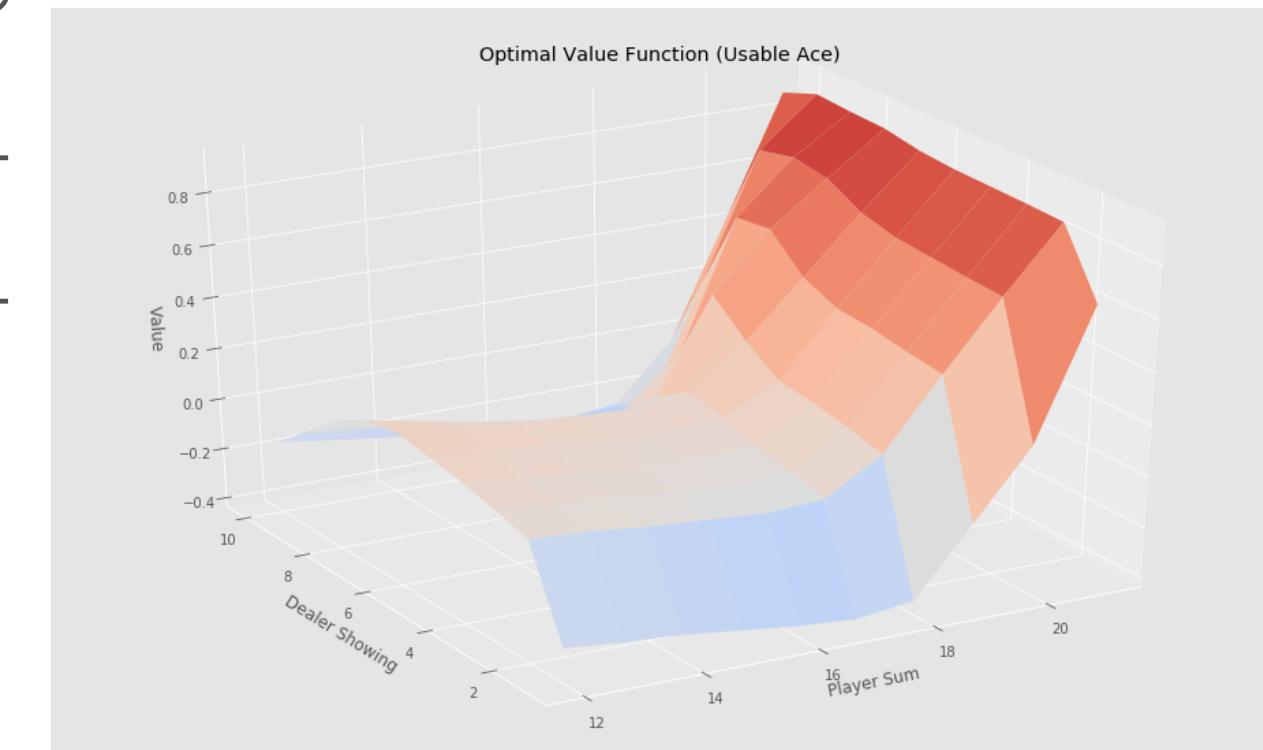
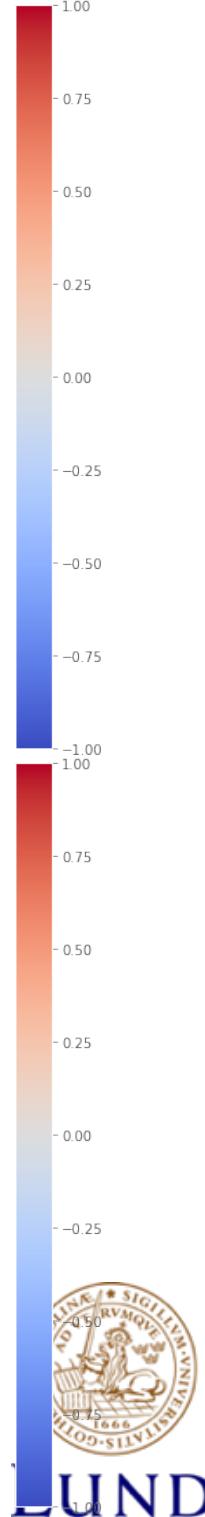
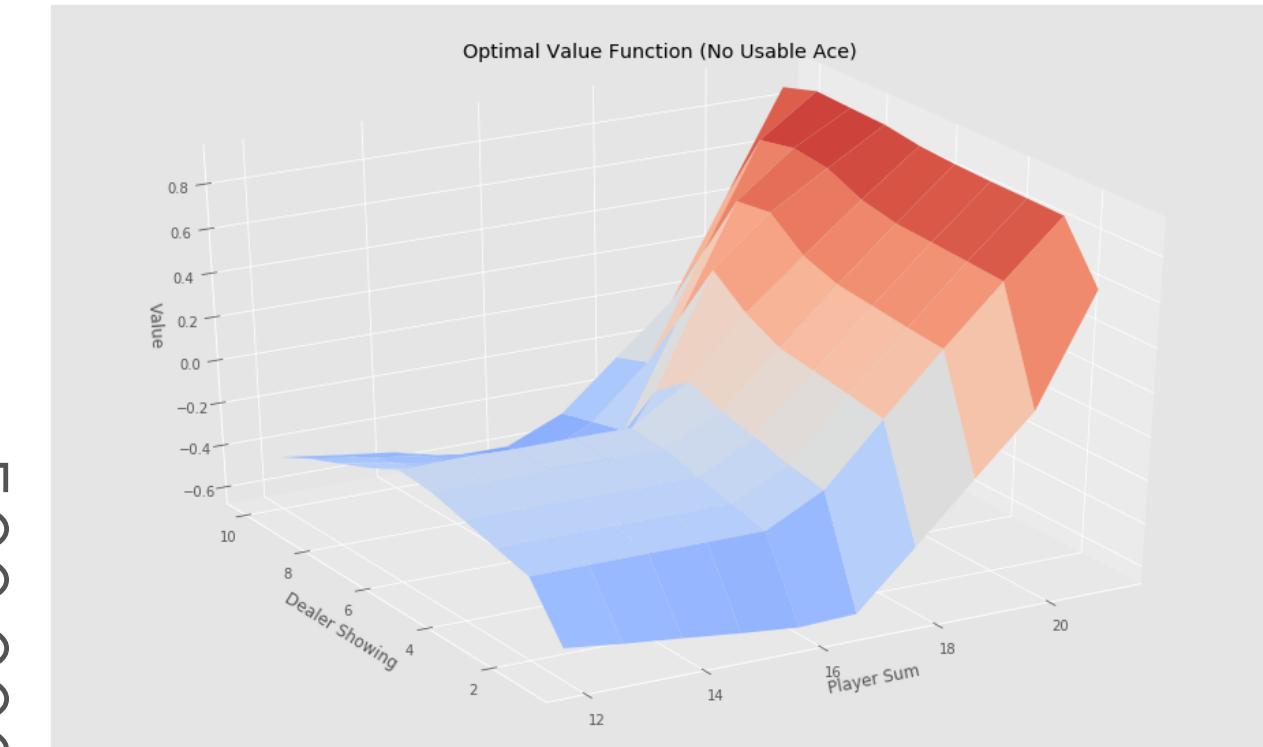
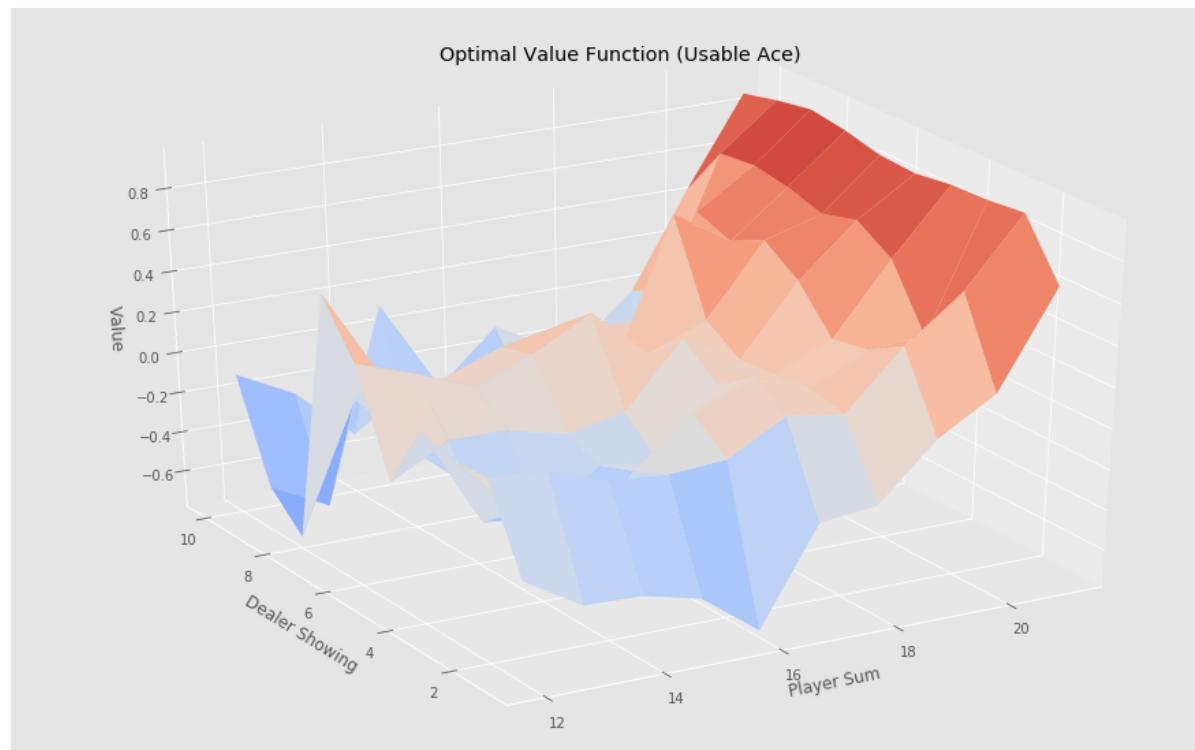
LUND
UNIVERSITY

Monte Carlo Control in Black Jack

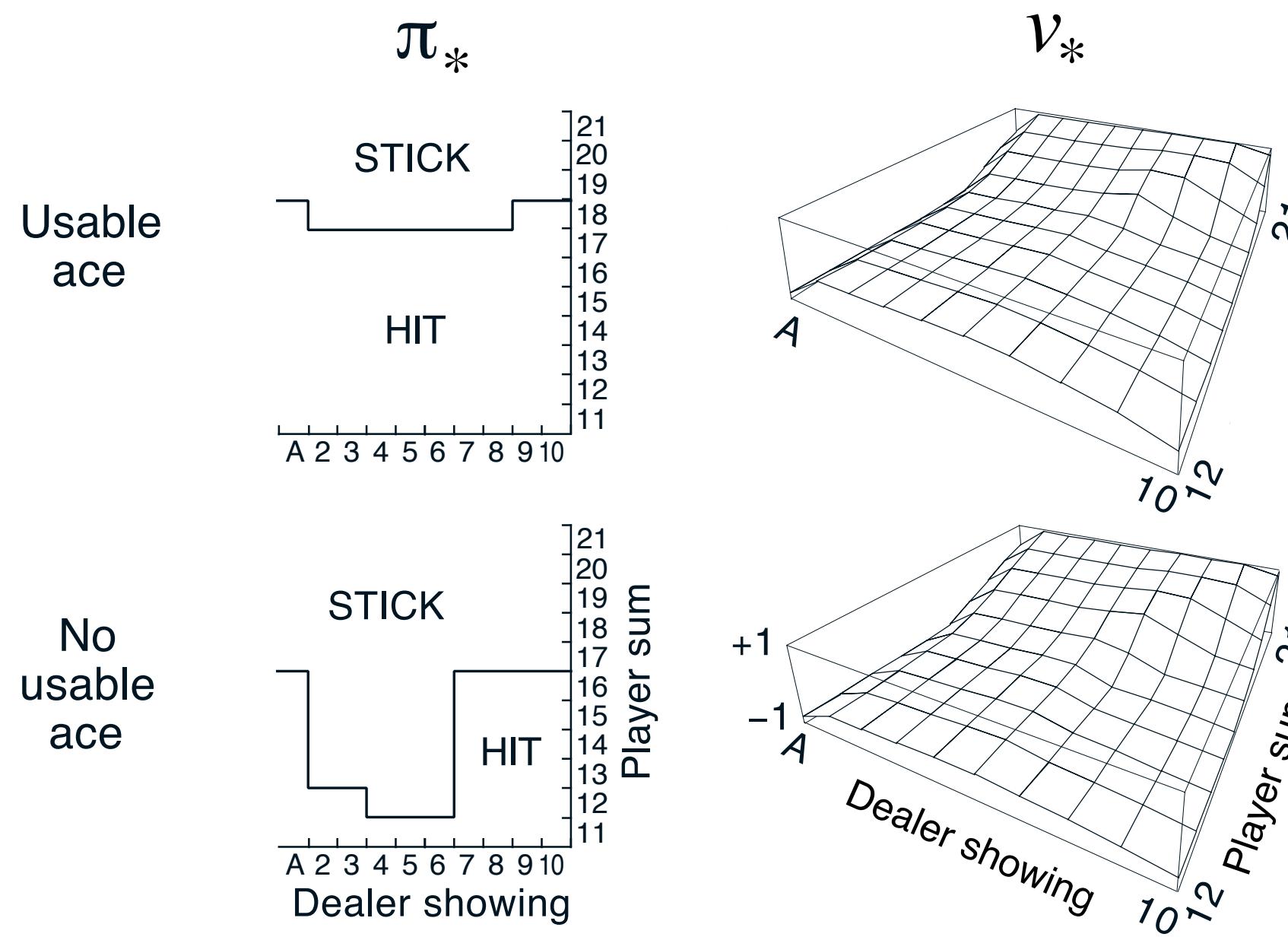
50.000 episodes



500.000 episodes

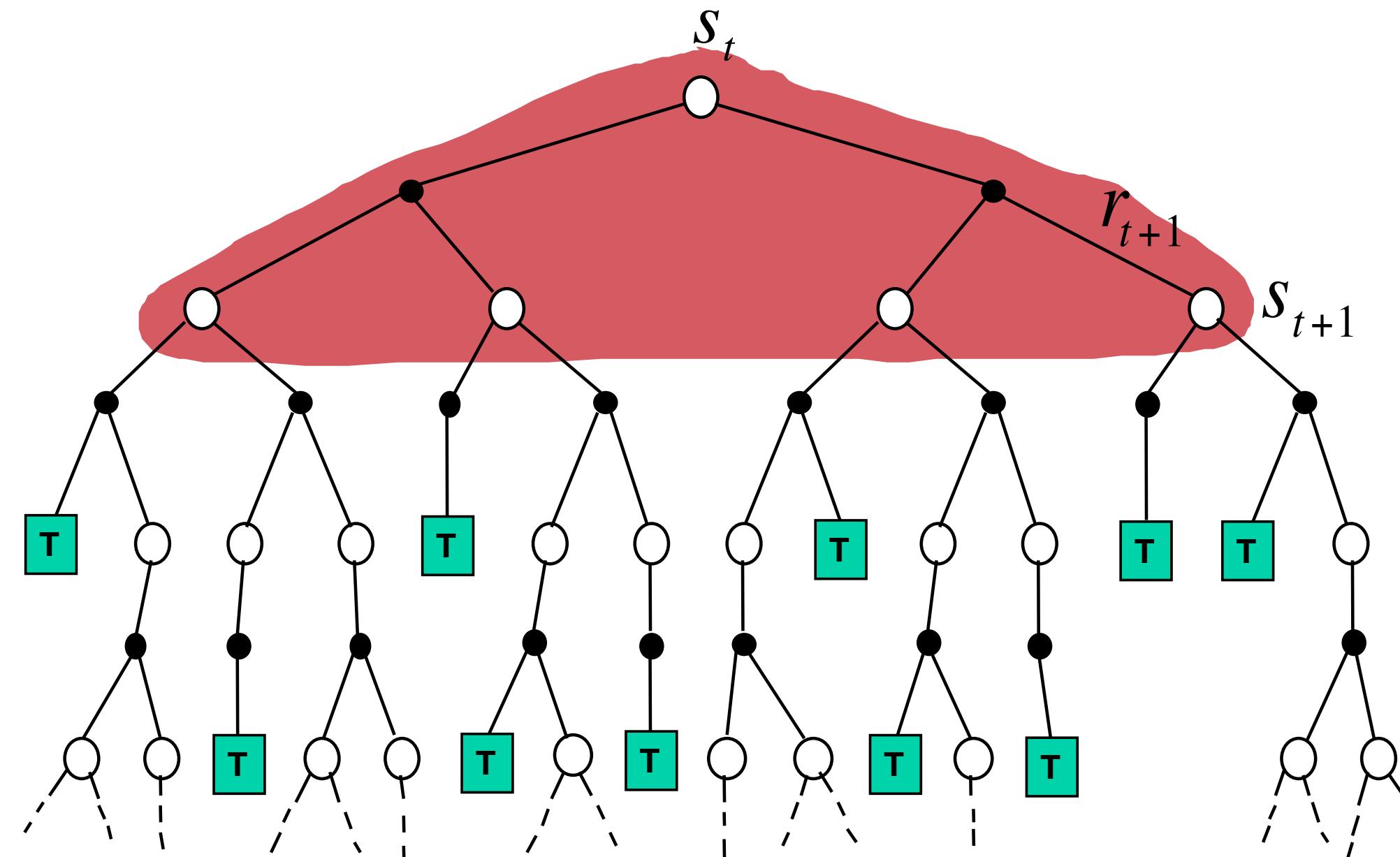


Monte Carlo Control in Black Jack



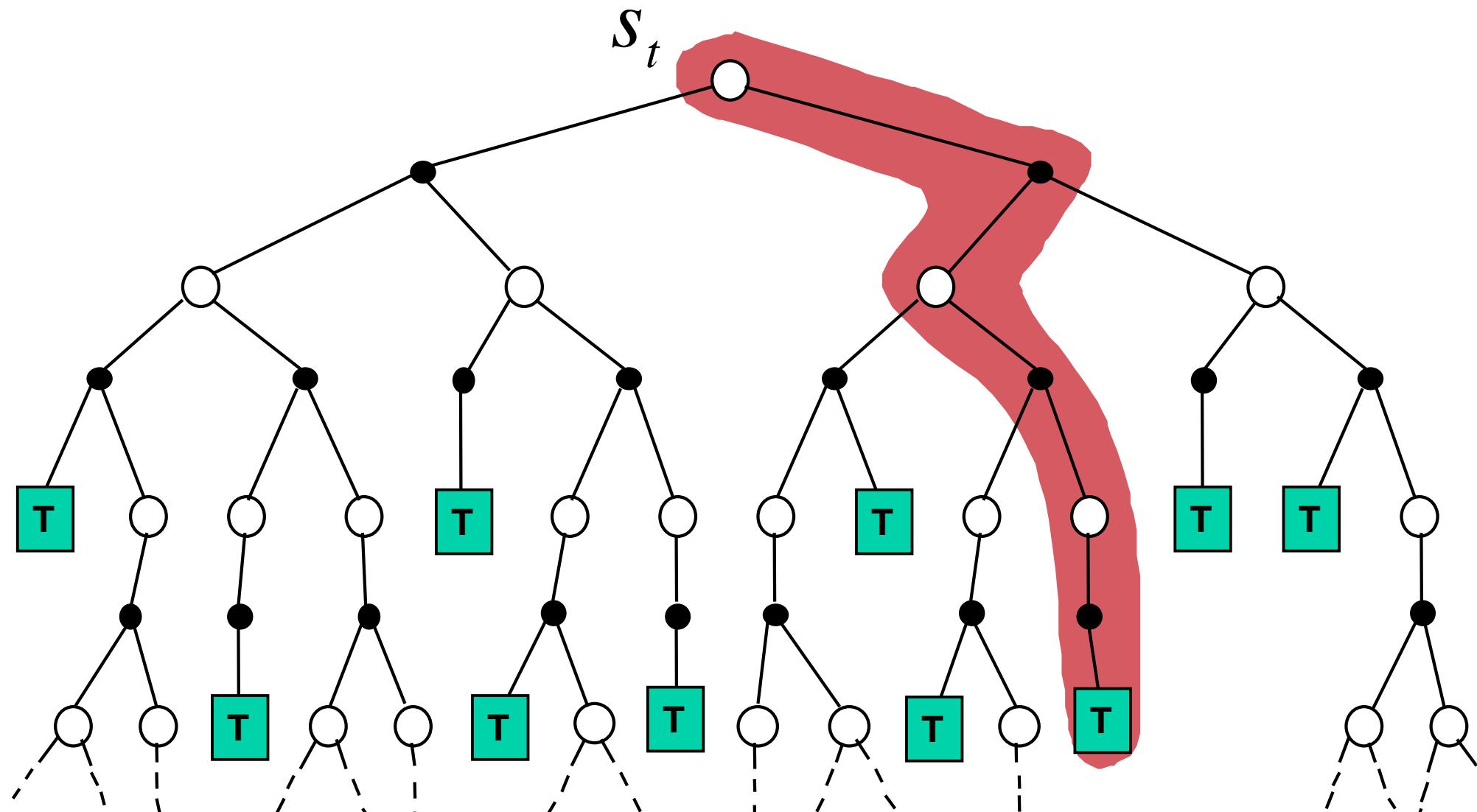
Dynamic programming backup

$$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$$



Monte Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



Summary Monte Carlo Learning

- First real RL approach for finding an optimised policy
- MC is *model-free*: no knowledge of the MDP transitions / rewards are needed
- Can apply MC only to *episodic* MDPs: all episodes must end.
- based on observations only
- policy optimisation done via the state action value Q
- ϵ -greedy approach for finding the best policy while exploring at the same time.
- GLIE Monte Carlo Control allows to slowly reduce ϵ to 0 while assuring convergence (for proof, see Sect 5 of the book).



Windy Gridworld

