PLANNING AND PREDICTION

Volker Krüger

REVISITING
MARKOV DECISION PROCESSES

MARKOV DECISION PROCESS

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

Definition: A **Markov Decision Process** is a tupel $(\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma)$

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

LUND UNIVERSITY

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

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

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

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

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

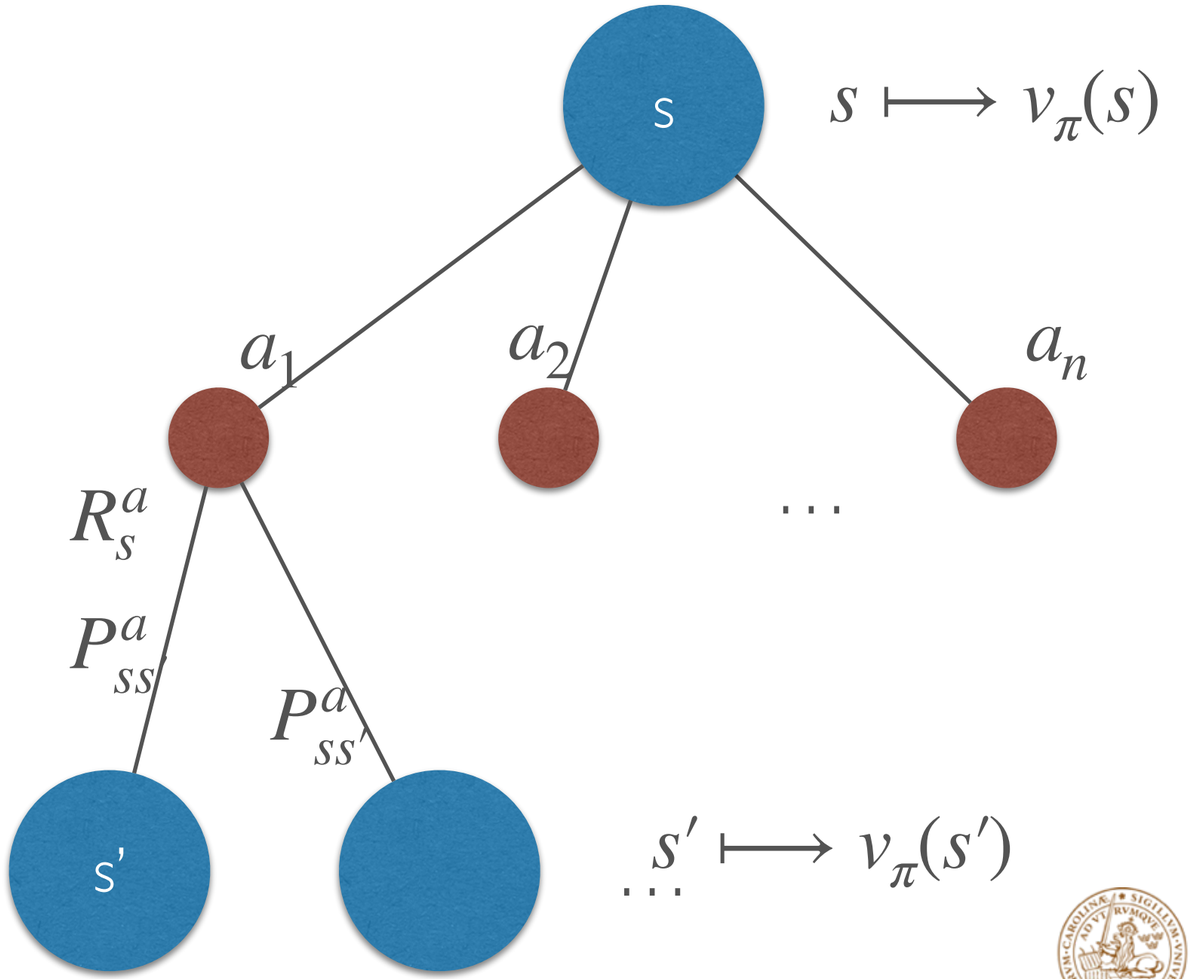
$$q_\pi(s, a) = E_\pi\left\{G_t \,|\, S_t = s, A_t = a\right\} = E_\pi\left\{R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a\right\}$$

LUND UNIVERSITY

BELLMAN EXPECTATION EQUATION

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

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

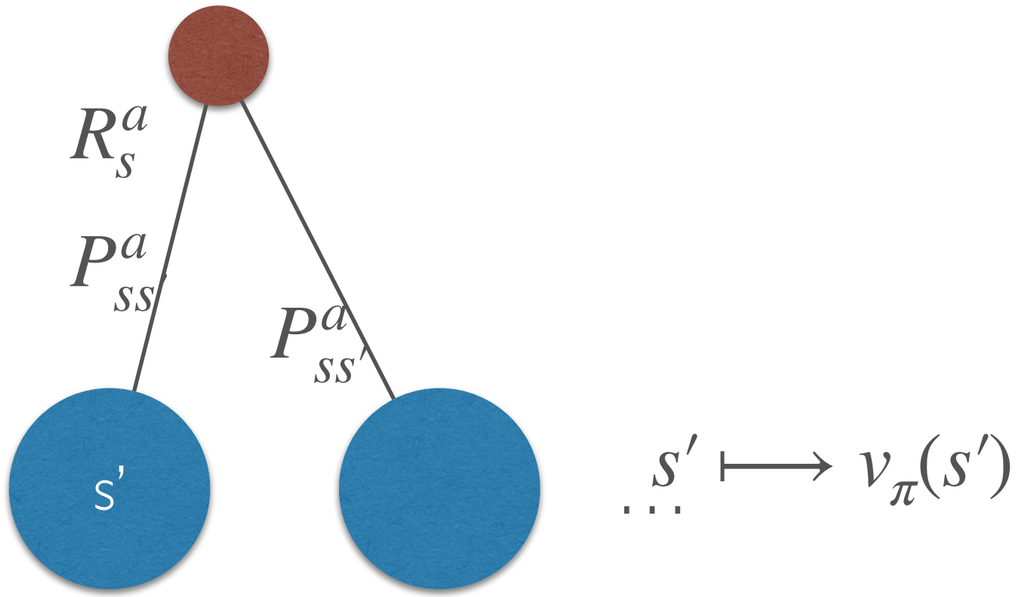


BELLMAN EXPECTATION EQUATION

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

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

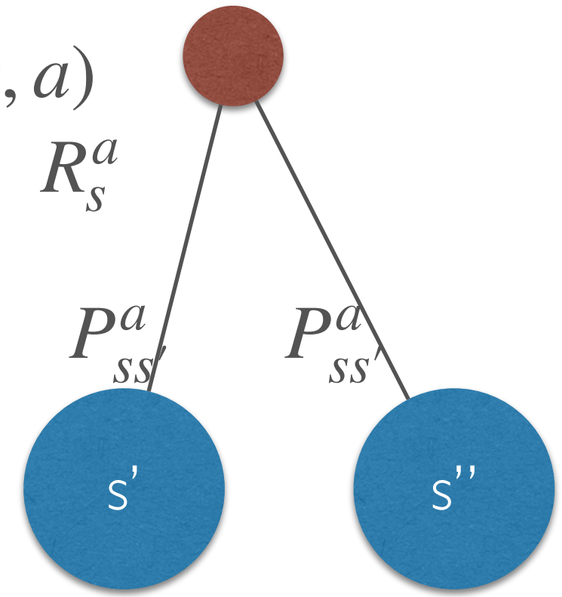
$$v_\pi(s) = \sum_{a \in \mathbf{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_\pi(s') \right)$$



LUND UNIVERSITY

BELLMAN EXPECTATION EQUATION

$$s, a \longmapsto q_\pi(s, a)$$



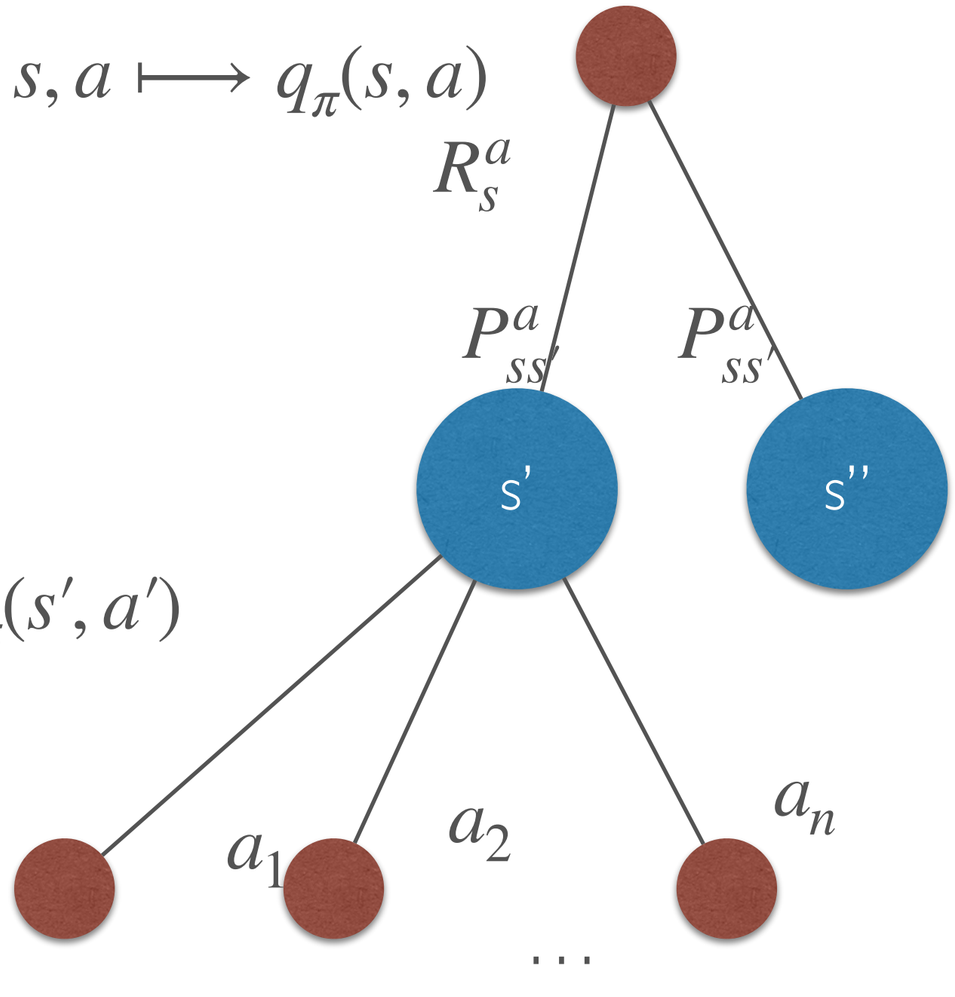
$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a \sum_{a' \in \mathbf{A}} \pi(a' \mid s') q_\pi(s', a')$$

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

LUND UNIVERSITY

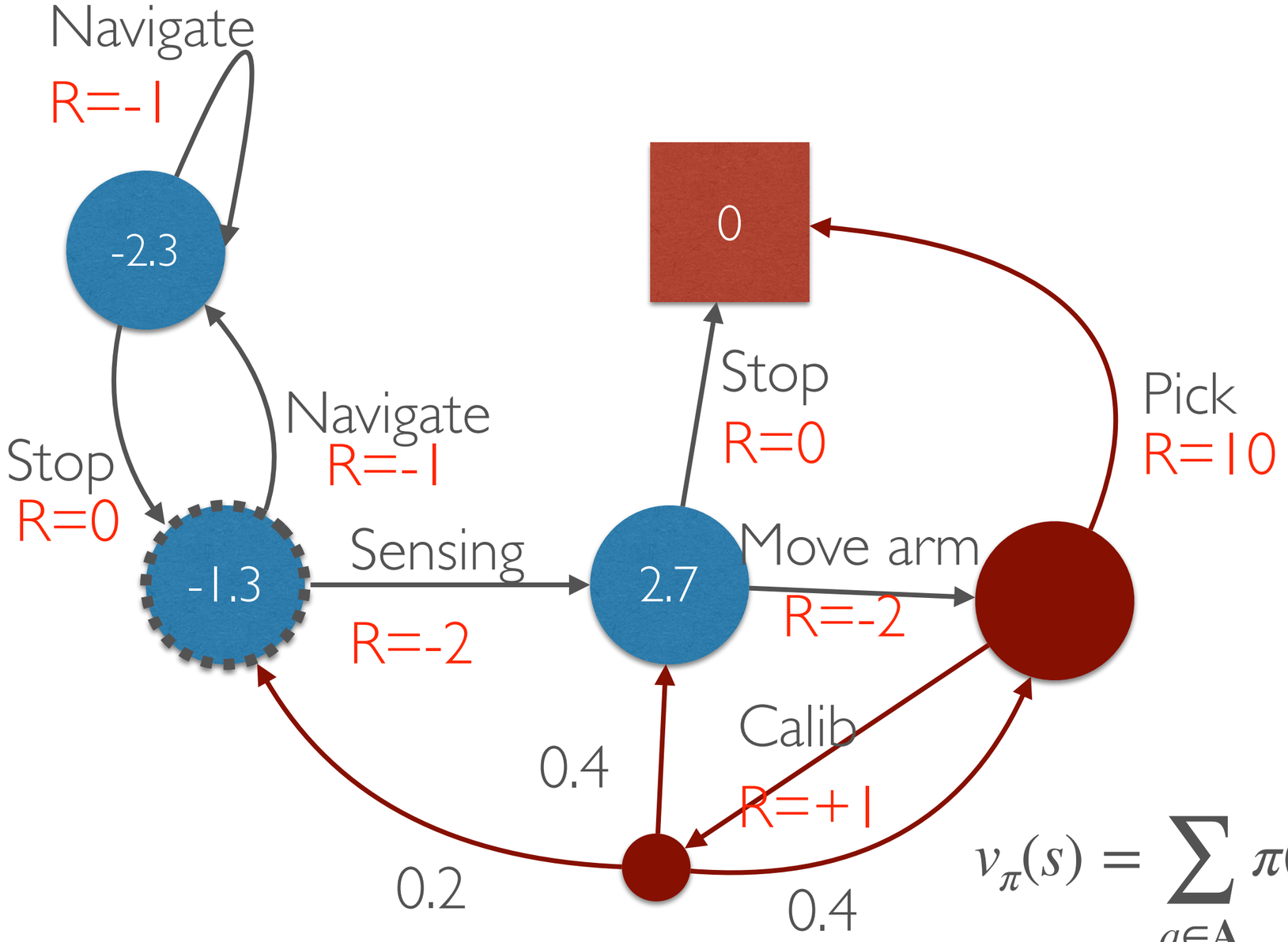
BELLMAN EXPECTATION EQUATION

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



BELLMAN EXPECTATION EQUATION

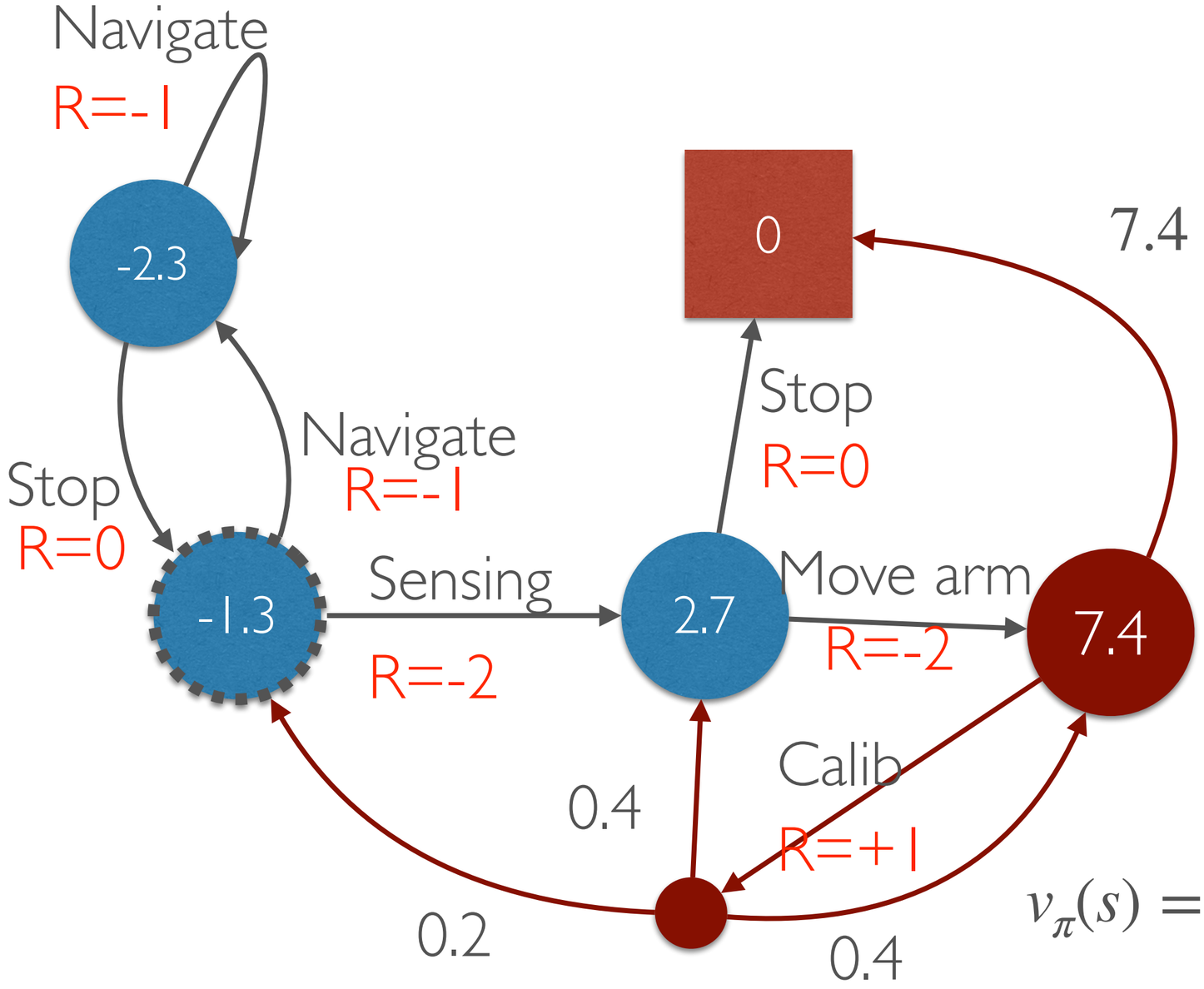
$v_\pi(s)$ for $\pi(s,a) = 0.5,\ \gamma = 1$



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

LUND UNIVERSITY

BELLMAN EXPECTATION EQUATION



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

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

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

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?
    - What should the autonomous system behave in a specific situation?
  - Start with a first simple solution: **Iterative Policy Evaluation** and **Policy Iteration**
  - Then: first steps in **Model-free Prediction and Control**

LUND UNIVERSITY

ITERATIVE POLICY EVALUATION

ASSUMPTIONS

- ***Planning*** in an MDP: *How do we achieve a goal within the MDP?*
- 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.

ITERATIVE POLICY EVALUATION

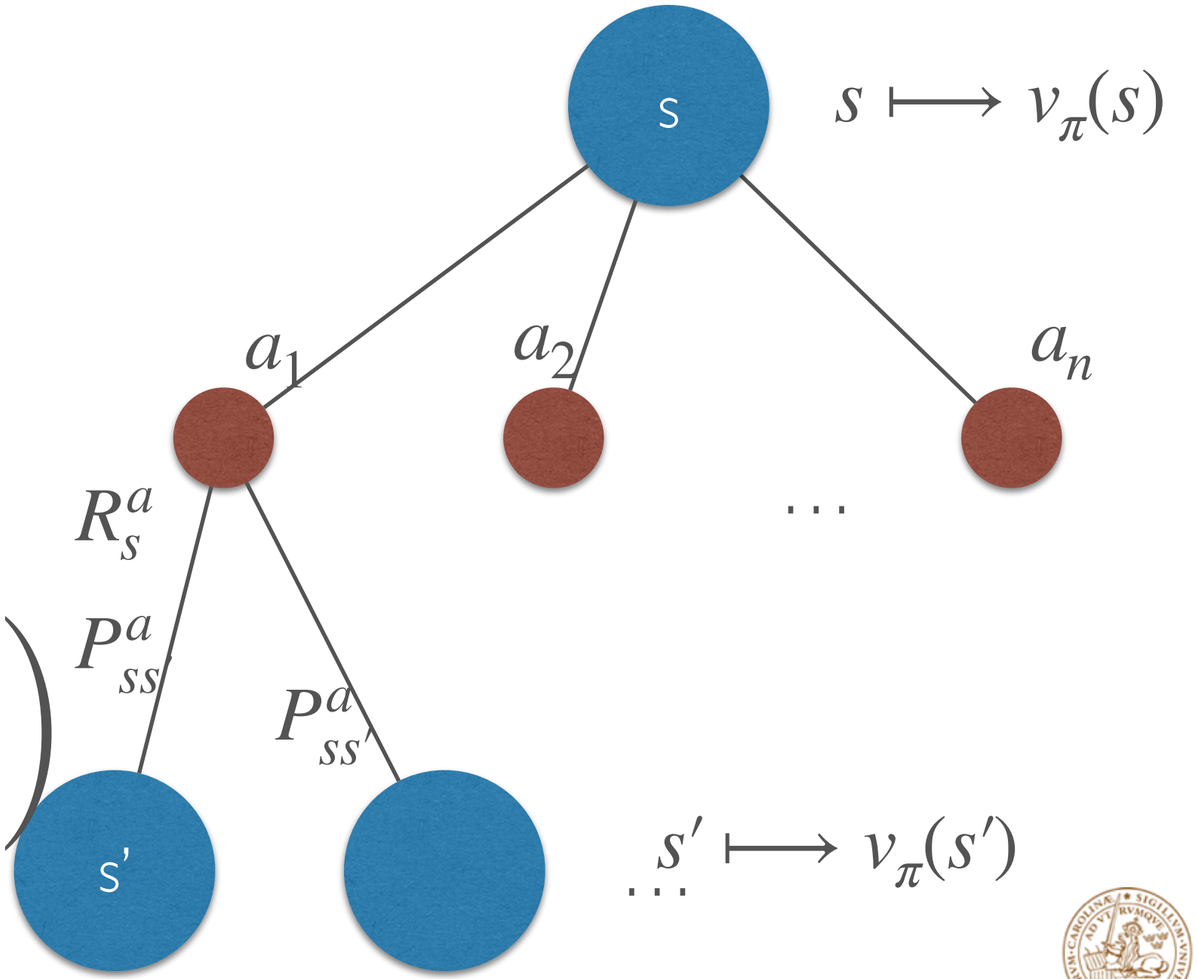
- Goal: evaluate a given policy π
- Solution: Iterative application of the Bellman expectation equation $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$

- 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'

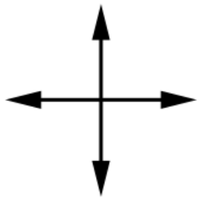
$$v_{k+1}(s) = \sum_{a \in \mathbf{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s') \right)$$

LUND UNIVERSITY

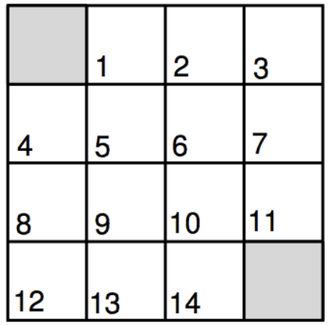
$$v_{k+1}(s) = \sum_{a \in \mathbf{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s') \right)$$



EXAMPLE: RANDOM POLICY IN THE SMALL GRIDWORLD



actions



$r = -1$ for all transition

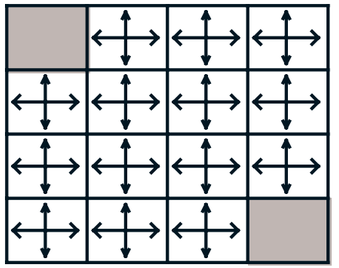
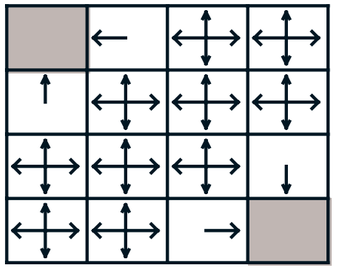
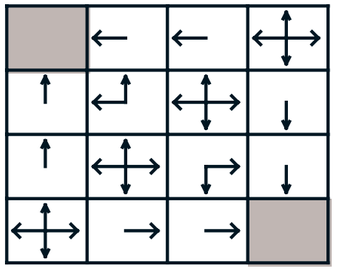
$\gamma = 1$

- Non-terminal states $1 \ldots 14$
- 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

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

LUND UNIVERSITY

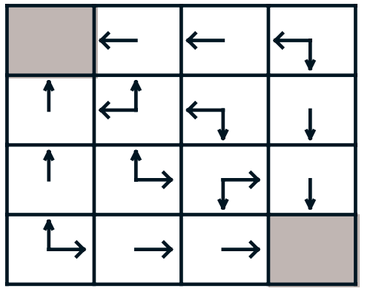
EXAMPLE: RANDOM POLICY IN THE SMALL GRIDWORLD

	v_k for the Random Policy	Greedy Policy w.r.t. v_k
$k = 0$	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	 ← 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

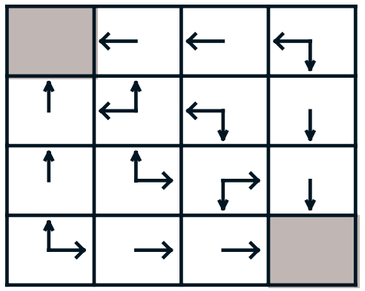
$k = 3$

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



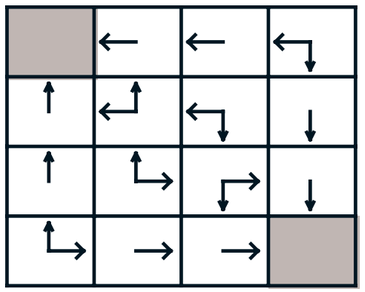
$k = 10$

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

DEMO

POLICY ITERATION

How to improve the policy

HOW TO IMPROVE THE POLICY

- Given a policy.
  - Evaluate the policy

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

  - Improve the policy by acting greedily with respect to v_π

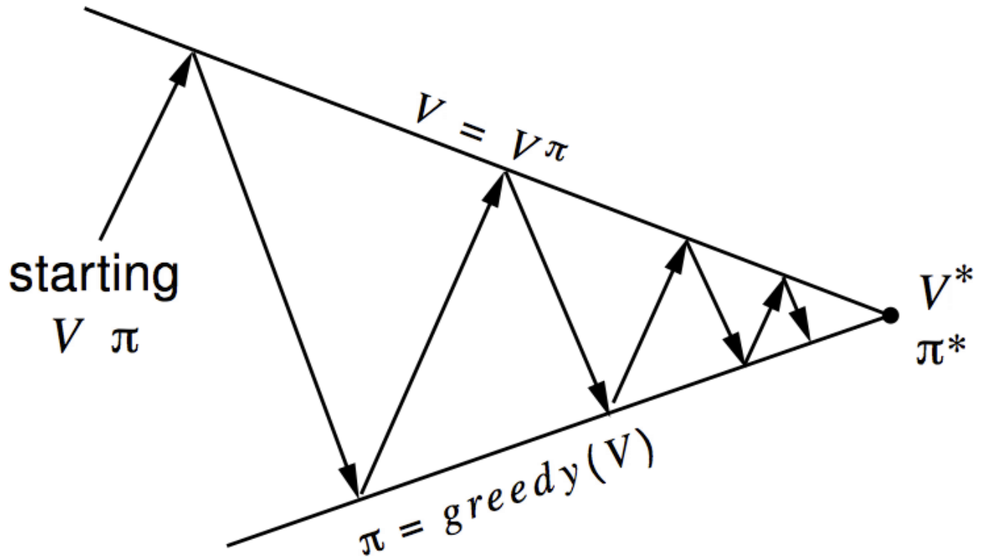
$$\pi' = \text{greedy}(v_\pi) \qquad \pi'(s) = \arg\max_{a \in A}\left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s')\right)$$

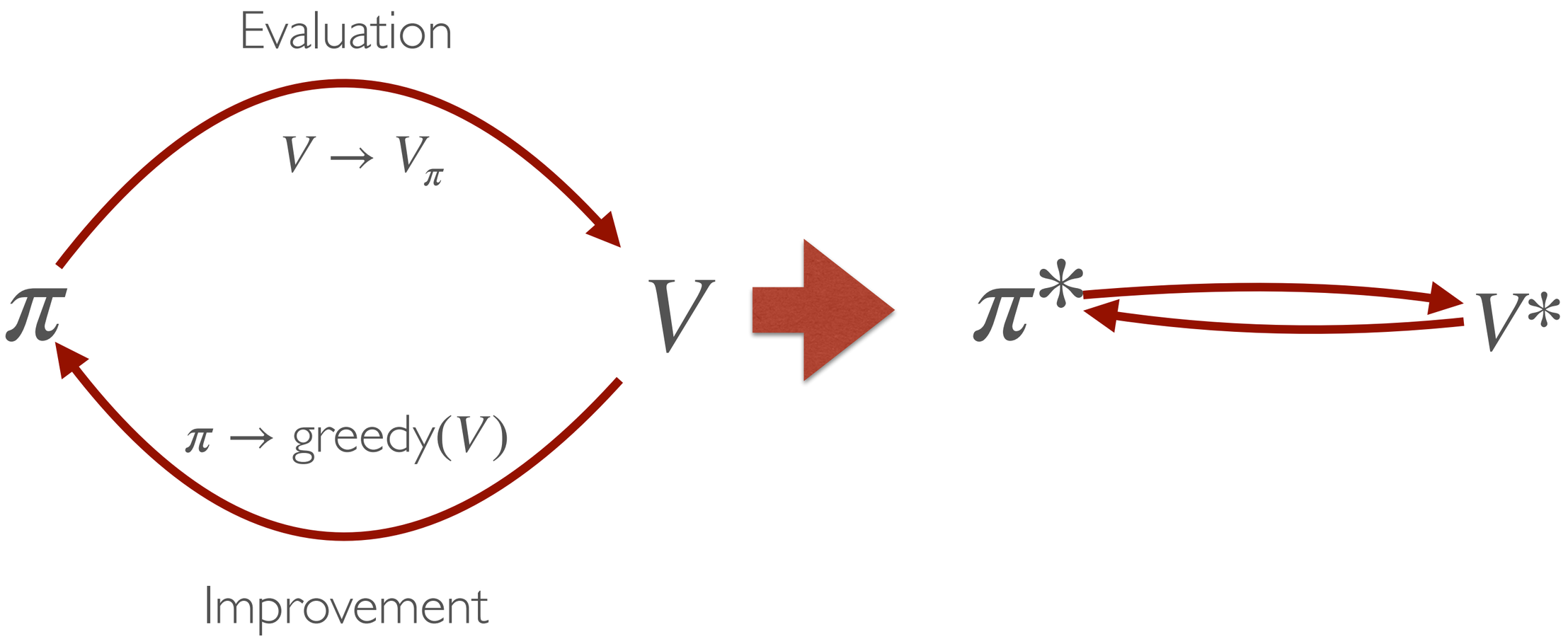
- In the Gridworld the improved policy was optimal $\pi' = \pi^*$
- In general, more iterations are needed
- But this process of **policy iteration** always converges to π^*

LUND UNIVERSITY

POLICY ITERATION

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



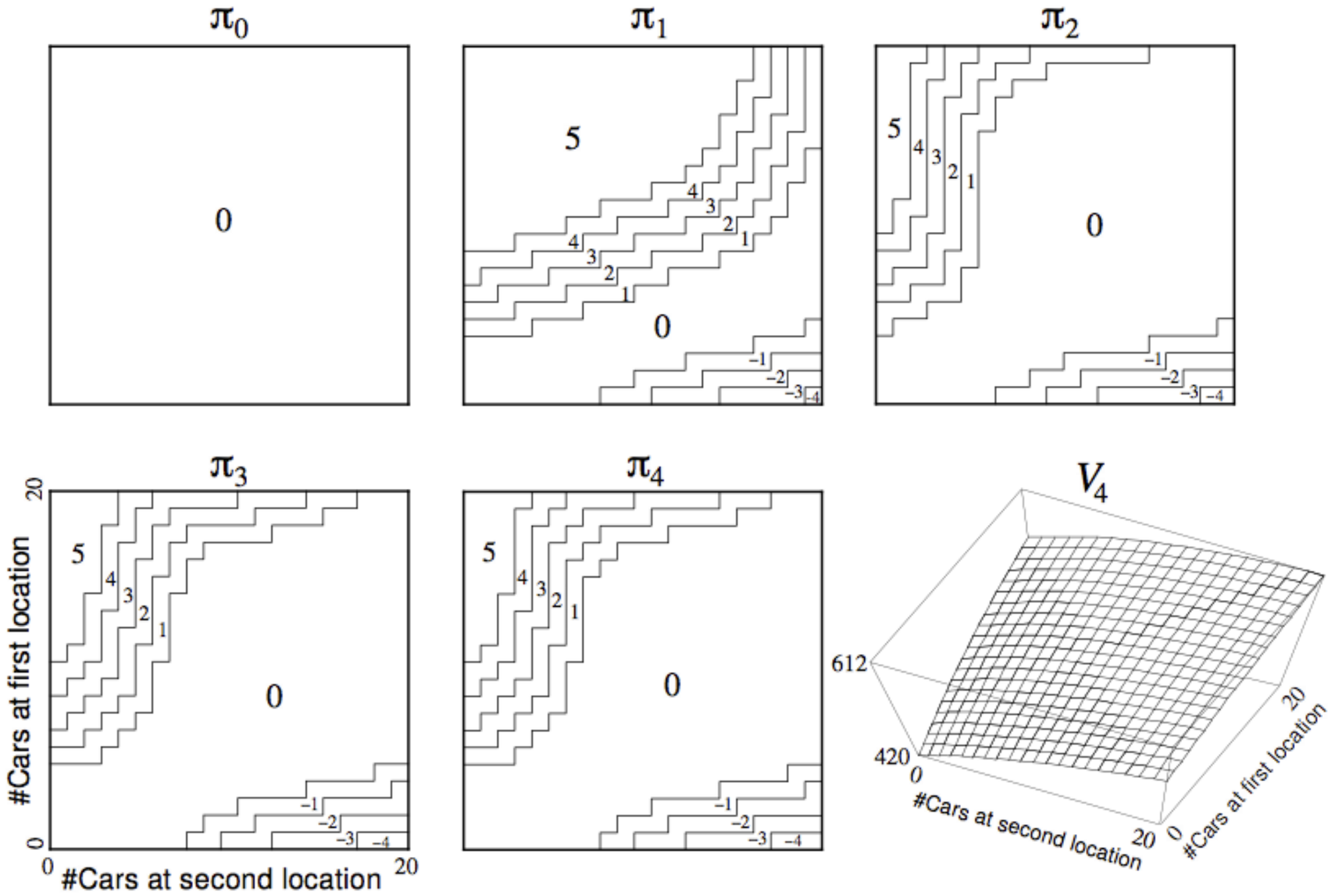


JACK'S CAR RENTAL

- 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

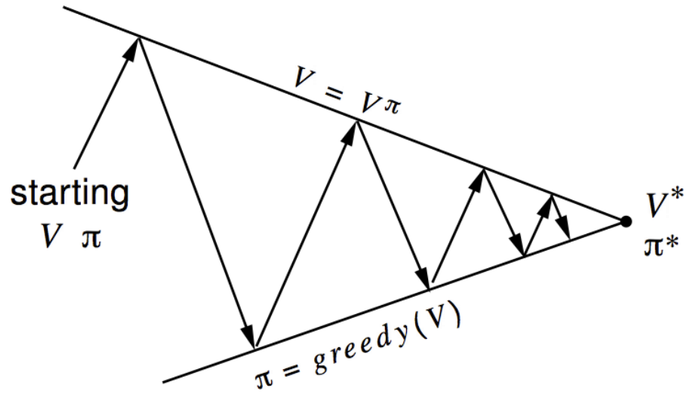
LUND UNIVERSITY

POLICY ITERATION IN JACK'S CAR RENTAL



SOME AMAZING FACTS

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



LUND UNIVERSITY

VALUE ITERATION

- Goal: find the optimal policy π
- Solution: Iterative application of the Bellman expectation equation $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$
- 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'

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

- Note: there is no(!!) explicit policy here.
- Intermediate value functions might not correspond to any policy.

LUND UNIVERSITY

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 $P^a_{s,s'}$ 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 Carlos Method

PLAN

- The transition function $P^a_{s,s'}$ and rewards R^s_s are not known, i.e, no model!!
  - 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

THE PROBLEM

- 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.

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

REMINDER

- Remember:
  - the return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
    - The value function is the expected return: $v_\pi(s) = E_\pi \left\{ G_t \,|\, S_t = s \right\}$

- The goal is to estimate the expected return from the observed episodes

$$E^1 : S_1^1, A_1^1, R_2^1, \ldots, S_k^1$$
$$E^2 : S_1^2, A_1^2, R_2^2, \ldots, S_k^2$$
$$\ldots$$
$$E^n : S_1^n, A_1^n, R_2^n, \ldots, S_k^n$$

LUND UNIVERSITY

MONTE CARLO POLICY EVALUATION

Simple Counting and Averaging

MONTE CARLO POLICY EVALUATION

Simple Counting and Averaging

MONTE CARLO POLICY EVALUATION

Simple Counting and Averaging

- For every episode E^i

MONTE CARLO POLICY EVALUATION

Simple Counting and Averaging

- For every episode E^i
  - Every times step t that state s is visited

MONTE CARLO POLICY EVALUATION

Simple Counting and Averaging

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

MONTE CARLO POLICY EVALUATION

Simple Counting and Averaging

- For every episode E^i
  - Every times step t that 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

Simple Counting and Averaging

- For every episode E^i
  - Every times step t that 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

Simple Counting and Averaging

- For every episode E^i
  - Every times step t that 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 $N(s) \rightarrow \infty$

EXAMPLE BLACKJACK

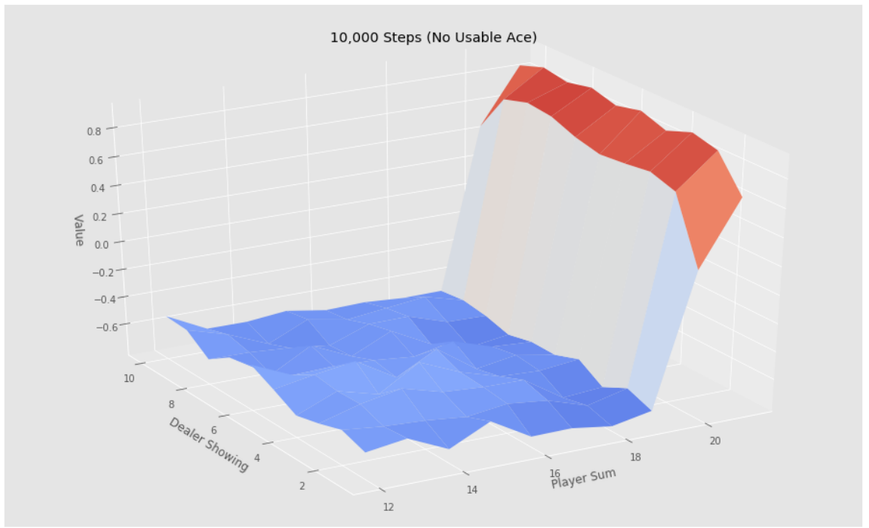
- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- Action **stick**: Stop receiving cards (and terminate)
- Action **twist**: Take another card (no replacement)
- Reward for **stick**:
  - +1 if sum of cards > sum of dealer cards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

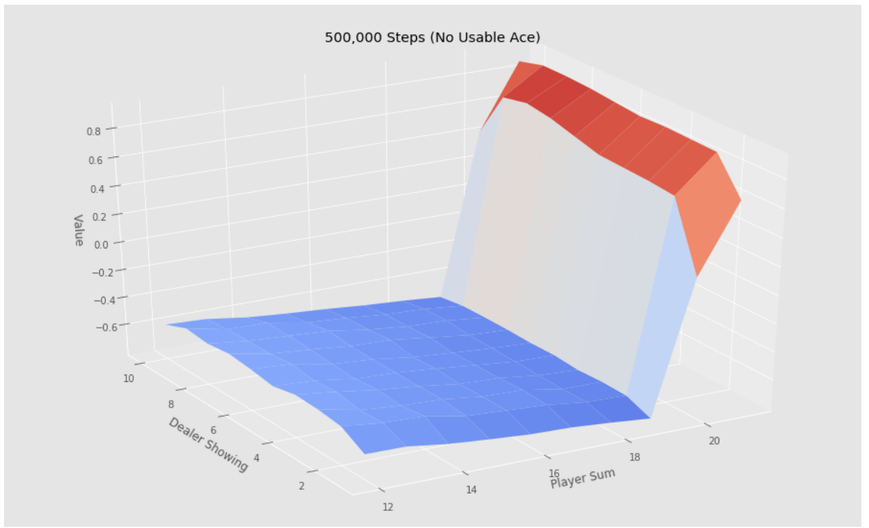
LUND UNIVERSITY

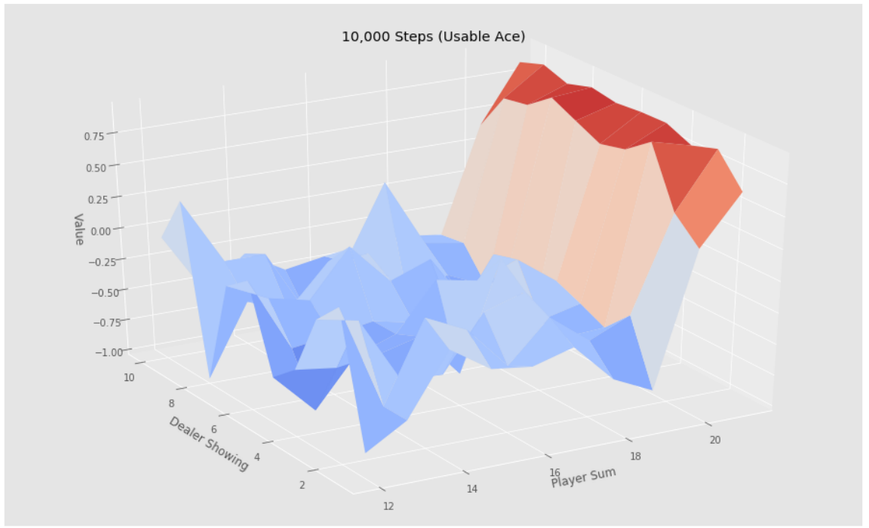
BLACKJACK VALUE FUNCTION WITH MONTE CARLO

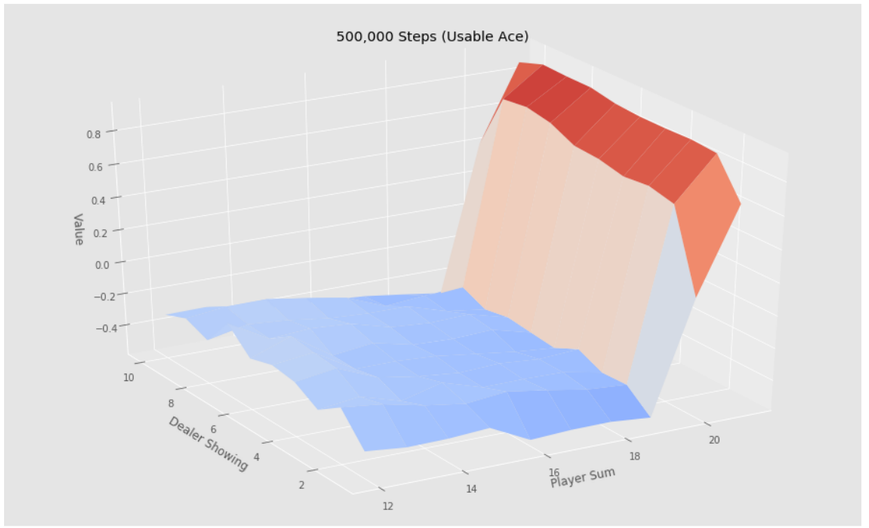
50,000 episodes

500,000 episodes









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

LUND UNIVERSITY

RECURSIVE COMPUTATION OF MEAN

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

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

OPTIMIZE POLICY

Monte Carlo Control

MONTE CARLO POLICY EVALUATION

- Update each $V(s)$ after each episode E^i:
  - 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: $V(s) \leftarrow S(s)/N(s)$

MONTE CARLO POLICY EVALUATION

- Update each $V(s)$ after each episode E^i:
  - 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: $V(s) \leftarrow S(s)/N(s)$

MONTE CARLO POLICY EVALUATION

- Update each $V(s)$ after each episode E^i:
  - 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: $V(s) \leftarrow S(s)/N(s)$

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

LUND UNIVERSITY

MONTE CARLO POLICY EVALUATION

- Update each $V(s)$ after each episode E^i:
  - 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: ~~$V(s) \leftarrow S(s)/N(s)$~~

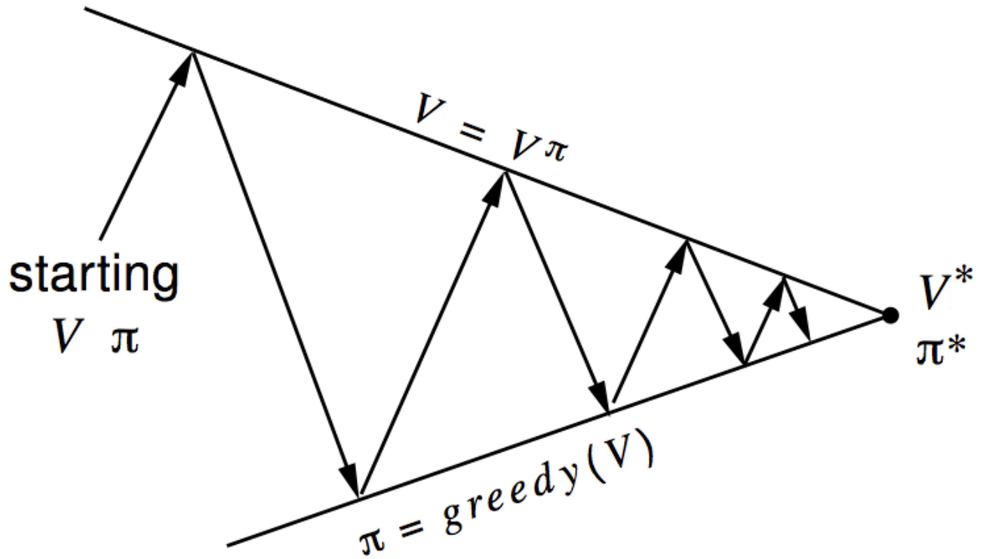
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} \left(G_t - V(S_t) \right)$$

LUND UNIVERSITY

POLICY ITERATION

- **Policy Evaluation**: Estimate v_π
  - Iterative Policy Evaluation

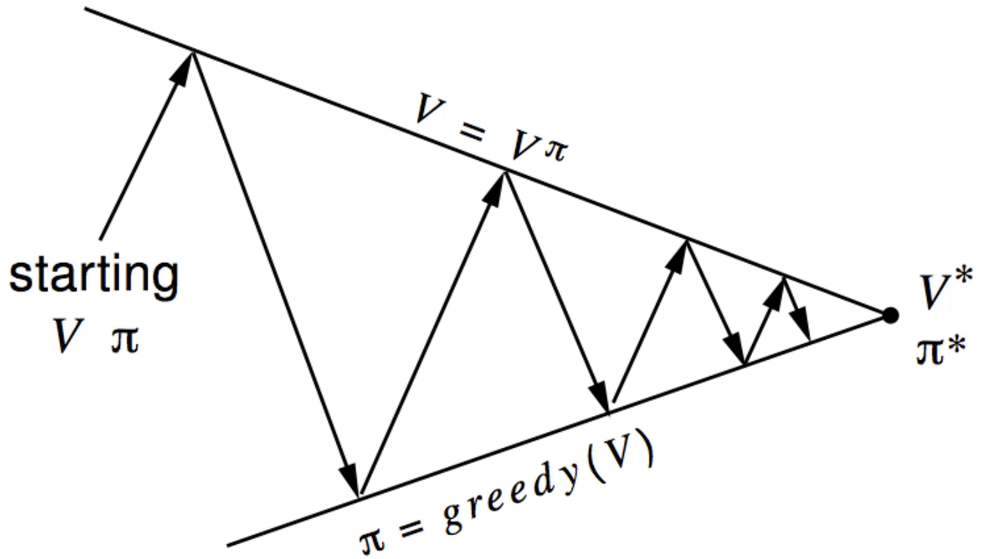
- **Policy improvement**: Generate $\pi' \geq \pi$
  - Greedy policy improvement



LUND UNIVERSITY

POLICY ITERATION

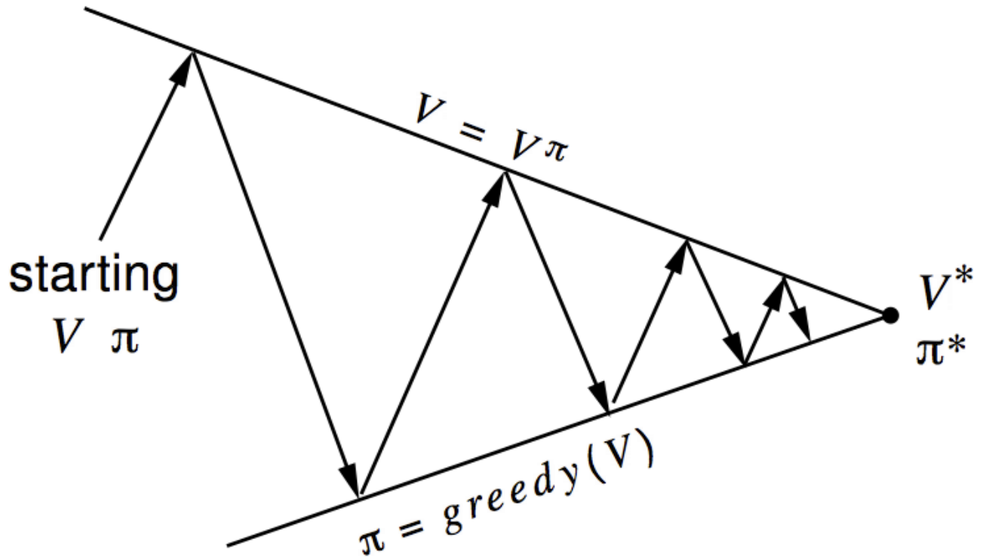
- **Policy Evaluation**: Estimate v_π
  - Iterative Policy Evaluation
  - Monte Carlo Policy Evaluation
- **Policy improvement**: Generate $\pi' \geq \pi$
  - Greedy policy improvement



LUND UNIVERSITY

POLICY ITERATION

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



LUND UNIVERSITY

COMPUTING THE POLICY

- Greedy policy improvement over $V(s)$ requires an MDP:

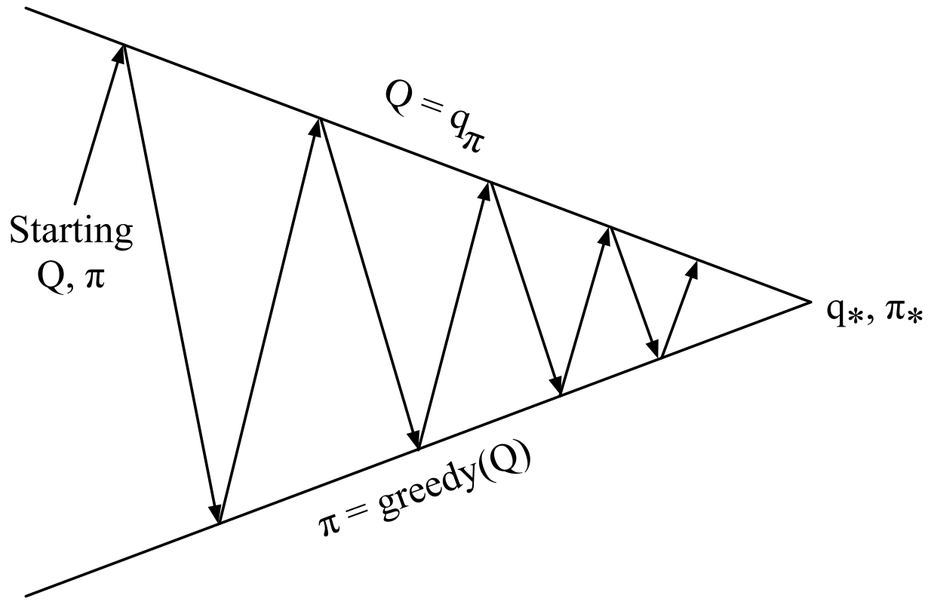
$$\pi'(s) = \arg\max_{a \in A} \left(R_s^a + P_{ss'}^a v_k(s') \right)$$

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

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

GENERALIZED POLICY ITERATION FOR ACTION VALUE FUNCTION

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



LUND UNIVERSITY

EXAMPLE OF A GREEDY ACTION

- Two restaurants you can choose from
  - The top one doesn't look cool
    - $V(top) = 0$
  - You go to the bottom one and get a reward +1
    - $V(bottom) = +1$
  - You go to the bottom one and get a reward +2
    - $V(bottom) = +2$
  - You go to the bottom one and get a reward +1
    - $V(bottom) = +1$
  - …
  - Are you sure you chose the right restaurant?

LUND UNIVERSITY

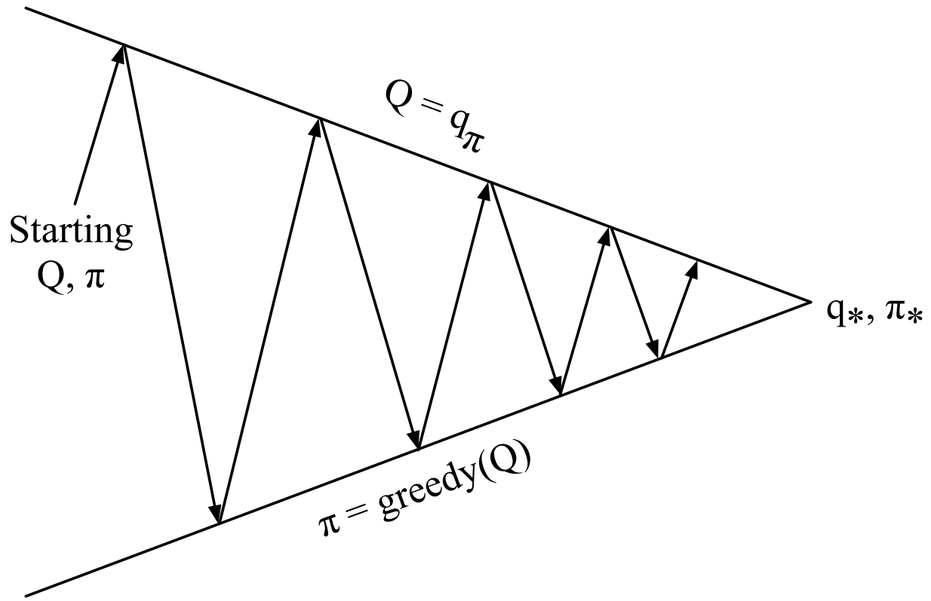
ϵ-GREEDY EXPLORATION

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

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

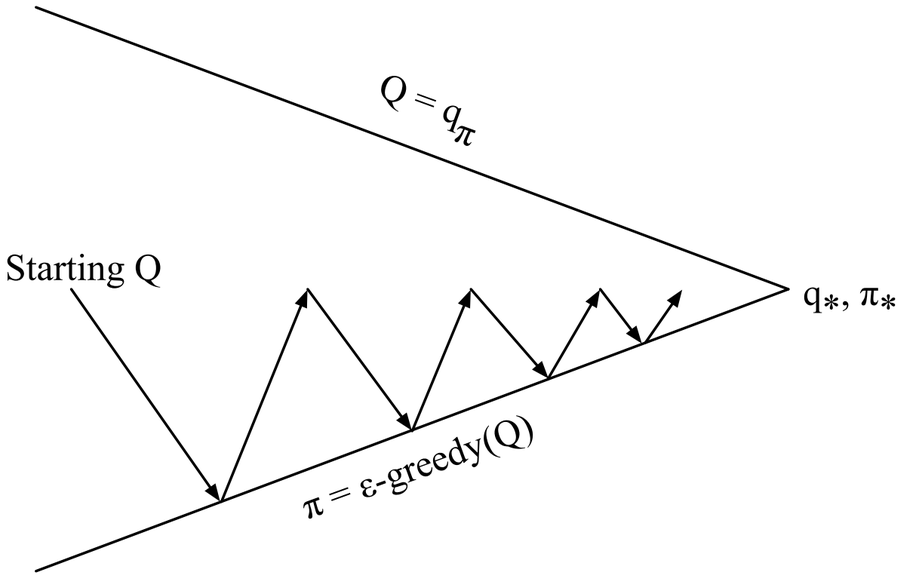
GENERALIZED POLICY ITERATION FOR ACTION VALUE FUNCTION

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



GENERALIZED POLICY ITERATION FOR ACTION VALUE FUNCTION

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



Greedy in the Limit of Infinite Exploration (GLIE) (w/o proof): $\epsilon_k = 1/k$

LUND UNIVERSITY

GLIE MONTE CARLO CONTROL

- Generate a new episode E^k using your policy $\pi : S_1, A_1, R_2, S_2, A_2, R_3, \ldots, R_T$
- For each state S_t and action A_t in the episode
  - $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left(G_t - Q(S_t, A_t)\right)$
- Improve the policy based on the new $Q(S, A)$
  - $\epsilon \leftarrow 1/k$
  - $\pi \leftarrow \epsilon$-greedy(Q)

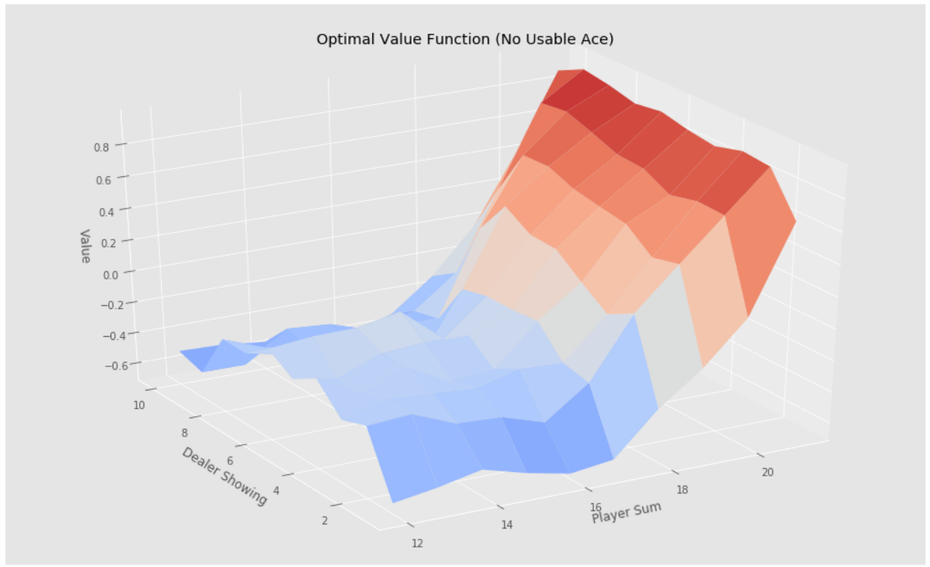
LUND UNIVERSITY

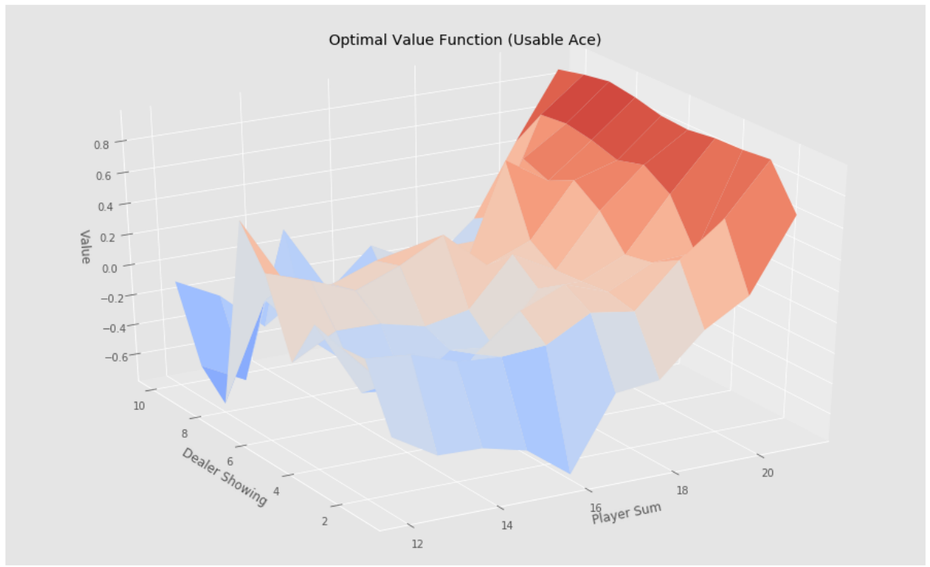
BACK TO BLACK JACK

LUND UNIVERSITY

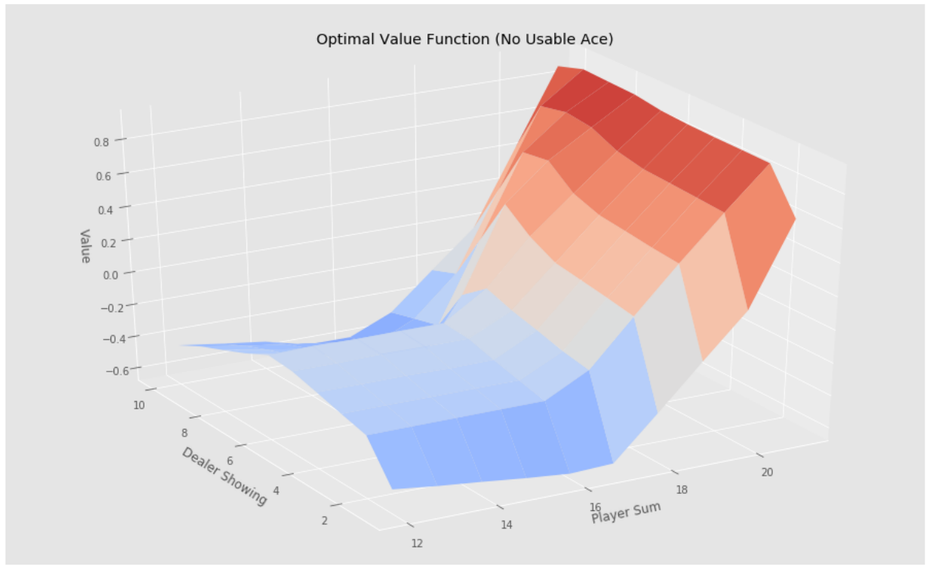
MONTE CARLO CONTROL IN BLACK JACK

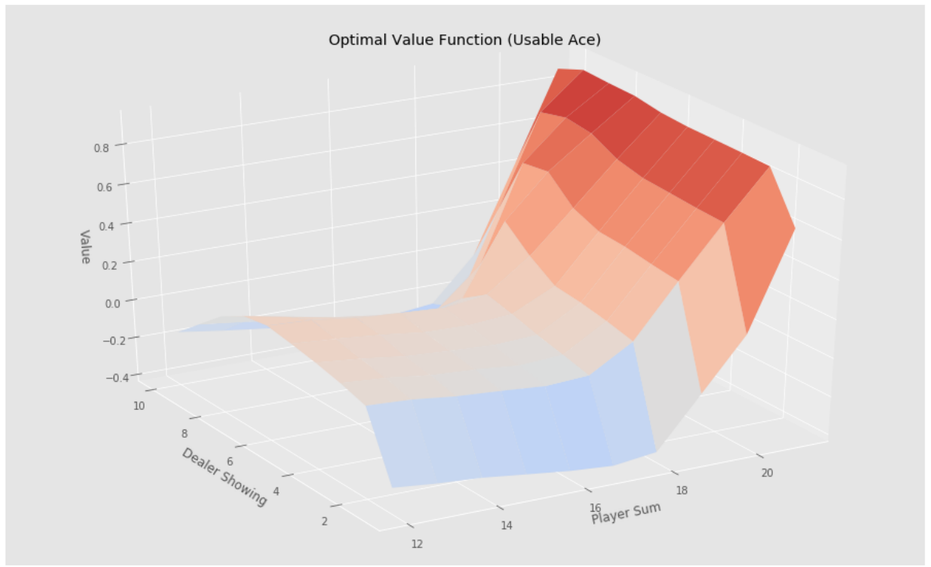
50.000 episodes





500.000 episodes





LUND UNIVERSITY

SUMMARY

- First real RL approach to optimise for finding a policy
- model-free
- 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).