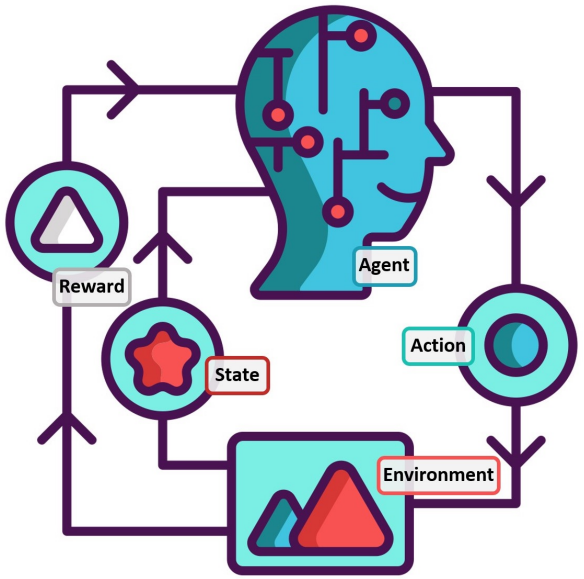
AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Lecture #05
Bellman Equations & Dynamic Programming for MDPs

Gian Antonio Susto



Announcements before starting

- Next lab on friday at 14:30 on Dynamic Programming
- Optional material (previous years recorded lectures by Prof. Carli) on the connection between Dynamic Programming and Optimal Control

The story so far

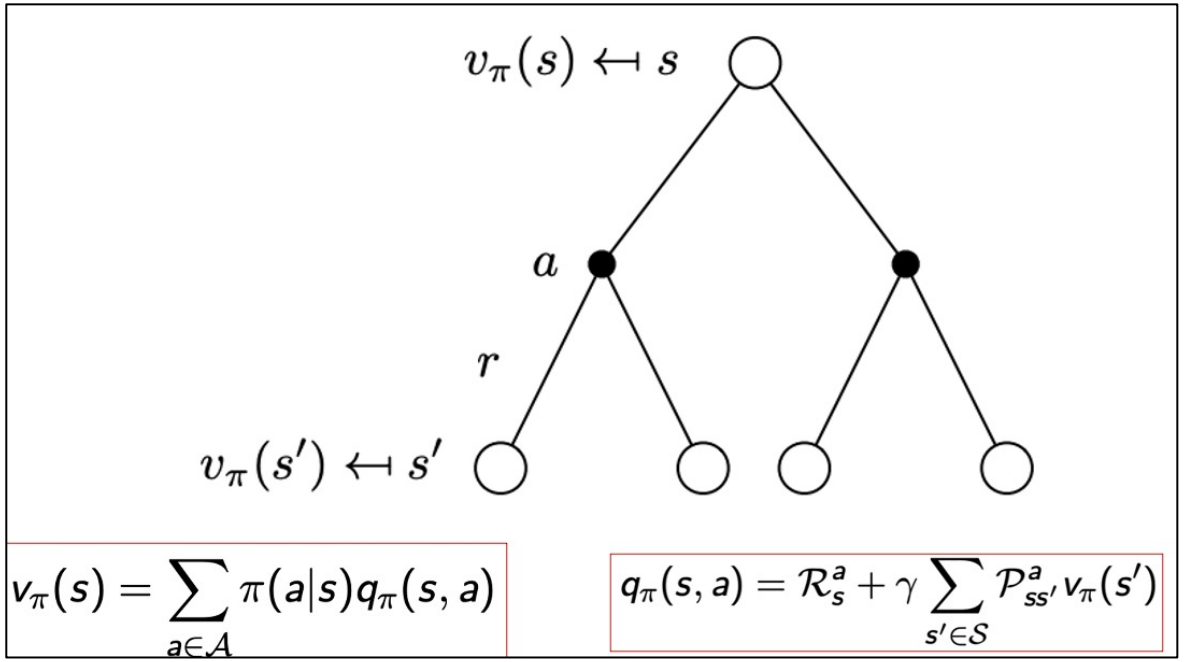
Richard E. Bellman

Bellman equations are our tool to 'solve' Markov Reward Processes (MRPs) and MDPs thanks to their recursive nature:

MRP	Bellman equation: for finding value functions	Linear: we can use it for 'small' MRPs. We need to resort to iterative approaches for 'large' MRPs
MDP	Bellman expectation equation: for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches for 'large' MDPs
MDP	Bellman optimality equation: for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

The story so far

Bellman equations are our tool to 'solve' Markov Reward Processes (MRPs) and MDPs thanks to their recursive nature:



MRP	Bellman equation: for finding value functions	Linear: we can use it for 'small' MRPs. We need to resort to iterative approaches for 'large' MRPs
MDP	Bellman expectation equation: for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches for 'large' MDPs
MDP	Bellman optimality equation: for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

The story so far

Bellman equations are our tool to 'solve' Markov Reward Processes (MRPs) and MDPs thanks to their recursive nature:

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

MRP	Bellman equation: for finding value functions	Linear: we can use it for 'small' MRPs. We need to resort to iterative approaches for 'large' MRPs
MDP	Bellman expectation equation: for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches for 'large' MDPs
MDP	Bellman optimality equation: for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

The story so far

Bellman equations are our tool to 'solve' Markov Reward Processes (MRPs) and MDPs thanks to their recursive nature:

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

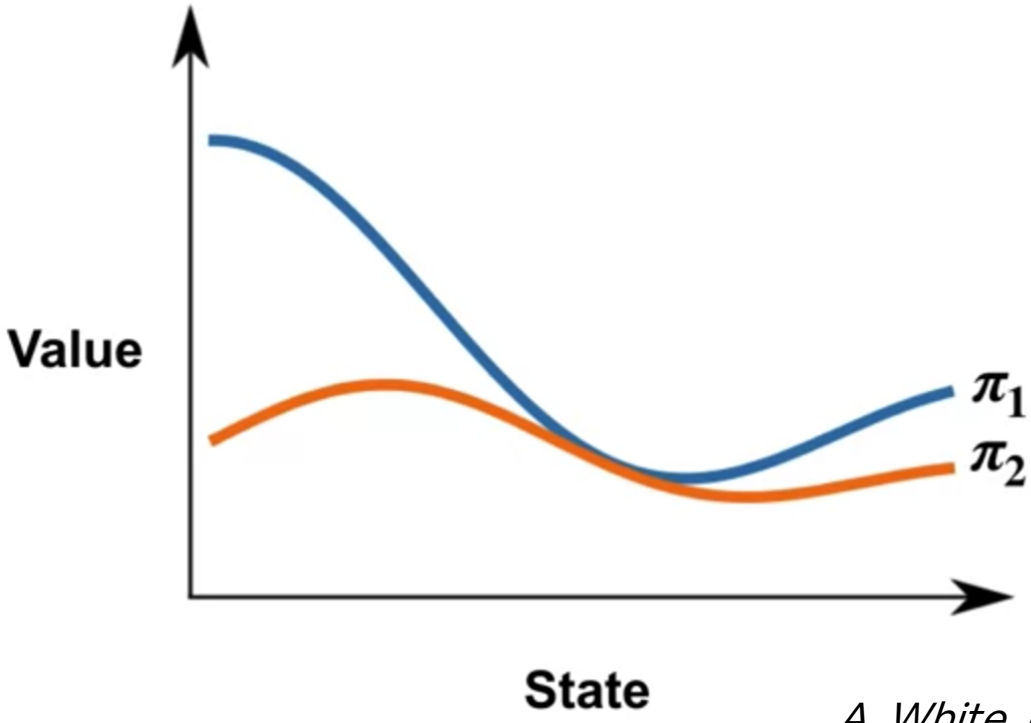
$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

MRP	Bellman equation: for finding value functions	Linear: we can use it for 'small' MRPs. We need to resort to iterative approaches for 'large' MRPs
MDP	Bellman expectation equation: for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches for 'large' MDPs
MDP	Bellman optimality equation: for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

The story so far: Optimal Policy

Our final goal is to find an optimal policy: the best way to act in a MDP!

We define an order over policies: $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for all s



$$\pi_1 \geq \pi_2$$

A. White, M. White 'Fundamentals of Reinforcement Learning'

The story so far: Optimal Policy

Theorem

For any MDP:

1. There exists an optimal policy π_* such that $\pi_* \geq \pi$ for all possible π
2. All optimal policies achieve the optimal value function

$$v_{\pi_*}(s) = v_*(s)$$

3. All optimal policies achieve the optimal value function

$$q_{\pi_*}(s, a) = q_*(s, a)$$

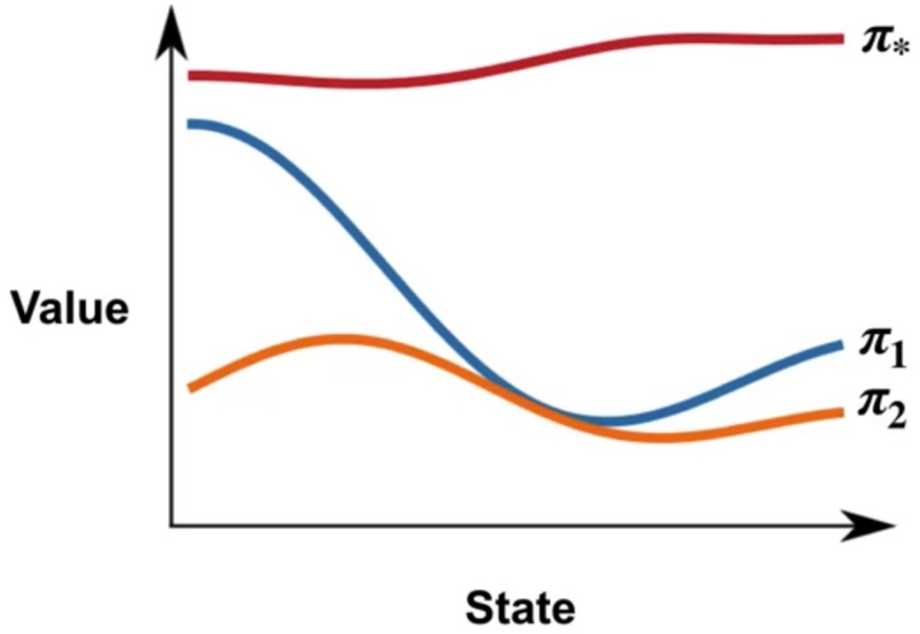
The story so far: Finding an Optimal Policy

- How to operationally find an optimal policy?

A simple approach, if we have $q_*(s,a)$, if by maximizing over:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

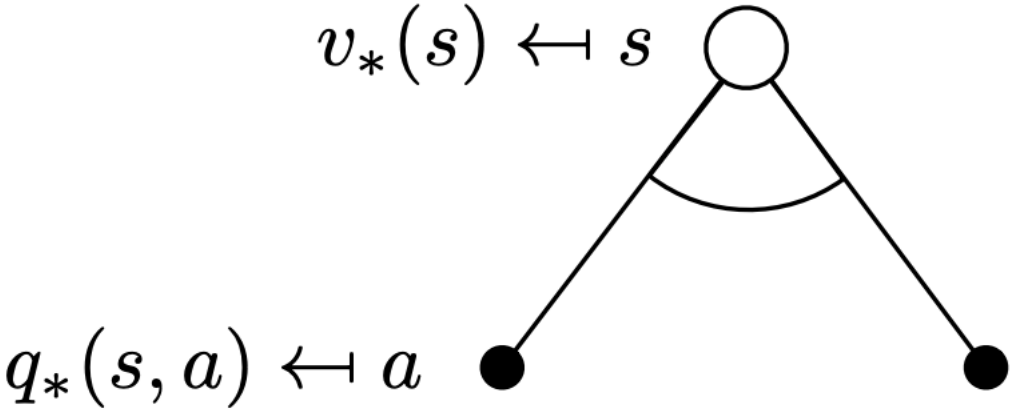
- There is always a deterministic optimal policy for any MDP



A. White, M. White 'Fundamentals of Reinforcement Learning'

Bellman Optimality Equation 1/3

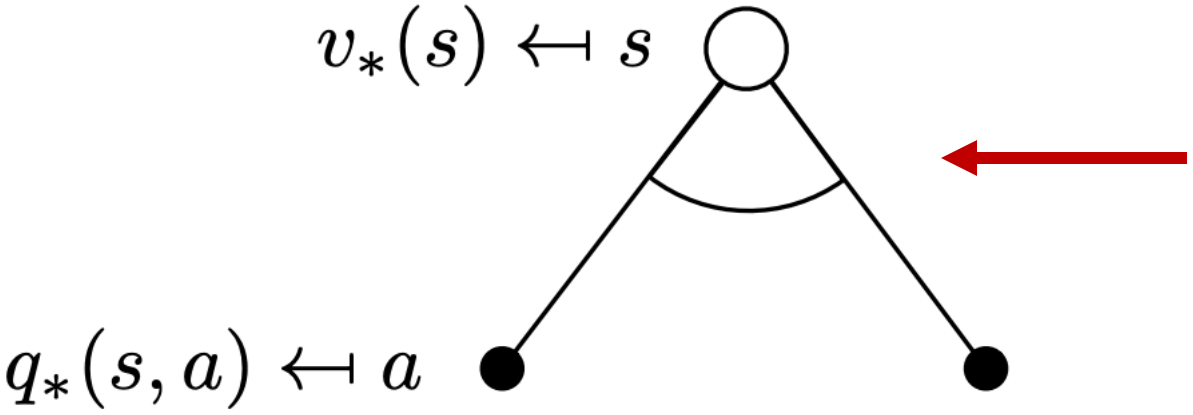
- Not to be confused with the Bellman Expectation Equation, that holds for a generic policy and it is a way to recursively define v_π and q_π
- The Bellman Optimality Equation is a way to define the optimal v_* with itself: we exploit again the 1-step look-ahead principle



$$v_*(s) = \max_a q_*(s,a)$$

Bellman Optimality Equation 1/3

- Not to be confused with the Bellman Expectation Equation, that holds for a generic policy and it is a way to recursively define v_π and q_π
- The Bellman Optimality Equation is a way to define the optimal v_* with itself: we exploit again the 1-step look-ahead principle

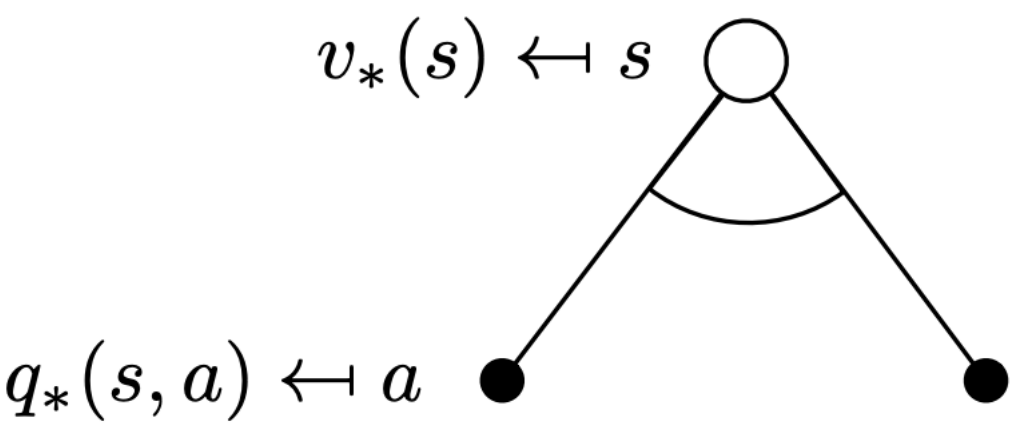


This symbol indicates the max over all possible choices for the action value: we don't need to average

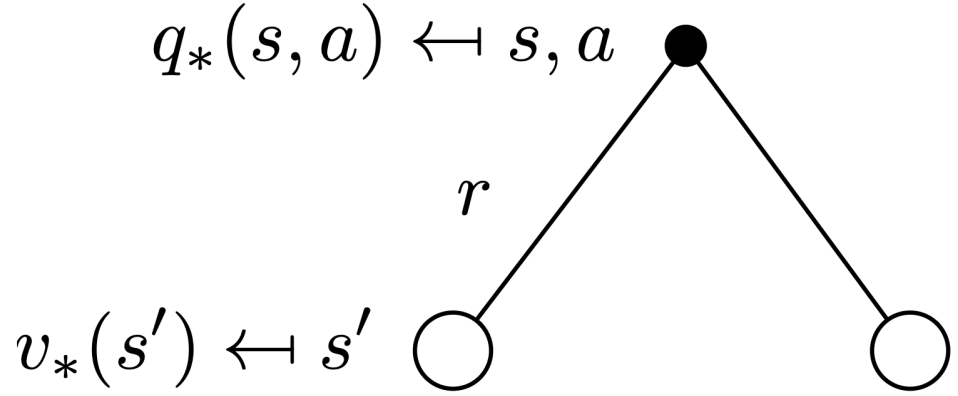
$$v_*(s) = \max_a q_*(s,a)$$

Bellman Optimality Equation 1/3

- Not to be confused with the Bellman Expectation Equation, that holds for a generic policy and it is a way to recursively define v_π and q_π
- The Bellman Optimality Equation is a way to define the optimal v_* with itself: we exploit again the 1-step look-ahead principle



Agent: here we pick the 'best' action

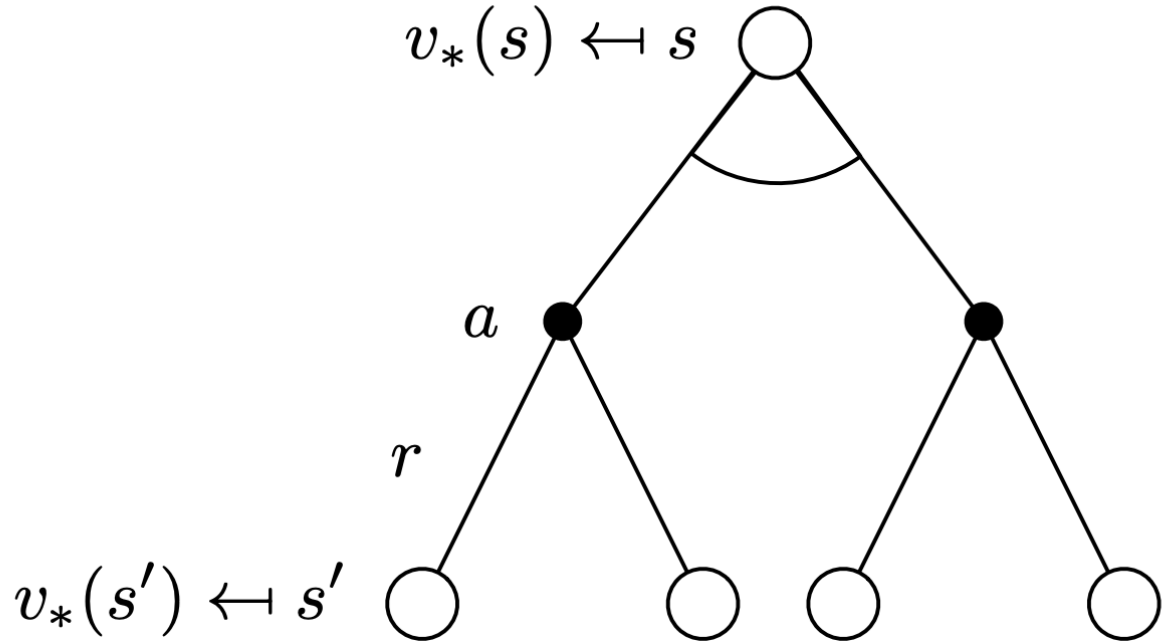


Environment: now we need to average!

$$v_*(s) = \max_a q_*(s,a)$$

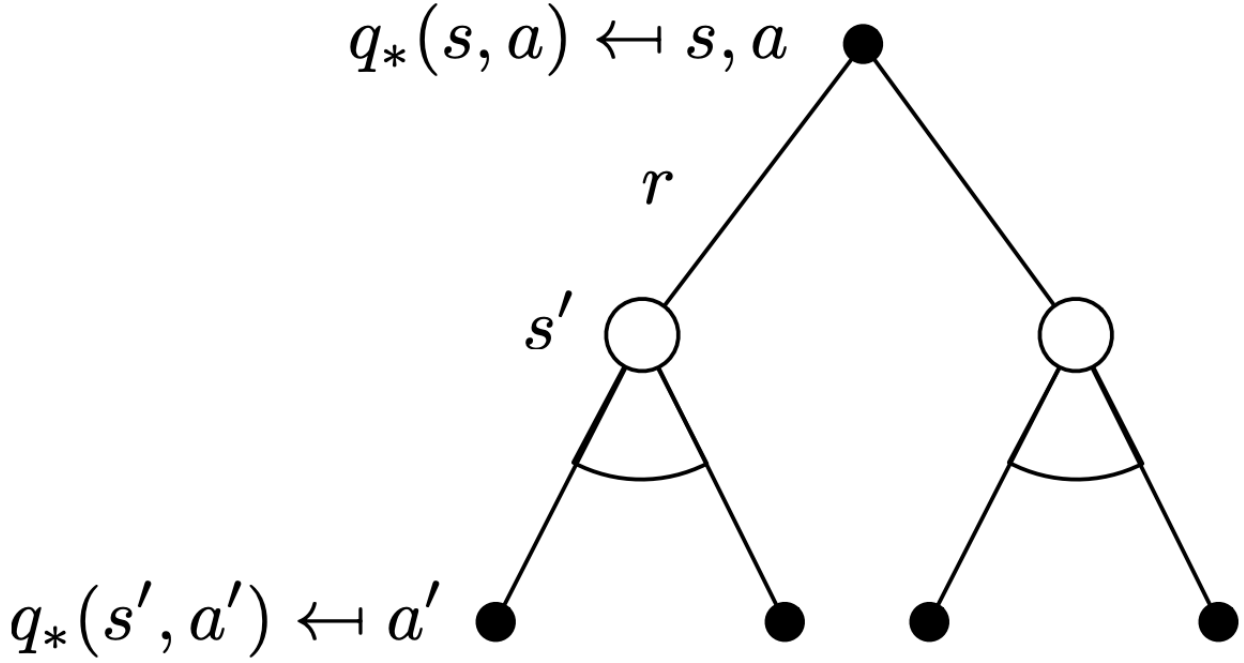
$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation 2/3



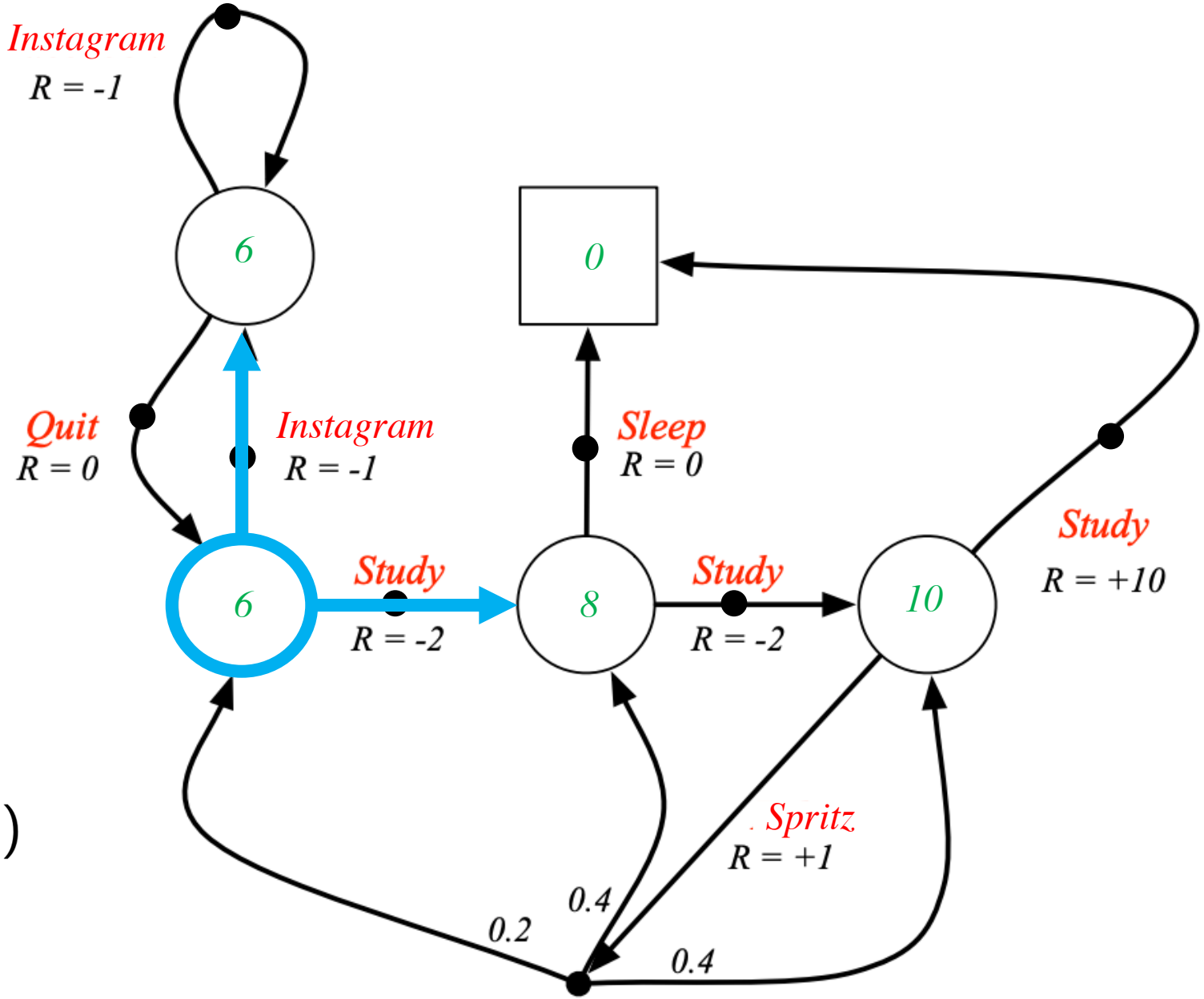
$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation 3/3



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

Bellman Optimal equation in the Student MDP



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

$$6 = max\ \{-2 + 8,\ \ -1 + 6\}$$

D. Silver 'Reinforcement Learning' @UCL

Bellman Optimality Equation

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

- Since there is a non-linear operation (a max) we don't have a closed-form solution
- We will resort to iterative methods: value iteration, policy iteration, q-learning, SARSA, …

Summarizing

Markov Decision Processes (MDPs) formally describe an environment for Reinforcement Learning

i. Markov Processes $\langle \mathcal{S}, \mathcal{P} \rangle$

ii. Markov Reward Processes $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

iii. Markov Decision Processes (MDPs) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

From now on we will deal with MDPs!

Markov Reward Processes and MDPs can be 'solved' with respect to:

- Deriving the value function and the action-value function (in MDP w.r.t. a policy)
- Finding the best policy

Summarizing

Markov Decision Processes (MDPs) formally describe an environment for Reinforcement Learning

i. Markov Processes $\langle \mathcal{S}, \mathcal{P} \rangle$
ii. Markov Reward Processes $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
iii. Markov Decision Processes (MDPs) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

From now on we will deal with MDPs!

Markov Reward Processes and MDPs can be 'solved' with respect to:

- Deriving the value function and the action-value function (in MDP w.r.t. a policy) -> (POLICY) EVALUATION
- Finding the best policy -> CONTROL / POLICY IMPROVEMENT

Richard E. Bellman

Summarizing

Bellman equations are our tool to 'solve' Markov Reward Processes (MRPs) and MDPs thanks to their recursive nature:

MRP	Bellman equation: for finding value functions	Linear: we can use it for 'small' MRPs. We need to resort to iterative approaches for 'large' MRPs
MDP	Bellman expectation equation: for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches for 'large' MDPs
MDP	Bellman optimality equation: for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

Summarizing

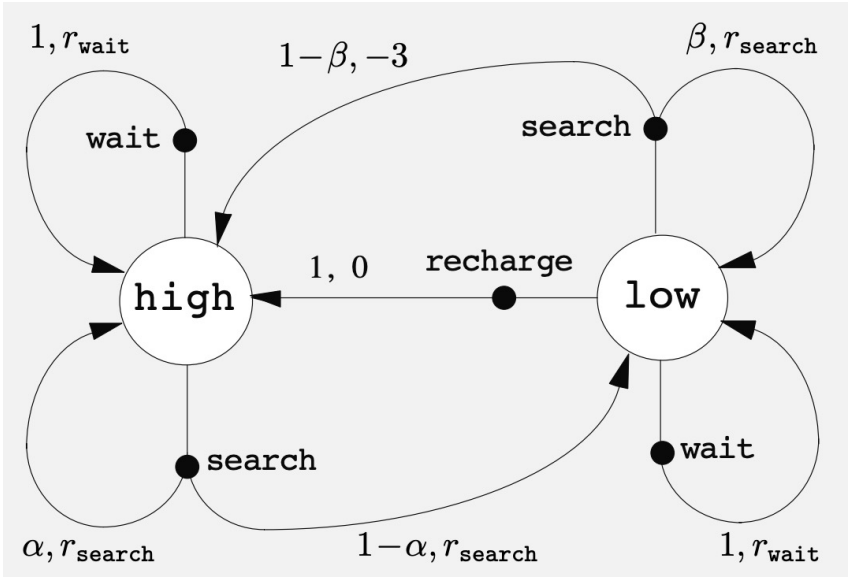
Richard E. Bellman

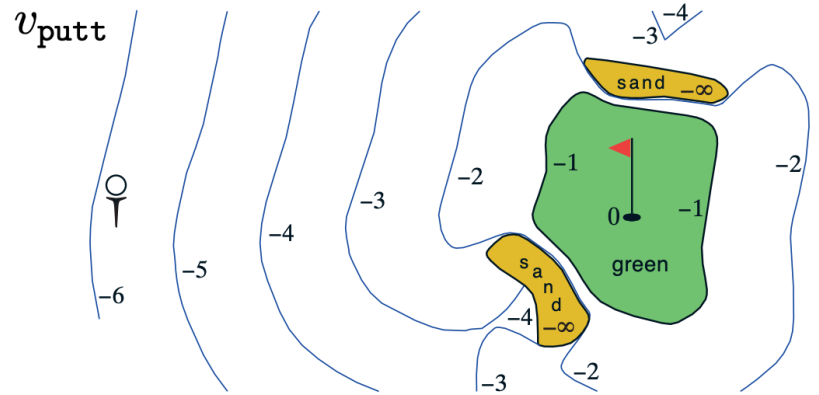
Bellman equations are our tool to 'solve' Markov Reward Processes (MRPs) and MDPs thanks to their recursive nature:

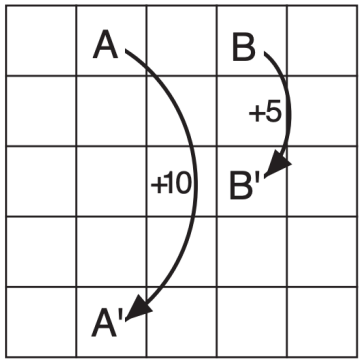
In the book

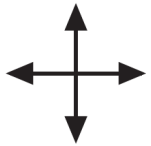
~~MRP~~	~~Bellman equation: for finding value functions~~	~~Linear: we can use it for 'small' MRPs. We need to resort to iterative approaches for 'large' MRPs~~
MDP	Bellman ~~expectation~~ equation: for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches for 'large' MDPs
MDP	Bellman optimality equation: for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

Examples in the Book (chapter 3)



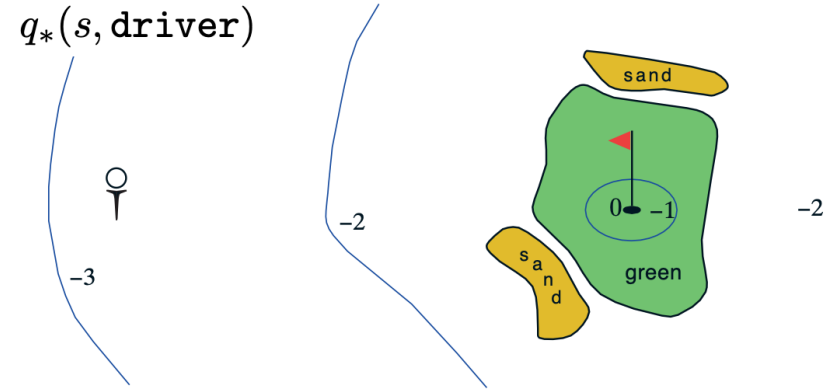




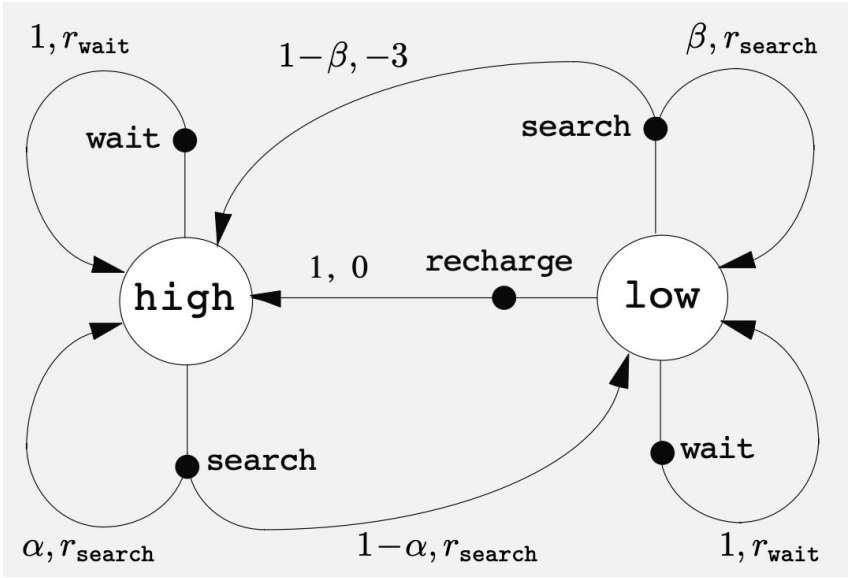


Actions

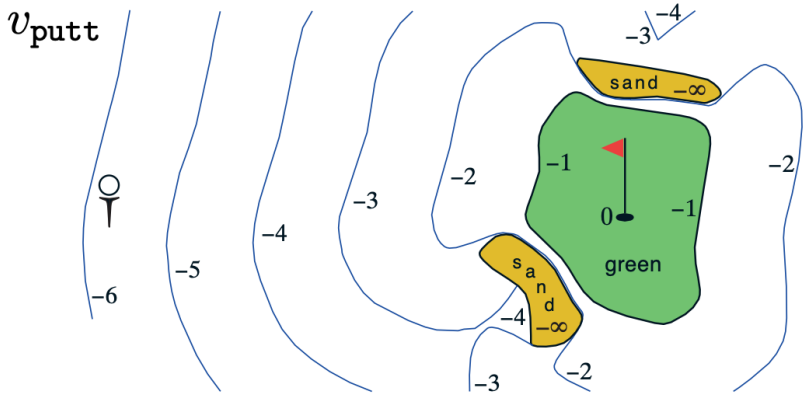
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

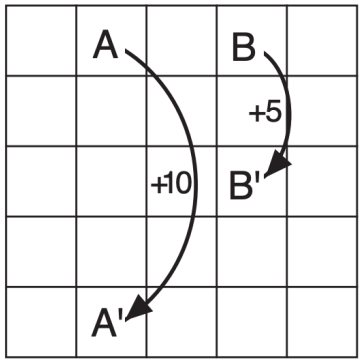


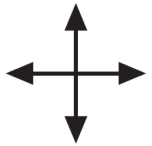
Examples in the Book (chapter 3)



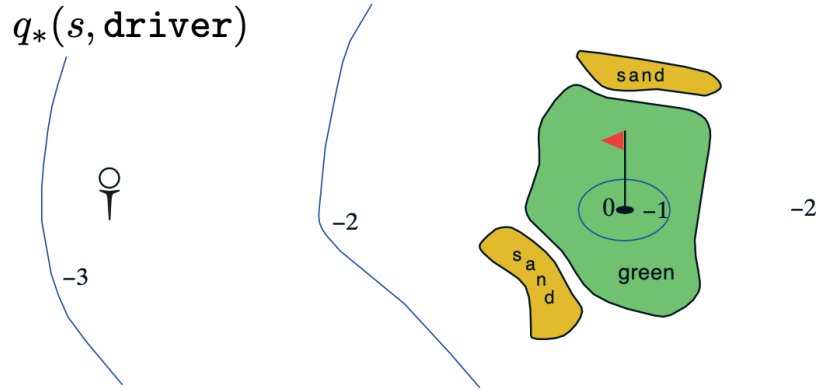
We will not see these in the laboratory (we will move directly to Chapter 4 examples): we suggest to take a look by yourselves!






Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

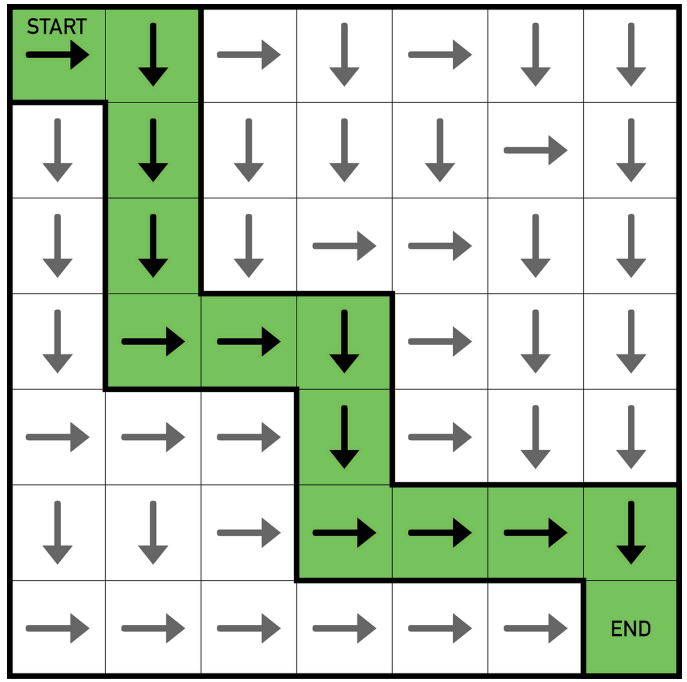


MDP: Exam

- All the content of Chapter 3 of the book may be exam material

- Keep in mind that the book presents directly MDPs: use these slides for a definition of Markov Processes and Markov Reward Processes

Dynamic Programming for MDPs (Chapter 4)



The story so far…

- MDPs [redacted] will be a fundamental formalization tool for RL problems, even if [redacted] are typically not given in 'real' RL problems

- Our goal is to 'solve the MDPs'

1. (Policy) evaluation/prediction: understand how good a policy π is, ie. typically deriving the value function $v_\pi(s)$ and q_π

   INPUT: MDP and a policy π

   OUTPUT: $v_\pi(s)$ and/or $q_\pi(s,a)$

2. (Policy improvement) control: improve the current policy π/find an optimal policy π^* typically deriving the value function $v^*(s)$ and the action-value function $q^*(s,a)$

   INPUT: MDP

   OUTPUT: π^*optionally $v^*(s)$ and/or $q^*(s,a)$

The story so far…

- MDPs $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ will be a fundamental formalization tool for RL problems, even if $\mathcal{P}$ and $\mathcal{R}$ are typically not given in 'real' RL problems

- Our goal is to 'solve the MDPs'

1. (Policy) evaluation/prediction: understand how good a policy π is, ie. typically deriving the value function $v_\pi(s)$ and q_π

   INPUT: MDP and a policy π

   OUTPUT: $v_\pi(s)$ and/or $q_\pi(s, a)$

2. (Policy improvement) control: improve the current policy π/find an optimal policy π^* typically deriving the value function $v^*(s)$ and the action-value function $q^*(s, a)$

   INPUT: MDP

   OUTPUT: π^*optionally $v^*(s)$ and/or $q^*(s, a)$

The story so far…

- MDPs $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ will be a fundamental formalization tool for RL problems, even if $\mathcal{P}$ and $\mathcal{R}$ are typically not given in 'real' RL problems

- Our goal is to 'solve the MDPs'

1. (Policy) evaluation/prediction: understand how good a policy π is, ie. typically deriving the value function $v_\pi(s)$ and q_π

   INPUT: MDP and a policy π

   OUTPUT: $v_\pi(s)$ and/or $q_\pi(s,a)$

2. (Policy improvement) control: improve the current policy π/find an optimal policy π^* typically deriving the value function $v^*(s)$ and the action-value function $q^*(s,a)$

   INPUT: MDP

   OUTPUT: π^*optionally $v^*(s)$ and/or $q^*(s,a)$

There can be multiple optimal policies, but they will all share the same $v^*(s)$ and $q^*(s,a)$

The story so far…

For today and partially next lecture, we will still make this hypothesis: $\mathcal{P}$ and $\mathcal{R}$ known…

- MDPs $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ will be a fundamental formalization tool for RL problems, even if $\mathcal{P}$ and $\mathcal{R}$ are typically not given in 'real' RL problems

- Our goal is to 'solve the MDPs'

1. (Policy) evaluation/prediction: understand how good a policy π is, ie. typically deriving the value function $v_\pi(s)$ and q_π

   INPUT: MDP and a policy π

   OUTPUT: $v_\pi(s)$ and/or $q_\pi(s, a)$

2. (Policy improvement) control: improve the current policy π/find an optimal policy π^* typically deriving the value function $v^*(s)$ and the action-value function $q^*(s, a)$

   INPUT: MDP

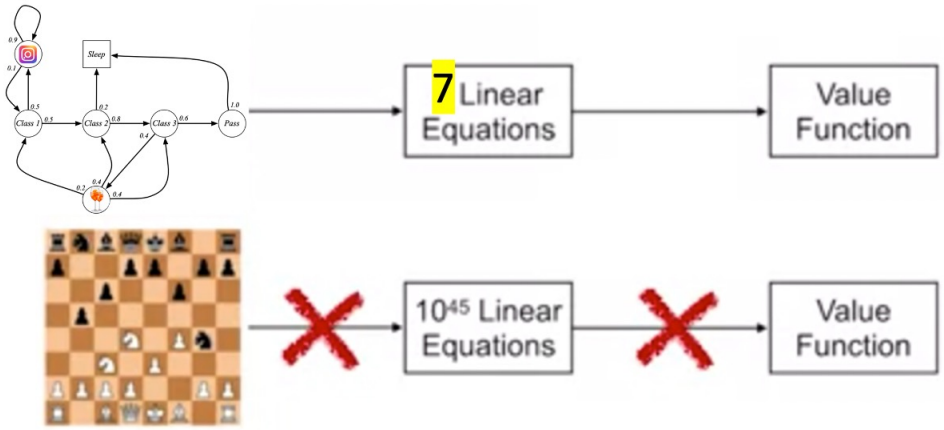
   OUTPUT: π^*optionally $v^*(s)$ and/or $q^*(s, a)$

The story so far…

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

- Even if $\mathcal{P}$ and $\mathcal{R}$ are known, solving (prediction/control) the MDP may not be straightforward

- While the evaluation problem in some cases can be solved with a set of linear equations, in most cases we need to resort to Dynamic Programming for solving MDPs (remember that Bellman Optimal Equation is non-linear)



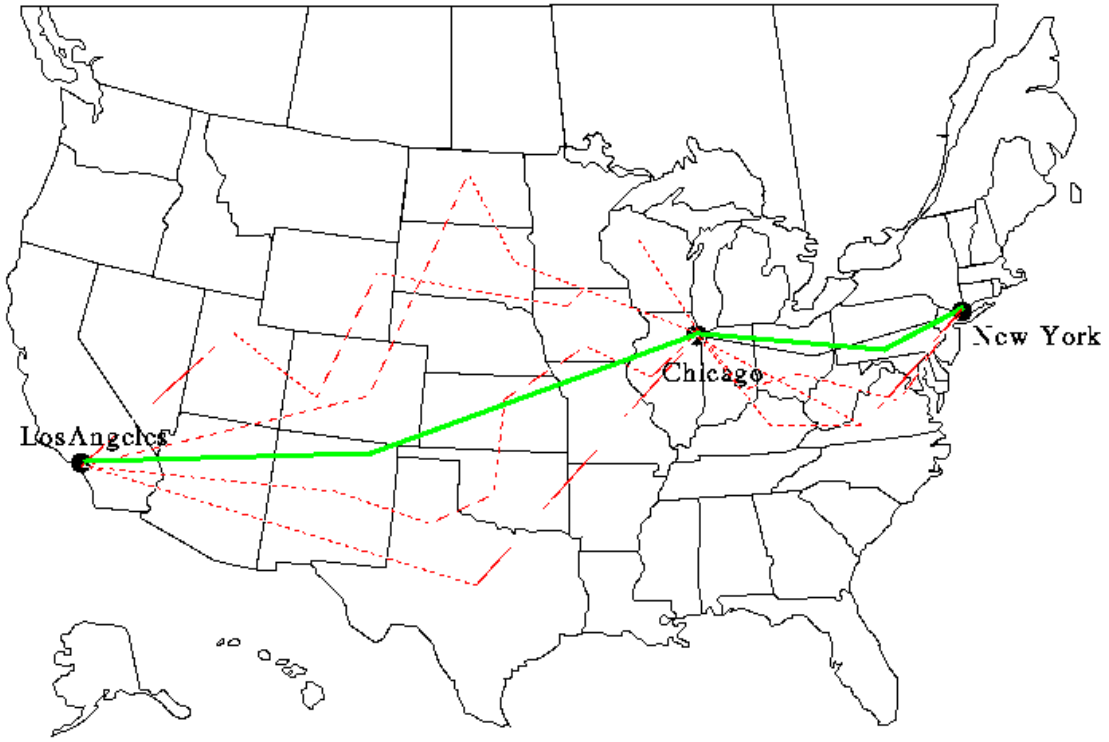
Bellman expectation equation (Prediction): for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches (Dynamic Programming) for 'large' MDPs
Bellman optimality equation (Control): for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

Dynamic Programming

Dynamic Programming (DP)

A class of methods that solves complex problems by breaking them down into subproblems (divide and conquer):

1. We first divide into subproblems (the path from NY to LA must pass from Chicago)
2. We solve the easier subproblems (find best path from NY to Chicago and the best from Chicago to LA)
3. We combine the solutions to the subproblems for solving the original one



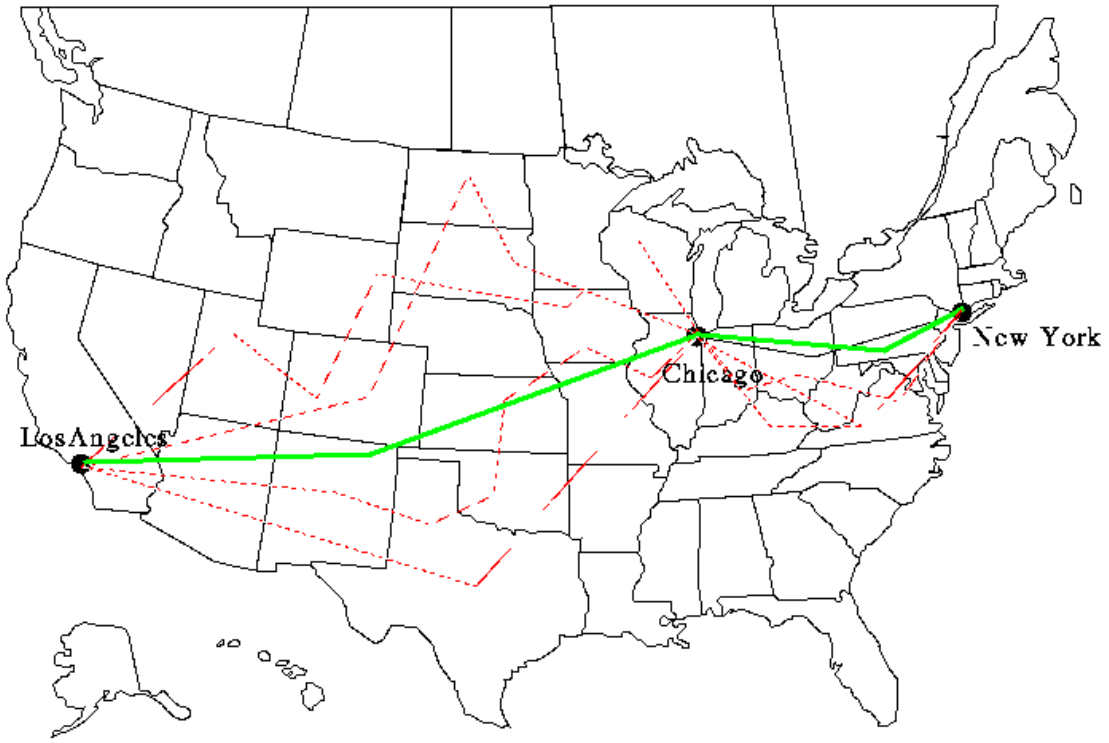
Raghavendra Singh https://sipi.usc.edu/~ortega/RD_Examples/boxDP.html

Dynamic Programming

Dynamic Programming (DP)

DP is a tool that allows to solve problems with the following properties (also outside of RL):

1. Problems that have a substructure, an optimal solutions can be decomposed into optimal subproblems (principle of optimality applies!)
2. Problems that have overlapping subproblems: the subproblems recur many times, so solutions of subproblems can be reused many times (think about backup diagrams)



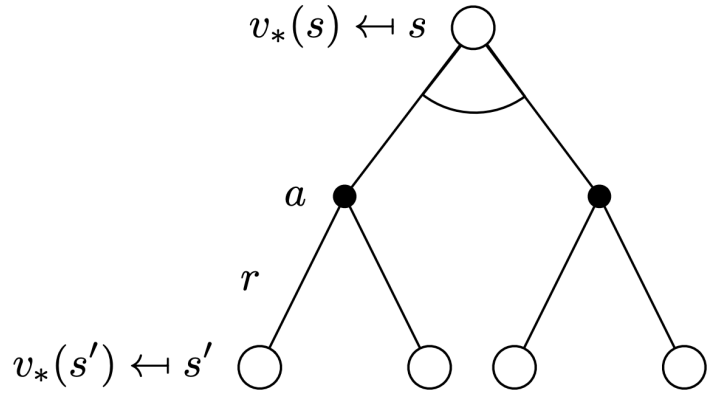
Raghavendra Singh https://sipi.usc.edu/~ortega/RD_Examples/boxDP.html

Dynamic Programming for MPDs

MDPs have recursive relationships that makes the application of DP particularly effective: the Bellman equations!

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

Bellman expectation equation (Prediction): for finding value functions and action-value functions	Linear: we can use it for small MDPs. We need to resort to iterative approaches for 'large' MDPs
Bellman optimality equation (Control): for finding optimal value functions and optimal action-value functions	Non-linear: we need iterative approaches even for small MDPs.

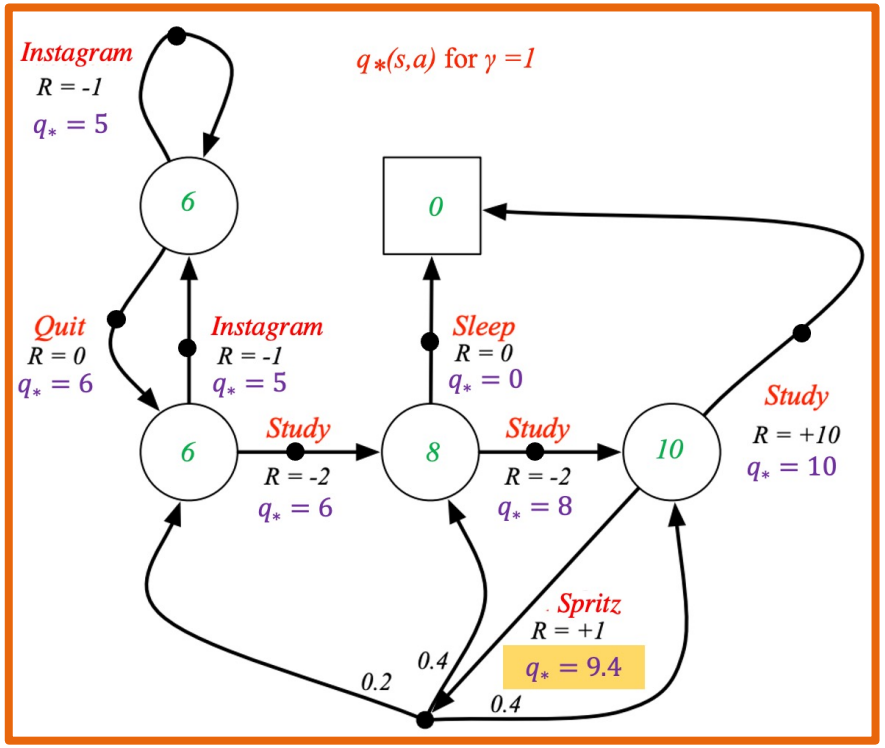


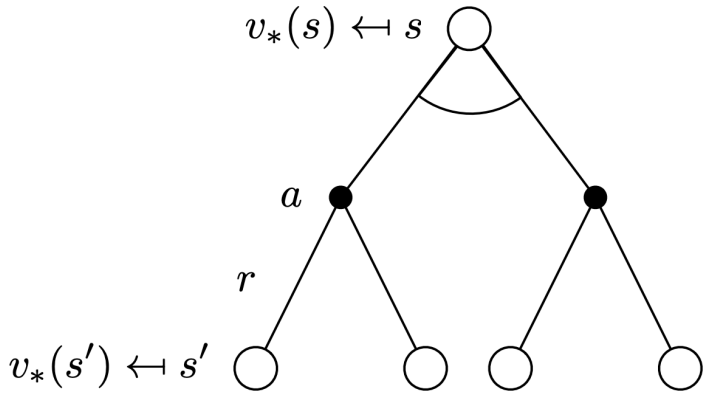
$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Dynamic Programming for MPDs

MDPs have recursive relationships that makes the application of DP particularly effective: the Bellman Equation!

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





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

We both have substructure and overlapping problems!

Dynamic Programming for MPDs

- Pay attention that we assume that the MDP is known! This is a strong assumption: in the 'full' RL problem this hypothesis will be relaxed

- When the MDP is known, people referred to this problem as the 'planning' problem

More formally, our task will be, given the full knowledge of the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

1. Prediction: find the value function v_π (optionally q_π)
2. Control: find π^* (optionally v_π / q_π)

Prediction: Policy Evaluation

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \boxed{\pi(a|s)} \sum_{s',r} \boxed{p(s', r \mid s, a)} \Big[r + \gamma v_\pi(s') \Big]
\end{aligned}
$$

AGENT

ENVIRONMENT

Prediction: Policy Evaluation

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big]
\end{aligned}
$$

AGENT ENVIRONMENT

Instead of solving the Bellman equation directly, we use it as an update rule (for all s in $\mathcal{S}$)

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

Prediction: Policy Evaluation

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]
\end{aligned}
$$

AGENT ENVIRONMENT

The dynamic programming procedure is associated with initial guesses of v that are iteratively improved thanks to a set of linear equations (overlapping subproblems) that are computed over and over

Instead of solving the Bellman equation directly, we use it as an update rule (for all s in S)

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

Prediction: Policy Evaluation

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]
\end{aligned}
$$

AGENT ENVIRONMENT

$v_k = v_\pi$ is a fixed point (the Bellman equation holds for v_π)

Iterative policy evaluation: the sequence $\{v_k\}$ converges to v_π for $k \to \infty$

Instead of solving the Bellman equation directly, we use it as an update rule (for all s in S)

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

Prediction: In-place vs Synchronous Policy Evaluation

Synchronous policy evaluation stores two copies of value function, for all states:

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right)$$

$$v_{old} \leftarrow v_{new}$$

In-place policy evaluation only stores one copy of value function:

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right)$$

Prediction: In-place vs Synchronous Policy Evaluation

Synchronous policy evaluation stores two copies of value function, for all states:

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right)$$

$$v_{old} \leftarrow v_{new}$$

In-place policy evaluation only stores one copy of value function:

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right)$$

While synchronous update seems more rigorous, in-place update achieves faster convergence since we use newer estimations as soon as they are available!

We will typically use in-place updates in practice!

Prediction: Policy Evaluation

‘In place’ update implementation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Updates are done in one ‘sweep’ trough the state space

Prediction: Policy Evaluation

'In place' update implementation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Termination criteria

Prediction: Policy Evaluation

'In place' update implementation

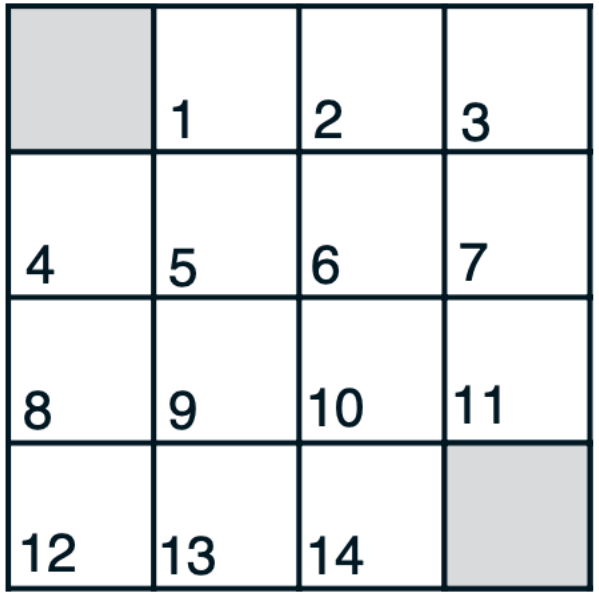
Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Inizialization (if you have domain knowledge, reasonable choices may allow you to converge faster)

Prediction: Policy Evaluation
A small gridworld example

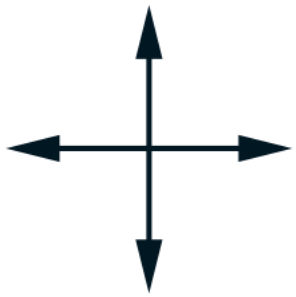


Nonterminal states:

$$\mathcal{S} = \{1, 2, \ldots, 14\}$$

Terminal states: grey boxes

Prediction: Policy Evaluation
A small gridworld example



actions

$\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{right}, \texttt{left}\}$

Deterministic transictions for each action

If an action takes the agent off the grid, the state remain unchanged

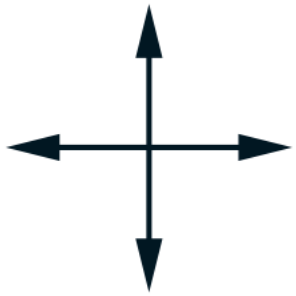
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

Nonterminal states:

$$\mathcal{S} = \{1, 2, \ldots, 14\}$$

Terminal states: grey boxes

Prediction: Policy Evaluation
A small gridworld example

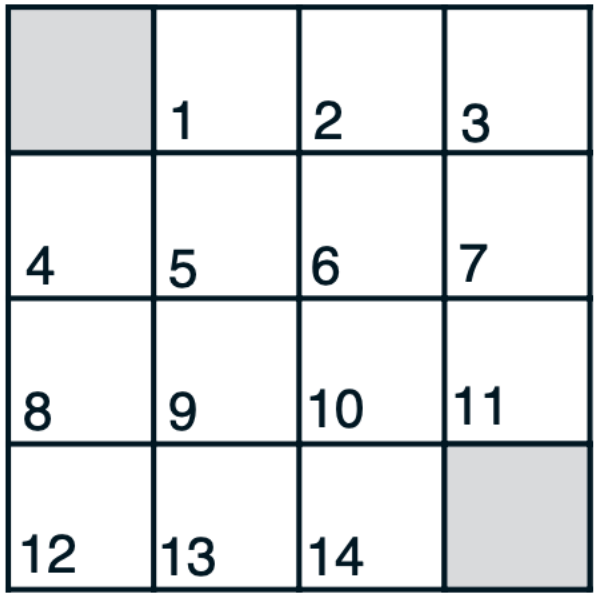


actions

$\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{right}, \texttt{left}\}$

Deterministic transictions for each action

If an action takes the agent off the grid, the state remain unchanged



Nonterminal states:

$$\mathcal{S} = \{1, 2, \ldots, 14\}$$

Terminal states: grey boxes

$$R_t = -1$$

on all transitions

In this set-up, the sooner a terminal state is reached, the better!

Let's evaluate a equiprobable random policy:

$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$

How many steps will it takes with random walk to reach a grey state?

Prediction: Policy Evaluation
A small gridworld example

$k = 0$

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

Initial estimation

Let's consider the synchronous case

Loop:
$\Delta \leftarrow 0$
Loop for each $s \in \mathcal{S}$:
$v \leftarrow V(s)$; $V_{OLD} = V$
$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V_{OLD}(s')\right]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Prediction: Policy Evaluation
A small gridworld example

$k = 0$

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

Let's consider the synchronous case

What is the estimation at k = 1?

Loop:
$\Delta \leftarrow 0$
Loop for each $s \in \mathcal{S}$:
$v \leftarrow V(s)$; $V_{OLD} = V$
$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V_{OLD}(s')\right]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Prediction: Policy Evaluation
A small gridworld example

$k = 0$

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

Let's consider the synchronous case

What is the estimation at k = 1?

0.25*(-1+0-1+0-1+0-1+0) = -1

$k = 1$

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

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$; $V_{OLD} = V$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_{OLD}(s') \right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Prediction: Policy Evaluation
A small gridworld example

Let's consider the synchronous case

What is the estimation at k = 1?

0.25*(-1+0+-1-1-1-1-1-1) = -1.75

$k = 0$

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

$k = 1$

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

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$; $V_{OLD} = V$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_{OLD}(s') \right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Prediction: Policy Evaluation
A small gridworld example

$k = 0$

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

$k = 1$

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

$k = 3$

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

Prediction: Policy Evaluation
A small gridworld example

$k = 0$

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

$k = 1$

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

$k = 3$

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

An interactive gridworld example by A. Karpathy

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

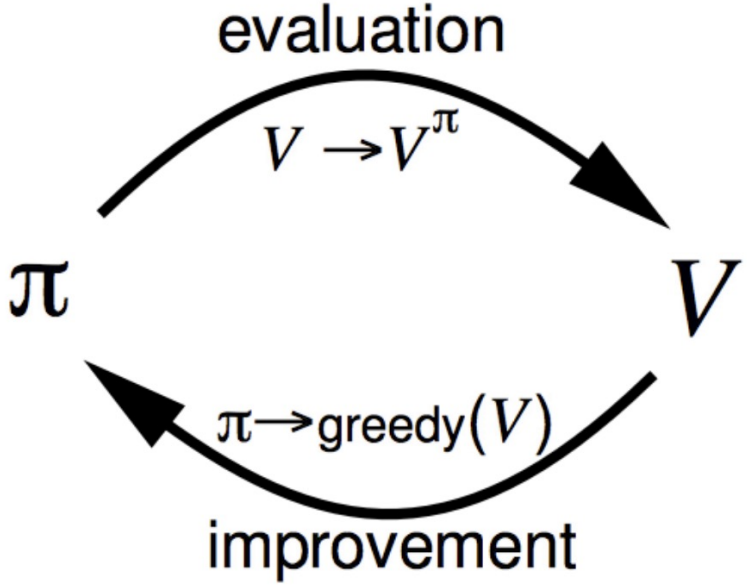
Control: Policy Improvement (Policy iteration)

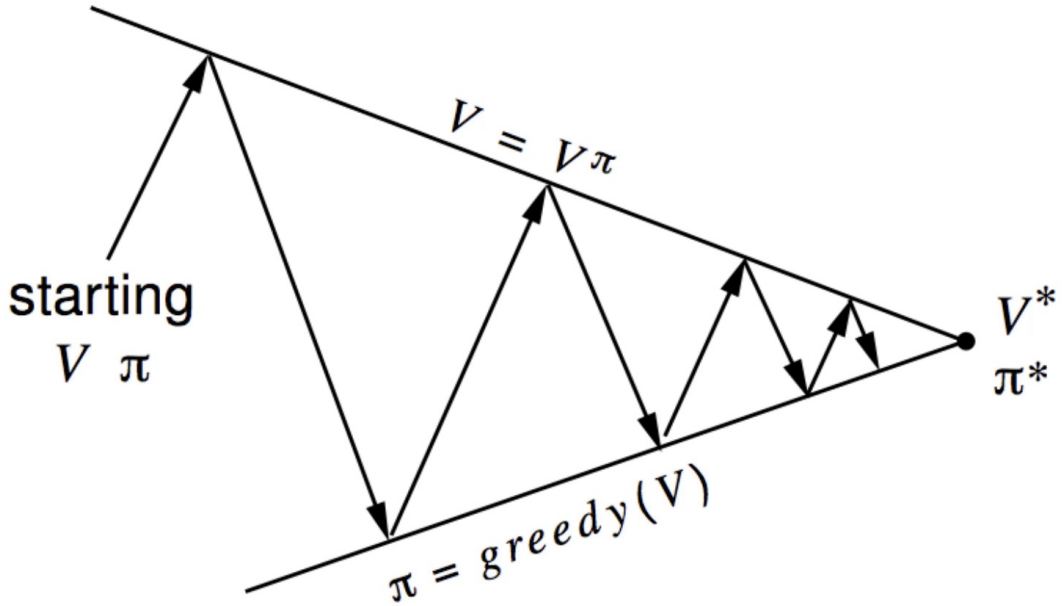
One way to improve a policy is the following:

- Evaluate a policy π
- Improve the policy by acting greedily w.r.t. v_π:

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

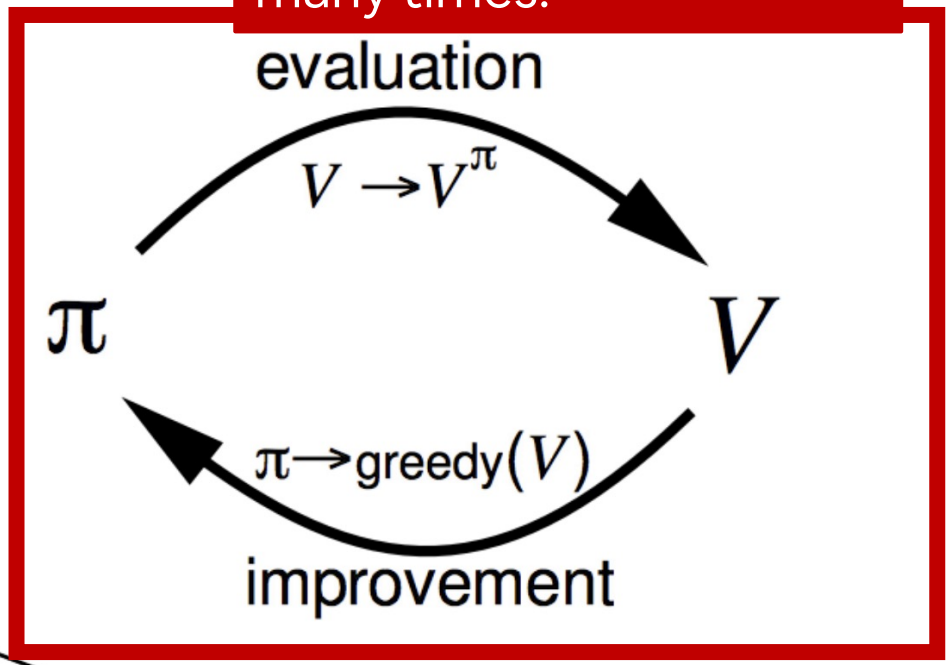
- We can evaluate $v_{\pi'}$ and then improve the policy again!
- This process of policy iteration will converge to the optimal policy π^*





Control: Policy Improvement (Policy iteration)

We'll see this approach many times!

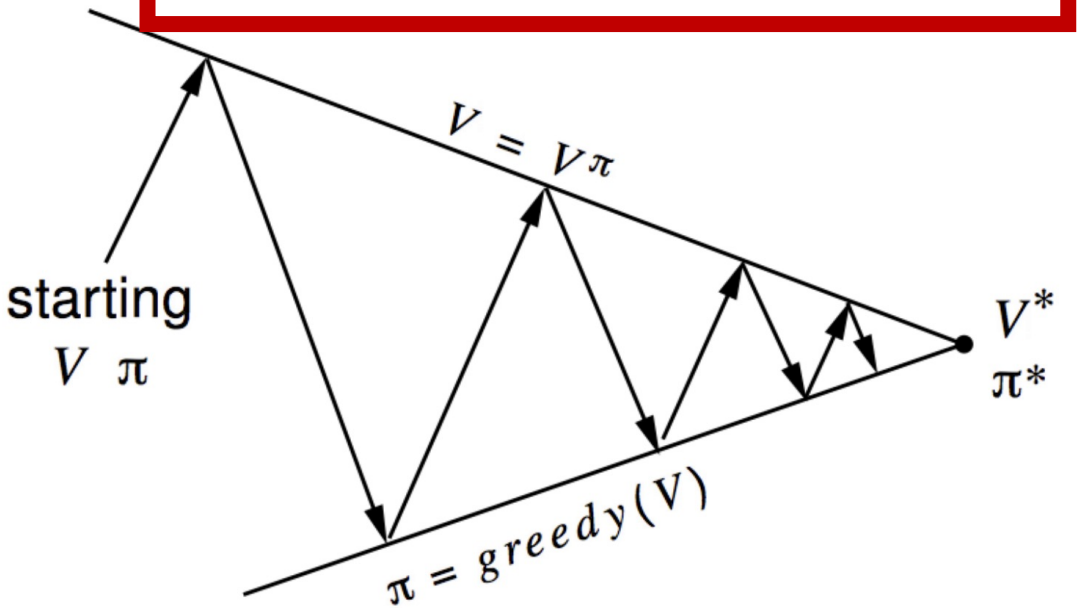


One way to improve a policy is the following:

- Evaluate a policy π
- Improve the policy by acting greedly w.r.t. v_{π}:

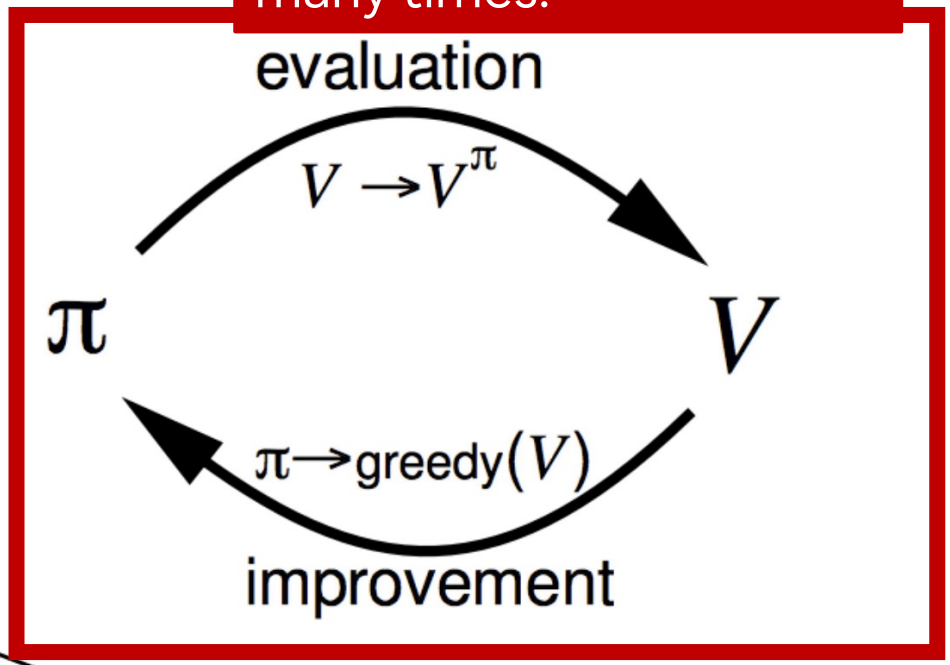
$$\pi' = \text{greedy}(v_{\pi})$$

- We can evaluate $v_{\pi'}$ and then improve the policy again!
- This process of policy iteration will converge to the optimal policy π^*



Control: Policy Improvement (Policy iteration)

We'll see this approach many times!



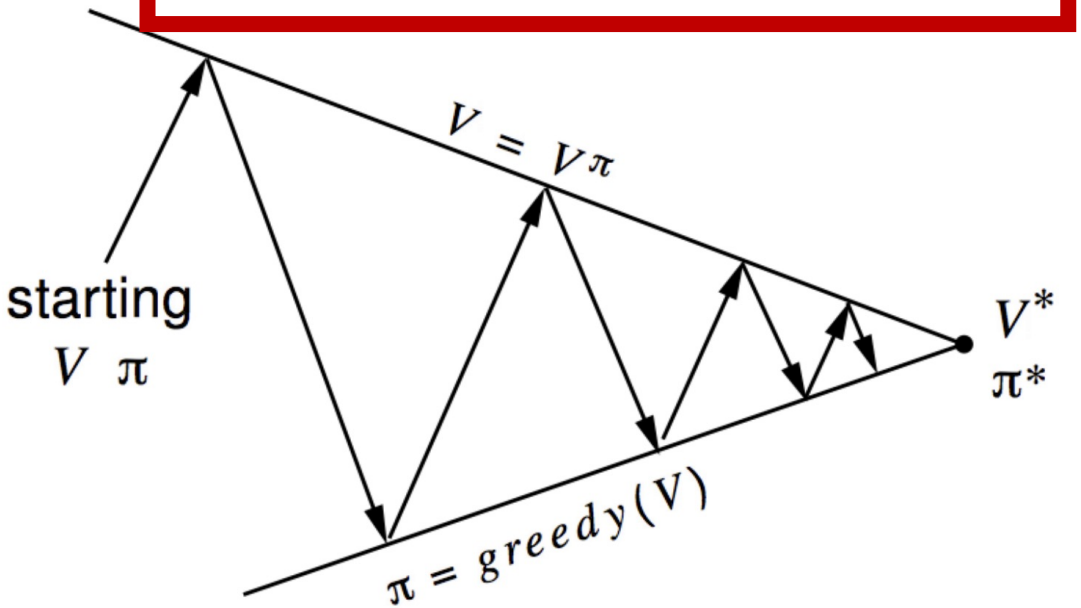
One way to improve a policy is the following:

- Evaluate
- Improve w.r.t. v_π:

With known MDP, this is equivalent:
$argmax_a q_\pi(s,a) = argmax_a \mathcal{R}_s^a + \gamma v_\pi(s')$

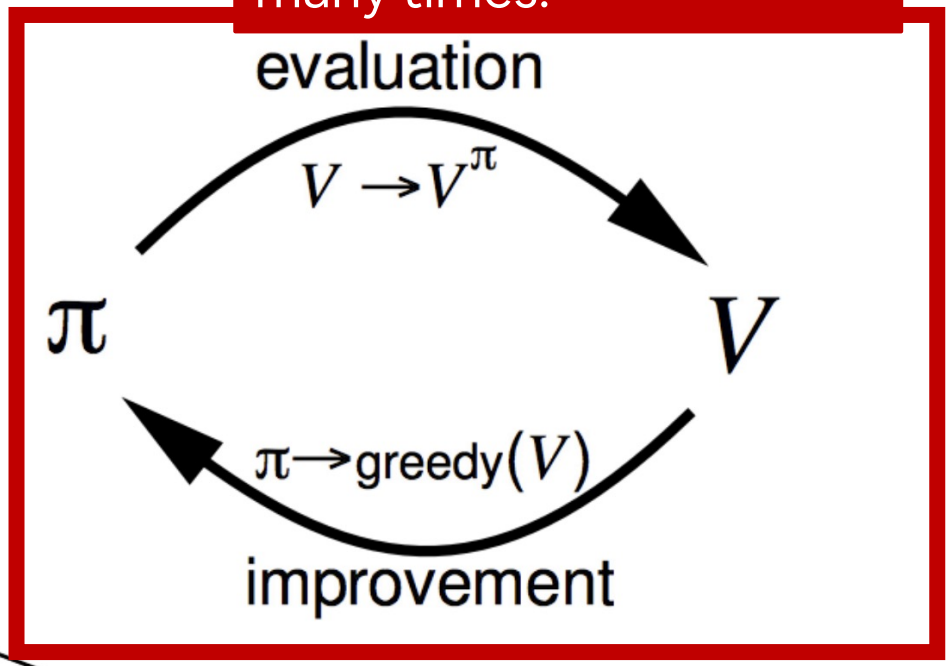
$$\pi' = \text{greedy}(v_\pi)$$

- We can evaluate $v_{\pi'}$ and then improve the policy again!
- This process of policy iteration will converge to the optimal policy π^*



Control: Policy Improvement (Policy iteration)

We'll see this approach many times!



One way to improve a policy is the following:

- Evaluate
- Improve w.r.t. v_π:

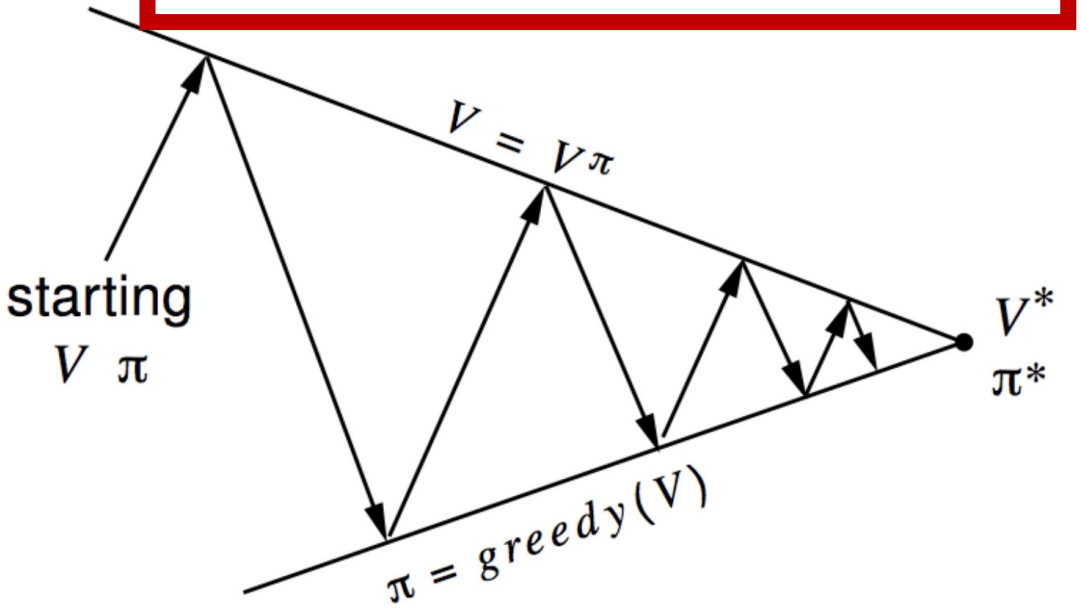
With known MDP, this is equivalent:
$argmax_a q_\pi(s,a) = argmax_a \mathcal{R}_s^a + \gamma v_\pi(s')$

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

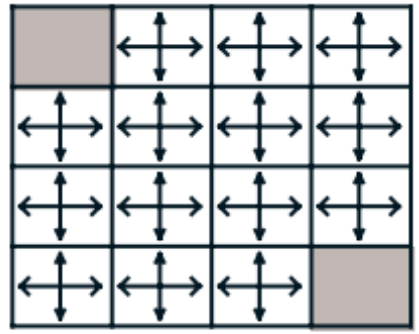
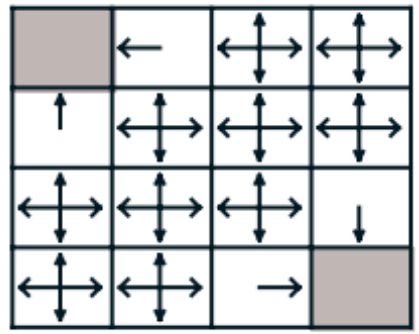
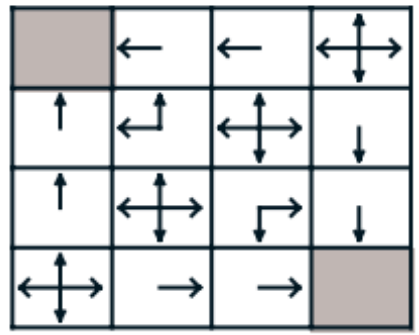
- We can evaluate v_π, and then improve the policy again!
- This process of policy iteration will converge to the optimal policy π^*
- In the following we'll show that:

1. acting greedily can improve the policy;
2. acting greedily will allow us to reach optimality

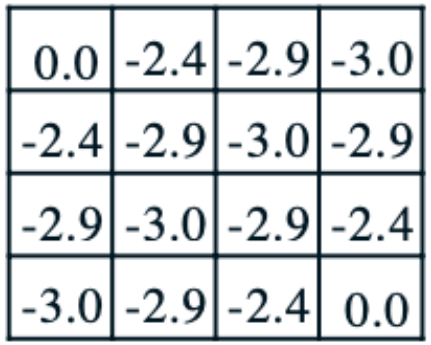
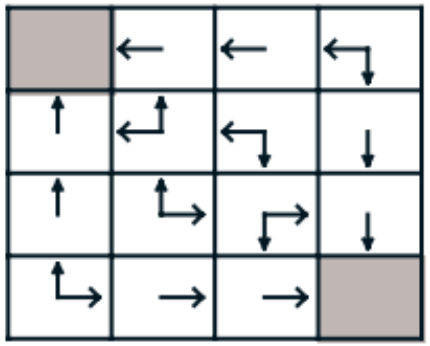
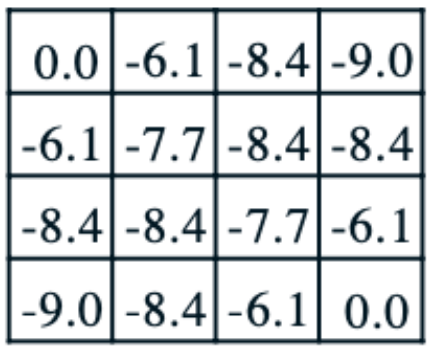
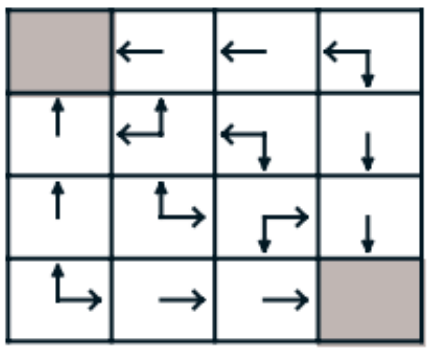
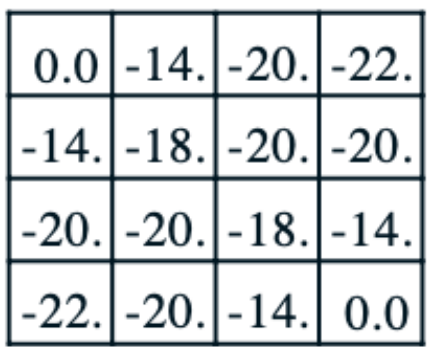
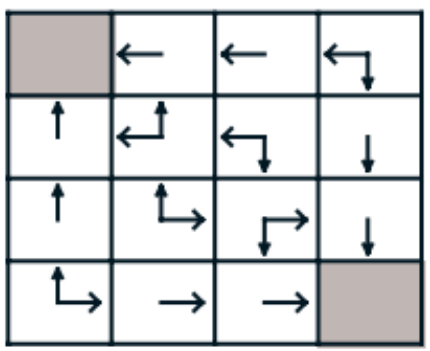
- But let's start with an example first!



Control: Policy iteration 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	

Control: Policy iteration in the small gridworld

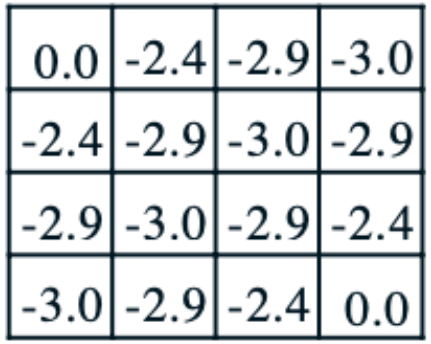
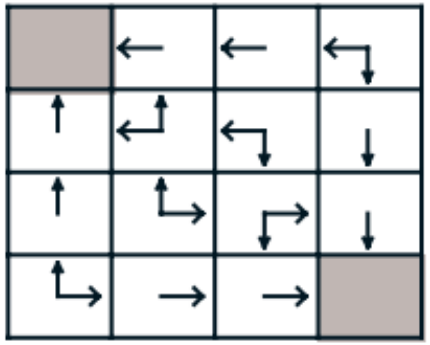
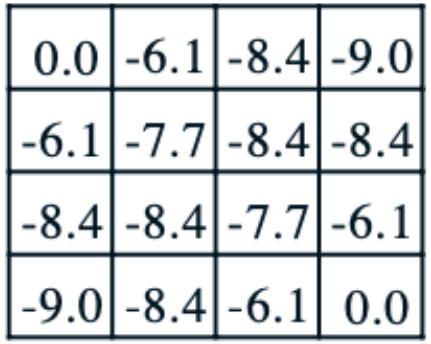
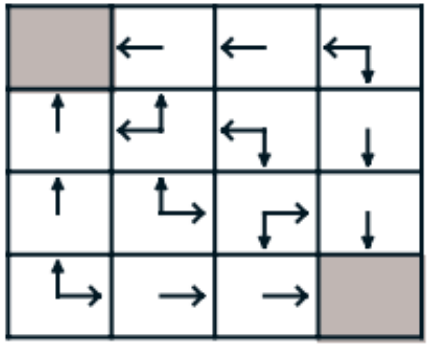
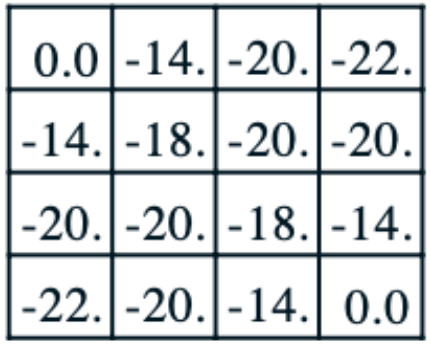
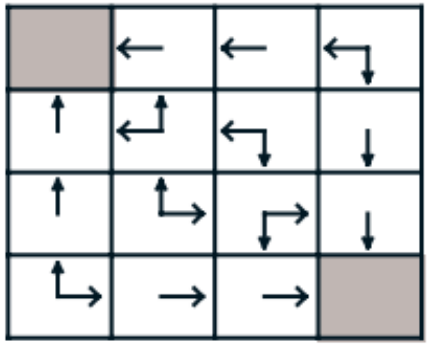
	v_k for the random policy	greedy policy w.r.t. v_k
$k = 3$		
$k = 10$		
$k = \infty$		

optimal policy

Control: Policy iteration in the small gridworld

This is a simple case: we know deterministically where the state s' when we apply action a when in state s

In other cases we'll have to consider all possible transitions with some probablities (SPOILER: in the full RL problem we will need to act on q instead on acting to v)

	v_k for the random policy	greedy policy w.r.t. v_k
$k = 3$		
$k = 10$		
$k = \infty$		

optimal policy

Control: Policy Improvement Theorem

- Let's consider a deterministic policy π
- We can improve the policy by acting greedly: $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_\pi(s, a)$

guarda anche da pdf 6 con i commenti aggiunti a lezione

(that is, we pick the one action that provides the best value and then we follow π - π' is the greedy policy)

Control: Policy Improvement Theorem

- Let's consider a deterministic policy π
- We can improve the policy by acting greedly: $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_\pi(s, a)$

(that is, we pick the one action that provides the best value and then we follow π - π' is the greedy policy)

- This improves for one step: $q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$

- A policy $\pi' \geq \pi$ if $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \subset S$
- The inequality holds because I'm maxing out

guarda anche da pdf 6 con i commenti aggiunti a lezione

Control: Policy Improvement Theorem

- Let's consider a deterministic policy π
- We can improve the policy by acting greedily: $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, q_\pi(s, a)$

guarda anche da pdf 6 con i commenti aggiunti a lezione

(that is, we pick the one action that provides the best value and then we follow π - π' is the greedy policy)

- This improves for one step: $q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
- And improves the whole value function

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

Control: Policy Improvement Theorem

- Let's consider a deterministic policy π
- We can improve the policy by acting greedly: $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_\pi(s, a)$

guarda anche da pdf 6 con i commenti aggiunti a lezione

(that is, we pick the one action that provides the best value and then we follow π - π' is the greedy policy)

- This improves for one step: $q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
- And improves the whole value function

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

After the next step, everything is equal (we are following π after the first step)

Control: Policy Improvement Theorem

- Let's consider a deterministic policy π
- We can improve the policy by acting greedily: $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, q_\pi(s, a)$

guarda anche da pdf 6 con i commenti aggiunti a lezione

(that is, we pick the one action that provides the best value and then we follow π - π' is the greedy policy)

- This improves for one step: $q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
- And improves the whole value function

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

I'm applying this again...

Control: Policy Improvement Theorem

- Let's consider a deterministic policy π
- We can improve the policy by acting greedly: $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_\pi(s, a)$

guarda anche da pdf 6 con i commenti aggiunti a lezione

(that is, we pick the one action that provides the best value and then we follow π - π' is the greedy policy)

- This improves for one step: $q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
- And improves the whole value function

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

... and again!

$$\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right]$$

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

$$\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s)$$

Control: Policy Improvement Theorem

- Let's consider a deterministic policy π
- We can improve the policy by acting greedly: $\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_\pi(s, a)$

guarda anche da pdf 6 con i commenti aggiunti a lezione

(that is, we pick the one action that provides the best value and then we follow π - π' is the greedy policy)

- This improves for one step: $q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
- And improves the whole value function

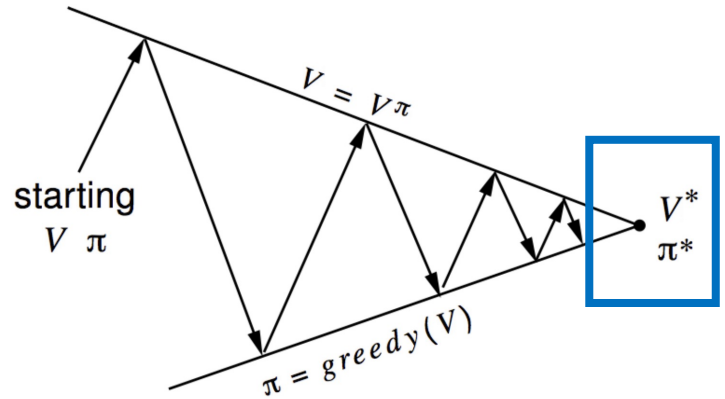
$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2},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

By acting greedly we are sure that we are not going to do worse... but do we reach optimality?

Control: Policy Improvement Theorem

- Let's consider the case in which improvement stops:

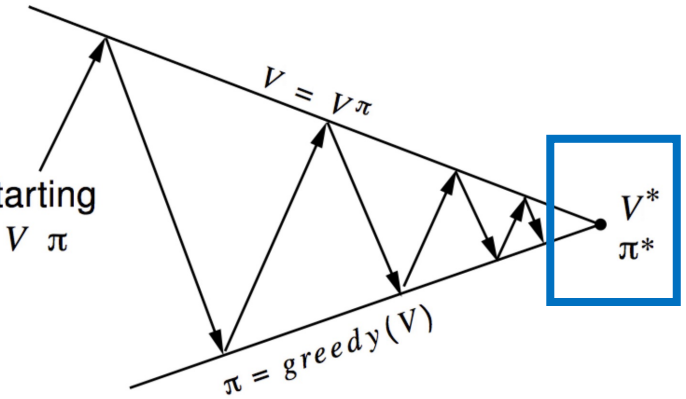
$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$



Control: Policy Improvement Theorem

- Let's consider the case in which improvement stops:

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



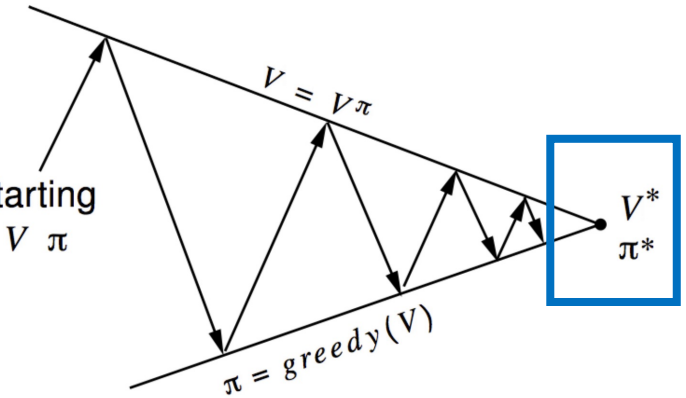
- But in this case, we have satisfied the Bellman optimality equation

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

Control: Policy Improvement Theorem

- Let's consider the case in which improvement stops:

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



- But in this case, we have satisfied the Bellman optimality equation

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

- Therefore, we reached the optimal policy (π is optimal):

$$v_\pi(s) = v_*(s) \text{ for all } s \in \mathcal{S}$$

- Policy iteration actually solves MDPs!

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

As before...

   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\left[r + \gamma V(s')\right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\left[r + \gamma V(s')\right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

The greedy action! The rest are just termination conditions

Pay attention to notation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Policy Iteration (using iterative policy evaluation) for estimat

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
     $\Delta \leftarrow 0$
     Loop for each $s \in \mathcal{S}$:
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
     old-action $\leftarrow \pi(s)$
     $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
     If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Pay attention to notation

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$$

=

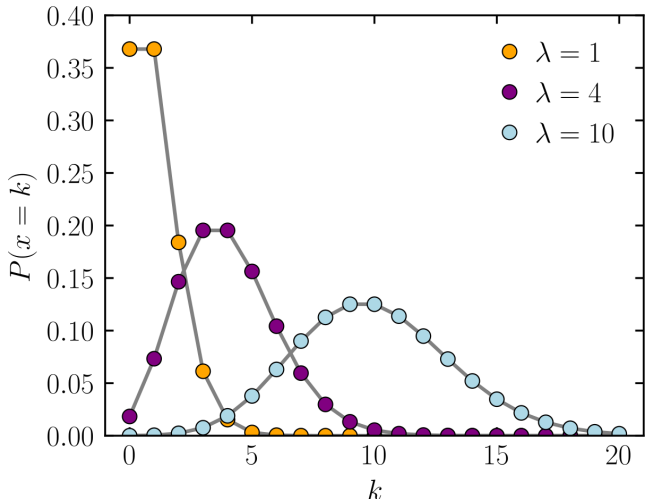
$$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\left[r + \gamma V(s')\right]$$

Control: Example – Jack's Car Rental

- Jack's manage 2 locations for a nationwide car rental company
- If a customer arrives at one location and if he has cars available, he rents the car for $10
- Jack can move the cars overnight for $2 per car moved

Control: Example – Jack's Car Rental

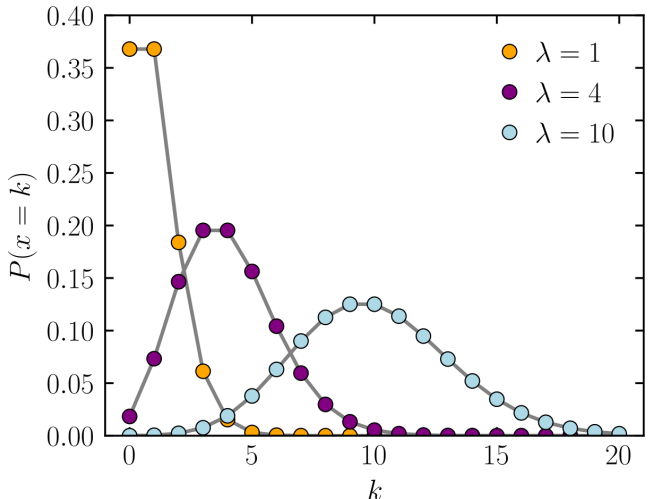
- Jack's manage 2 locations for a nationwide car rental company
- If a customer arrives at one location and if he has cars available, he rents the car for $10
- Jack can move the cars overnight for $2 per car moved
- The number n of cars requested and returned $\frac{\lambda^n}{n!}e^{-\lambda}$ are Poisson random variables
- 1° location: $\lambda = 3$ rental requests, $\lambda = 3$ returns
- 2° location: $\lambda = 4$ rental requests, $\lambda = 2$ returns
- No more than 20 cars can be at each location (additional cars are returned to the nationwide company) and no more than 5 cars can be moved overnight



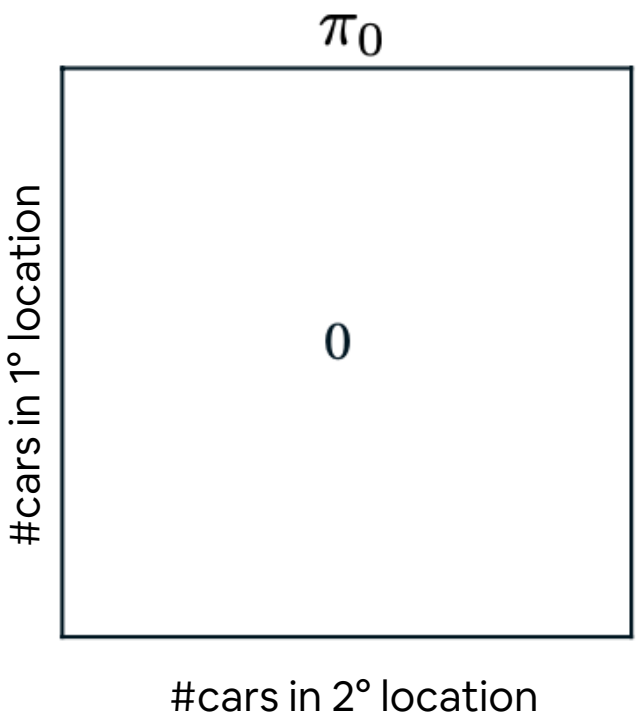
Control: Example – Jack's Car Rental

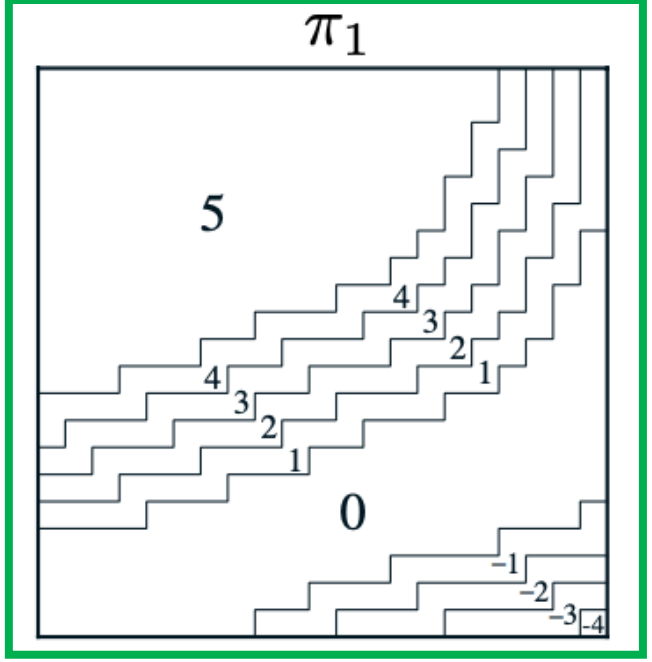
- We consider a discount rate $\gamma = 0.9$
- Time steps are days
- Actions are the net numbers of cars moved

- ions for a nationwide car
- t one location and if he has s the car for $10
- Jack can move the cars overnight for $2 per car moved
- The number n of cars requested and returned $\frac{\lambda^n}{n!}e^{-\lambda}$ are Poisson random variables
- 1° location: $\lambda = 3$ rental requests, $\lambda = 3$ returns
- 2° location: $\lambda = 4$ rental requests, $\lambda = 2$ returns
- No more than 20 cars can be at each location (additional cars are returned to the nationwide company) and no more than 5 cars can be moved overnight

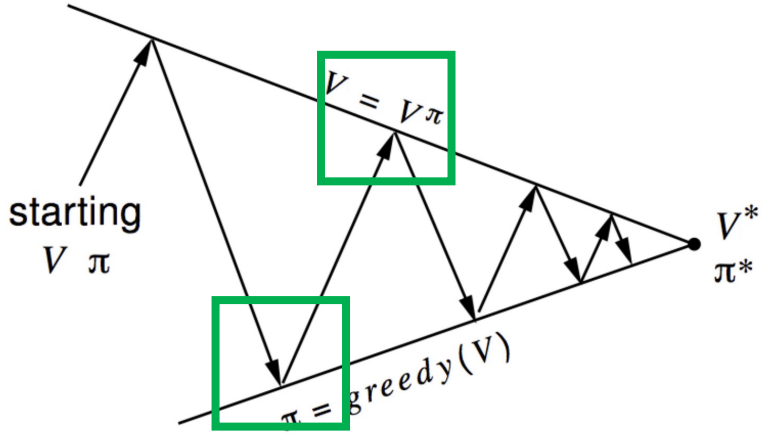


We start with the policy π_0 that moves zero cars. We then apply policy iteration

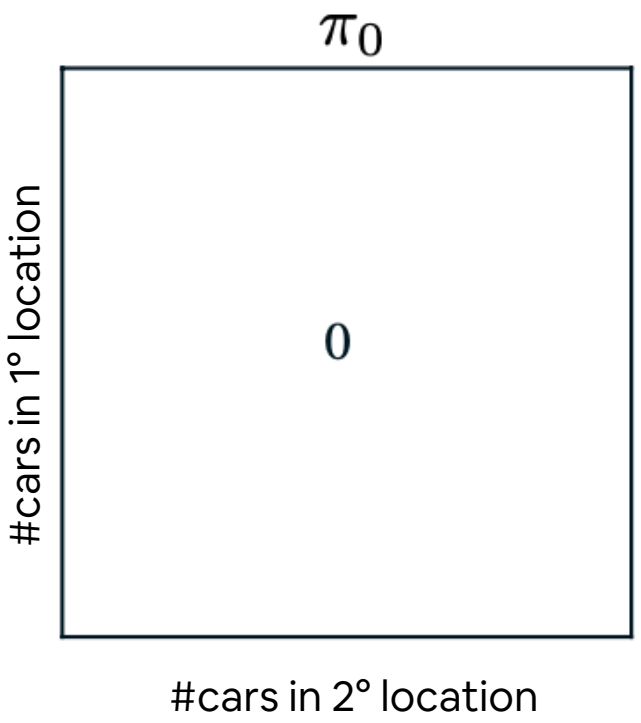


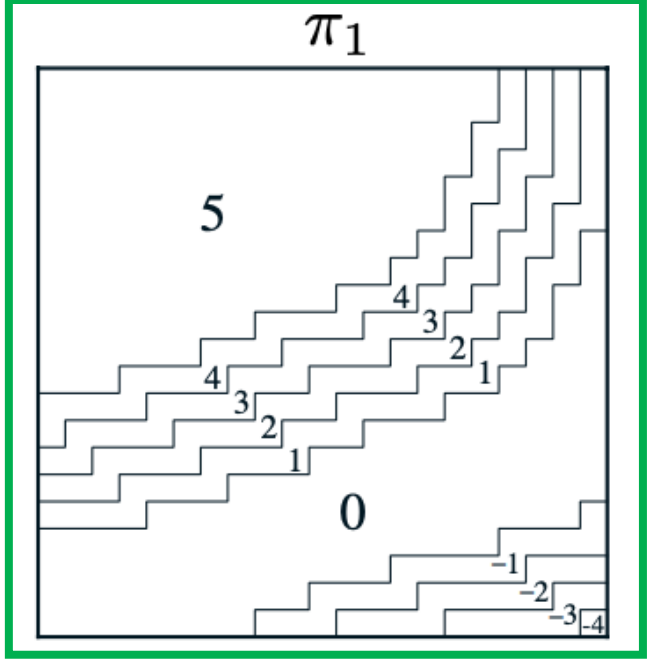


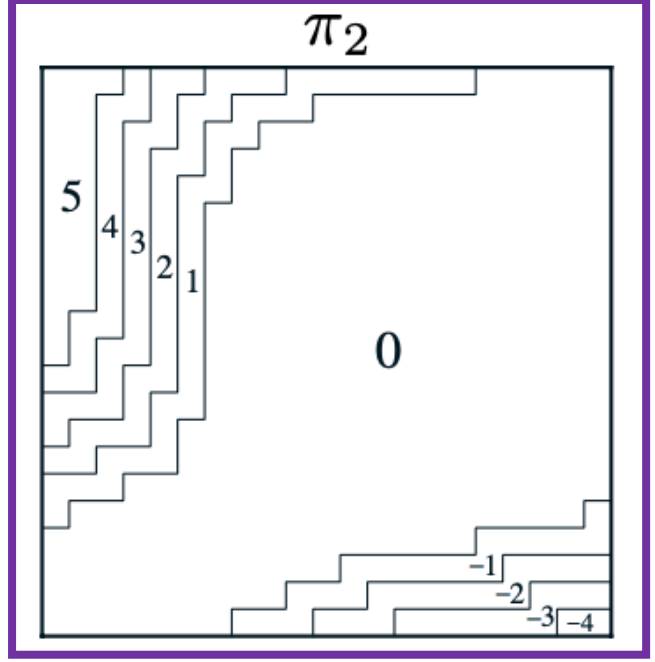
- 1° location:
$\lambda = 3$ rental requests,
$\lambda = 3$ returns
- 2° location:
$\lambda = 4$ rental requests,
$\lambda = 2$ returns

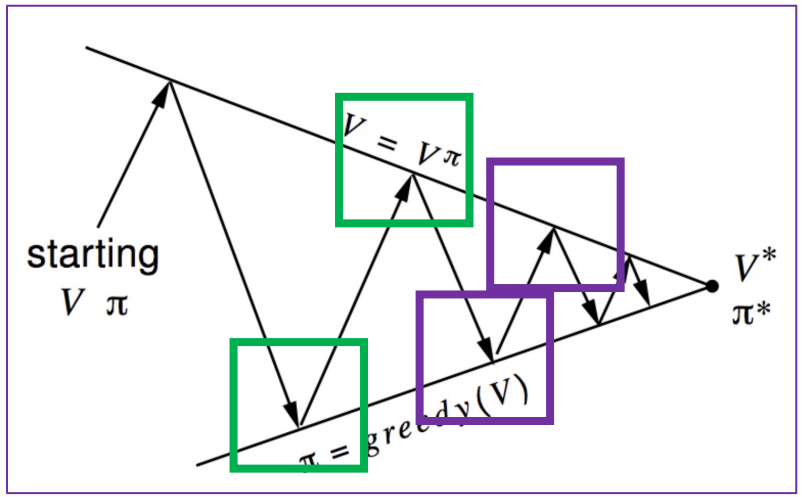


We start with the policy π_0 that moves zero cars. We then apply policy iteration

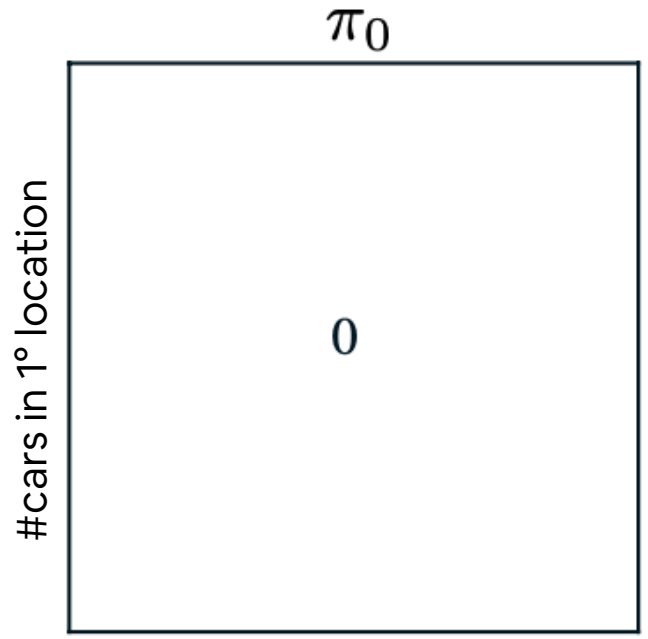


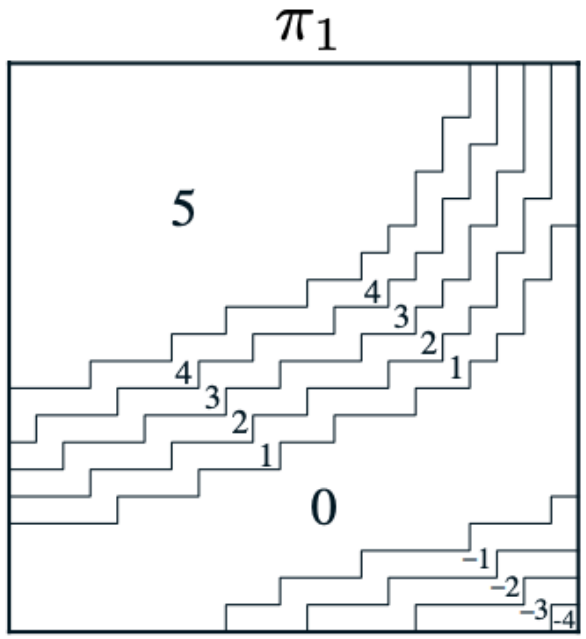


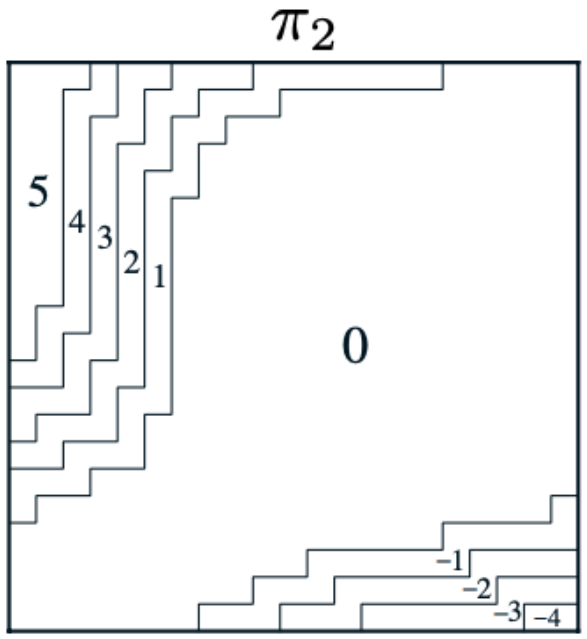




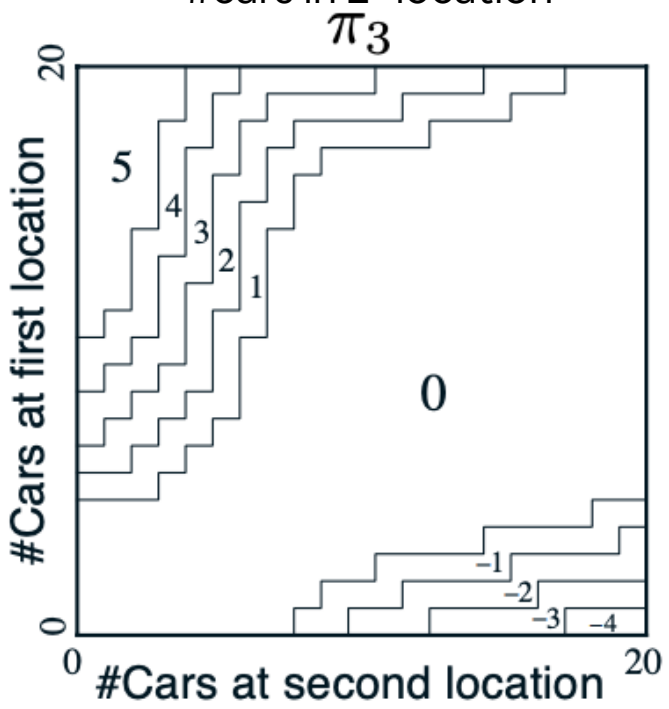
We start with the policy π_0 that moves zero cars. We then apply policy iteration



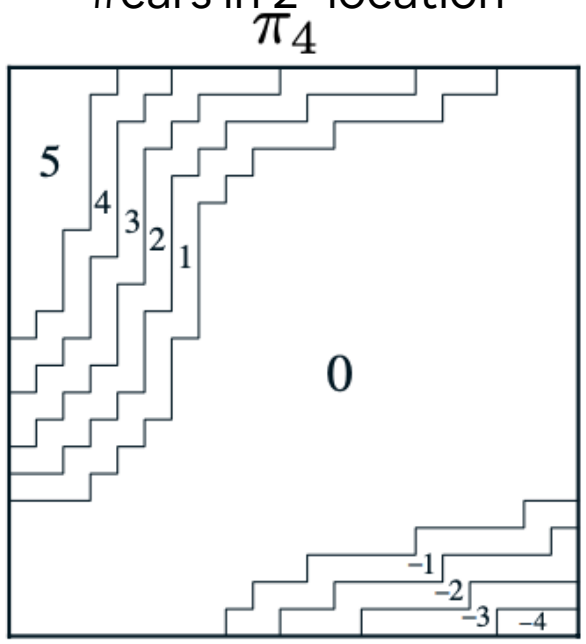


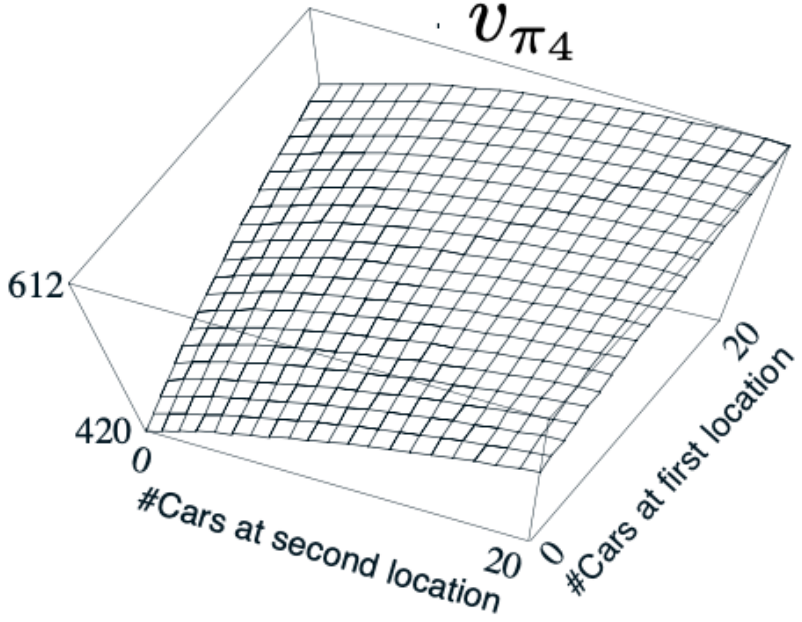


#cars in 2° location



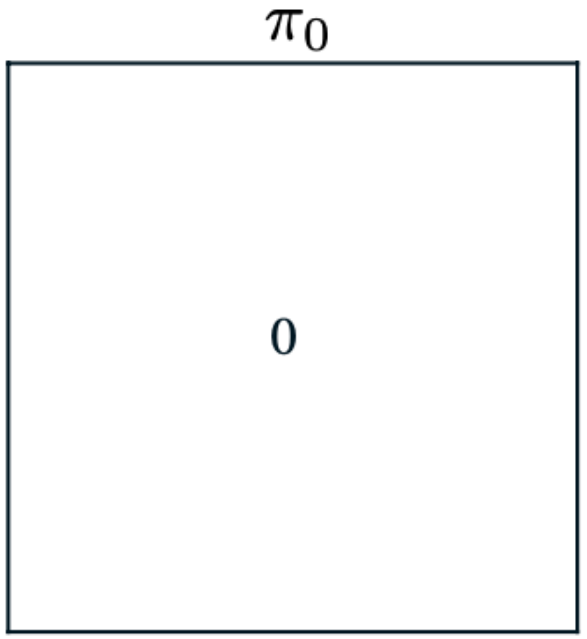
#cars in 2° location

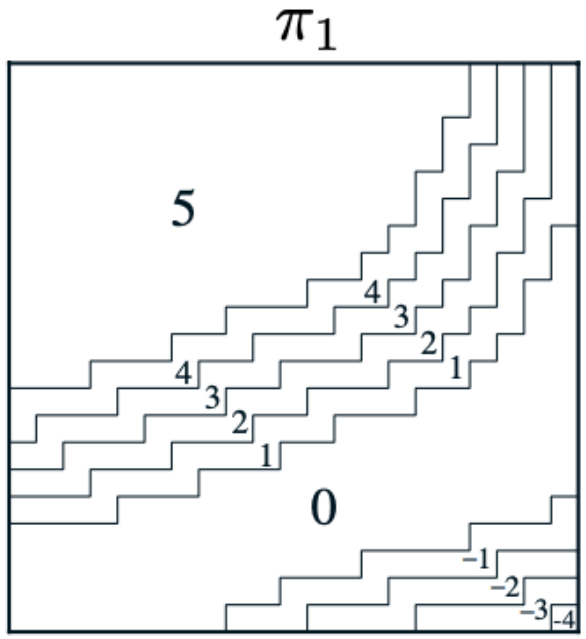


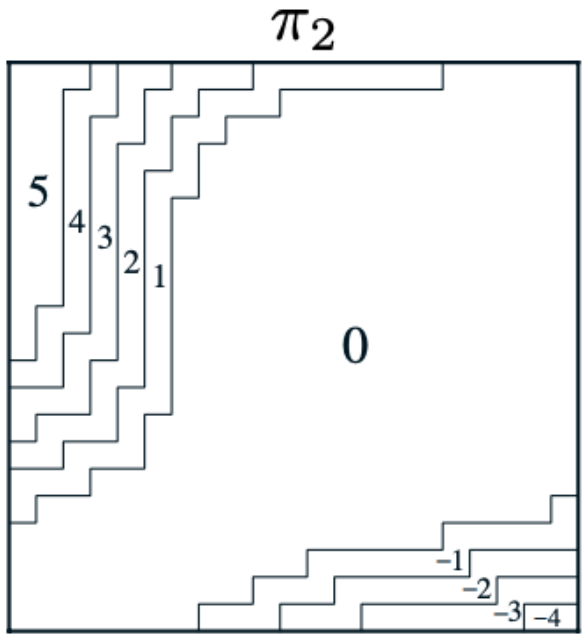


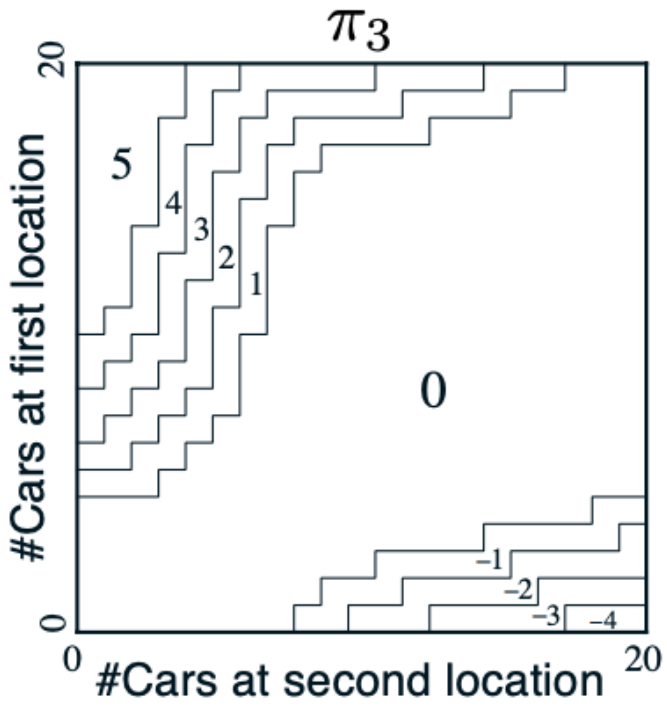
We start with the policy π_0 that moves zero cars. We then apply policy iteration

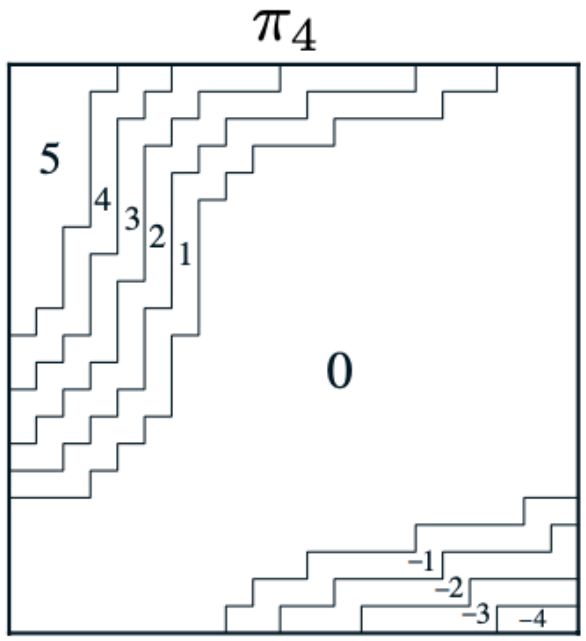
More on this on lab #3

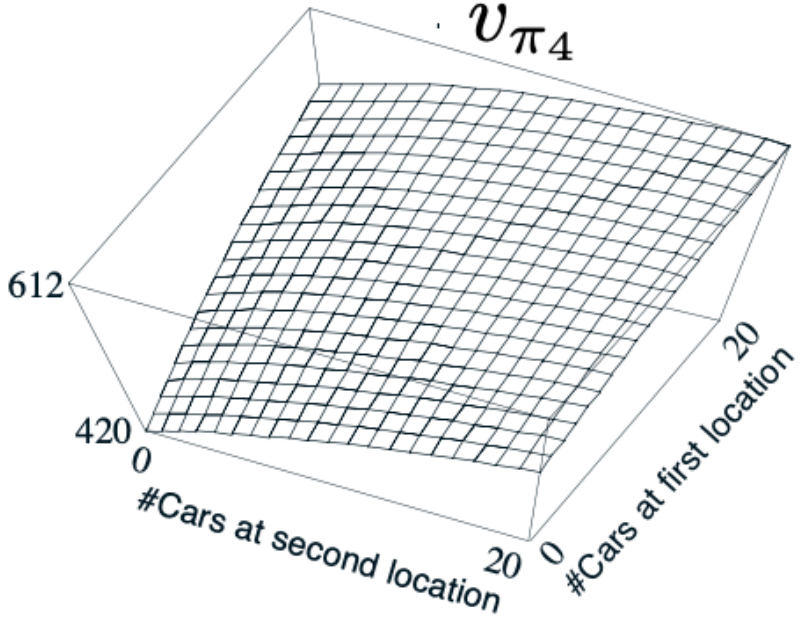










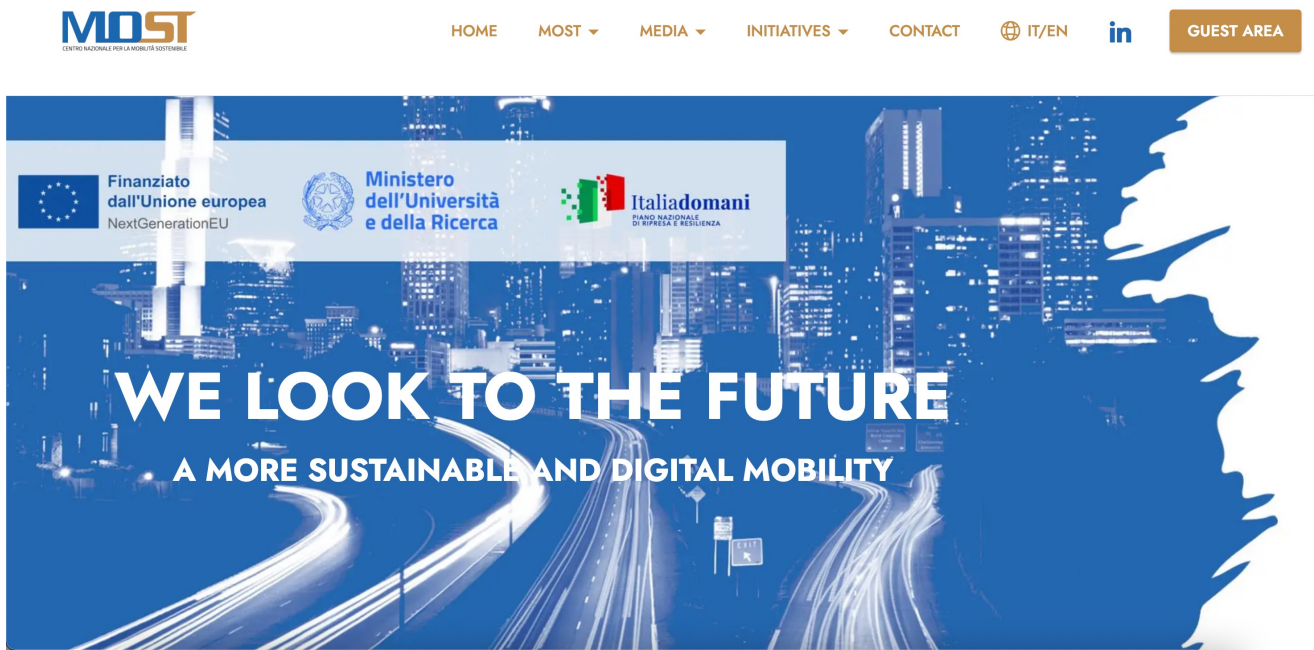


Control: Example – Jack's Car Rental

- We are dealing with similar problems for smart mobility tasks
- Optimization and fairness in micro-mobility services (ex. bike sharing)

https://arxiv.org/abs/2403.15780

https://www.centronazionalemost.it/eg/



Credits

- Image of the course is taken from C. Mahoney ‘Reinforcement Learning’ https://towardsdatascience.com/reinforcement-learning-fda8ff535bb6

AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning 2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Thank you! Questions?

Lecture #05
Bellman Equations & Dynamic Programming for MDPs

Gian Antonio Susto