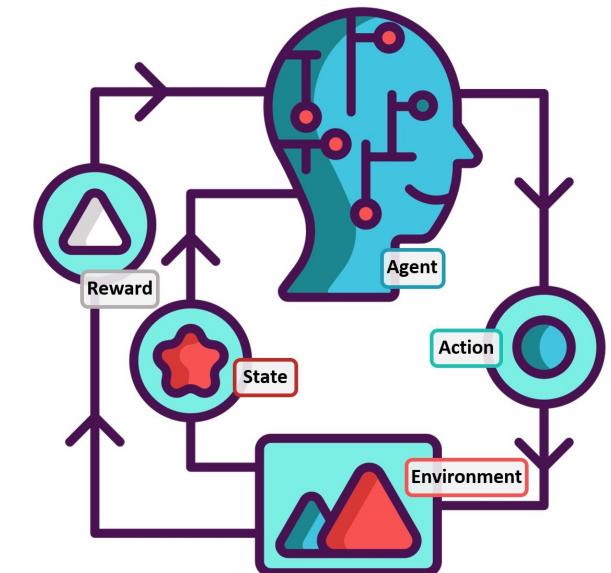


Lecture #06

Dynamic Programming for MDPs & Monte Carlo

Gian Antonio Susto



Recap

In MDPs we have two problems (two goals to ‘solve the MDP’)

1. Prediction/evalution



2. Control



Recap

In MDPs we have two problems (two goals to ‘solve the MDP’)

1. Prediction/evalution (of a Policy)

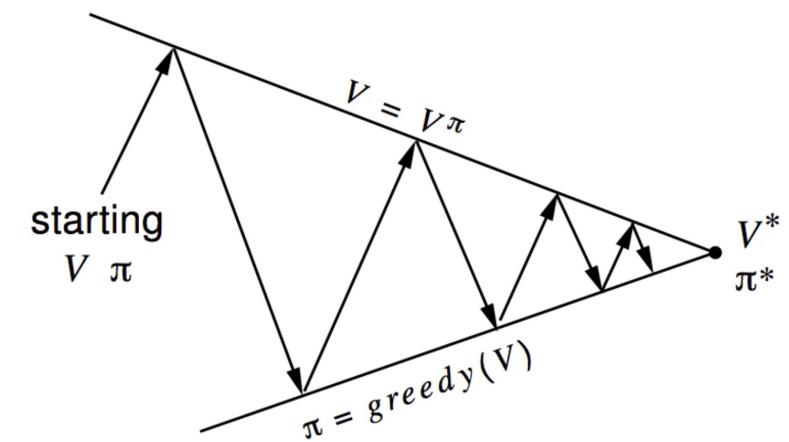
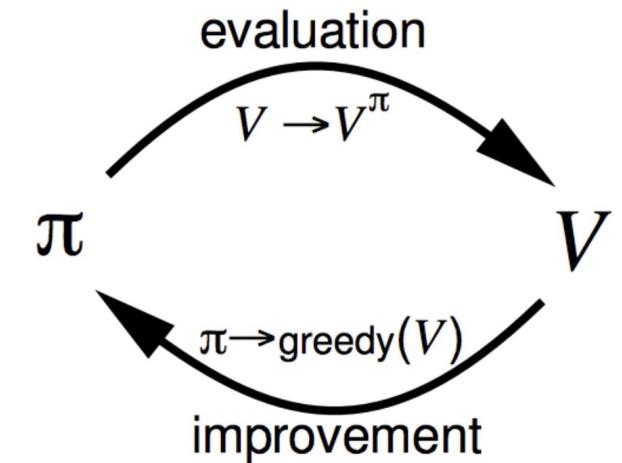
Understand how good a policy π is, ie. typically deriving the value function $v_\pi(s)$ and q_π

2. Control (Policy improvement)

Improve the current policy π /find the optimal policy typically deriving the value function $v^*(s)$ and the action-value function $q^*(s, a)$

Recap

Problem	Bellman Equation	Algorithm
Prediction	Bellman expectation equation	Iterative Policy Evaluation
Control	Bellman expectation equation + Greedy Policy Improvement	Policy Iteration



Control: Policy Iteration (Recap)

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 $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

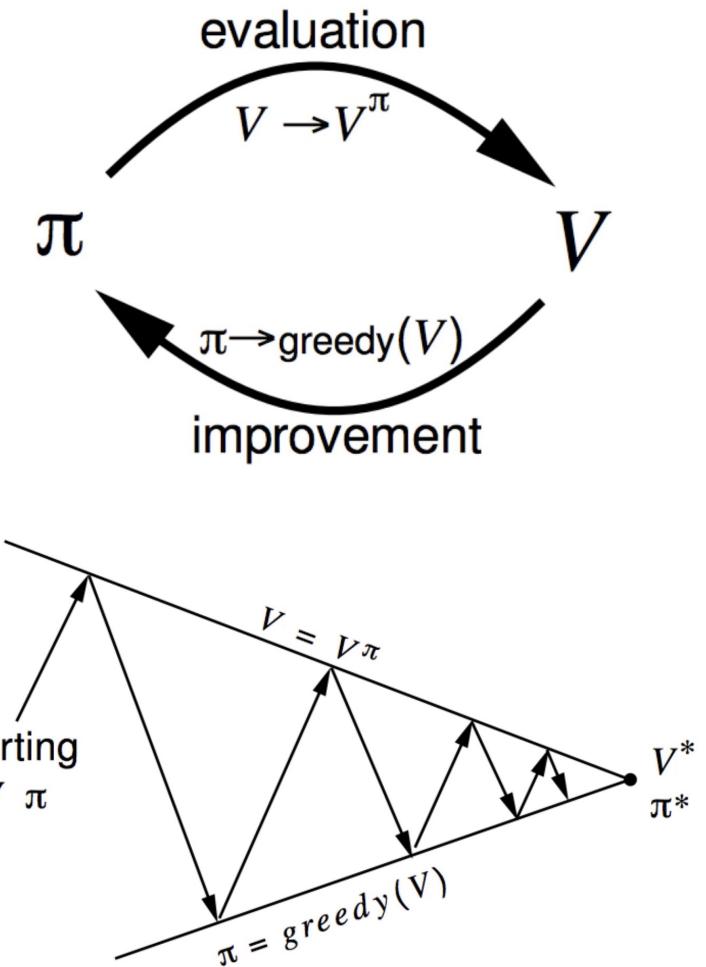
For each $s \in \mathcal{S}$:

$$\text{old-action} \leftarrow \pi(s)$$

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

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2



Control: Policy Improvement Theorem

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

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

Control: Policy Improvement Theorem

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

(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 $s \in S$
- The inequality holds because I'm maxing out

Control: Policy Improvement Theorem

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

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

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

Control: Policy Improvement Theorem

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

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

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

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

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

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

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

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

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

I'm applying this again...

Control: Policy Improvement Theorem

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

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

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

... and again!

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

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

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

Control: Policy Improvement Theorem

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

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

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

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

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

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

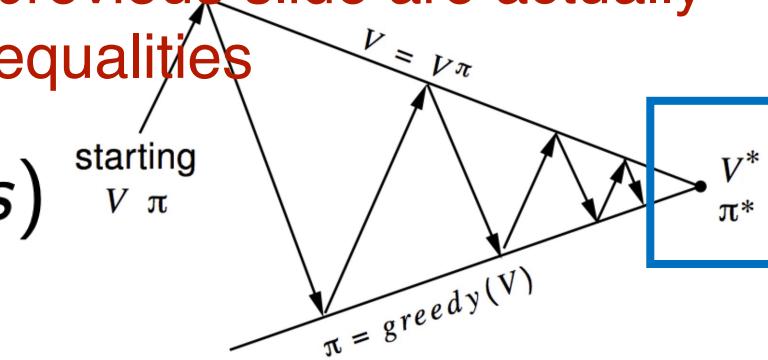
By acting greedily we are sure that we are not going to do worse... but do we reach optimality?

Control: Policy Improvement Theorem

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

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

case where inequalities in previous slide are actually equalities



Control: Policy Improvement Theorem

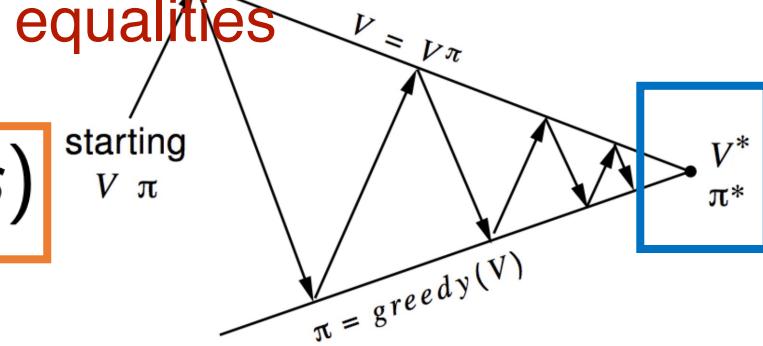
- Let's consider the case in which improvement stops:

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

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

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

case where inequalities in previous slide are actually equalities



Control: Policy Improvement Theorem

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

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

case where inequalities in previous slide are actually equalities

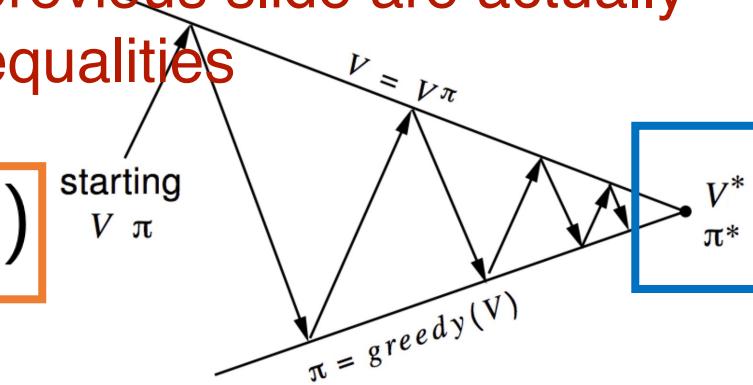
- But in this case, we have satisfied the Bellman optimality equation

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

- 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!



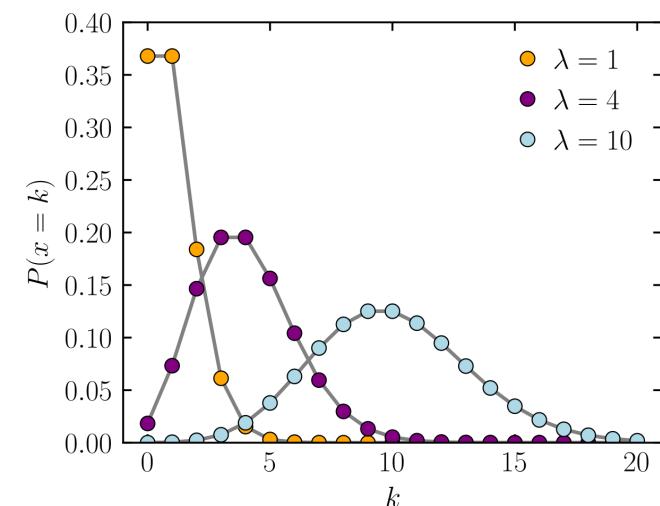
Control: Example – Jack's Car Rental

- 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

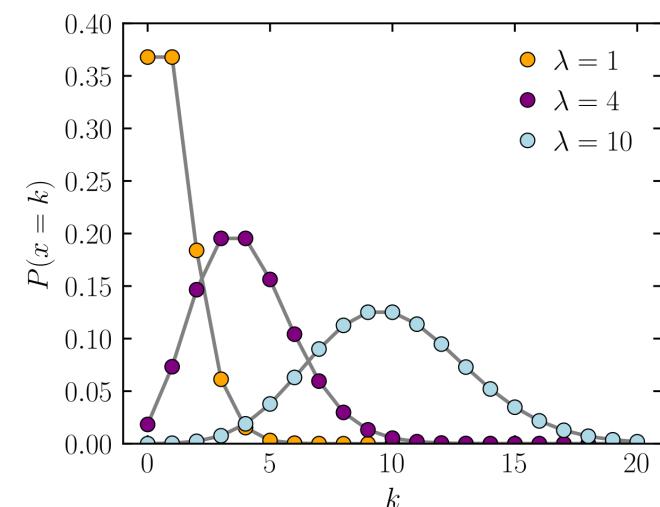
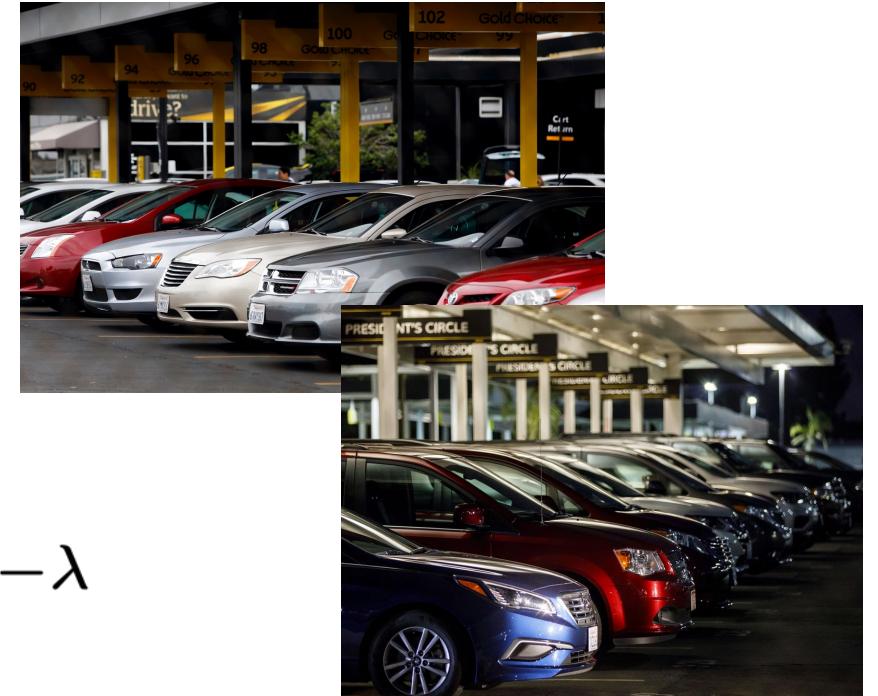


Control: Example – Jack’s Car Rental

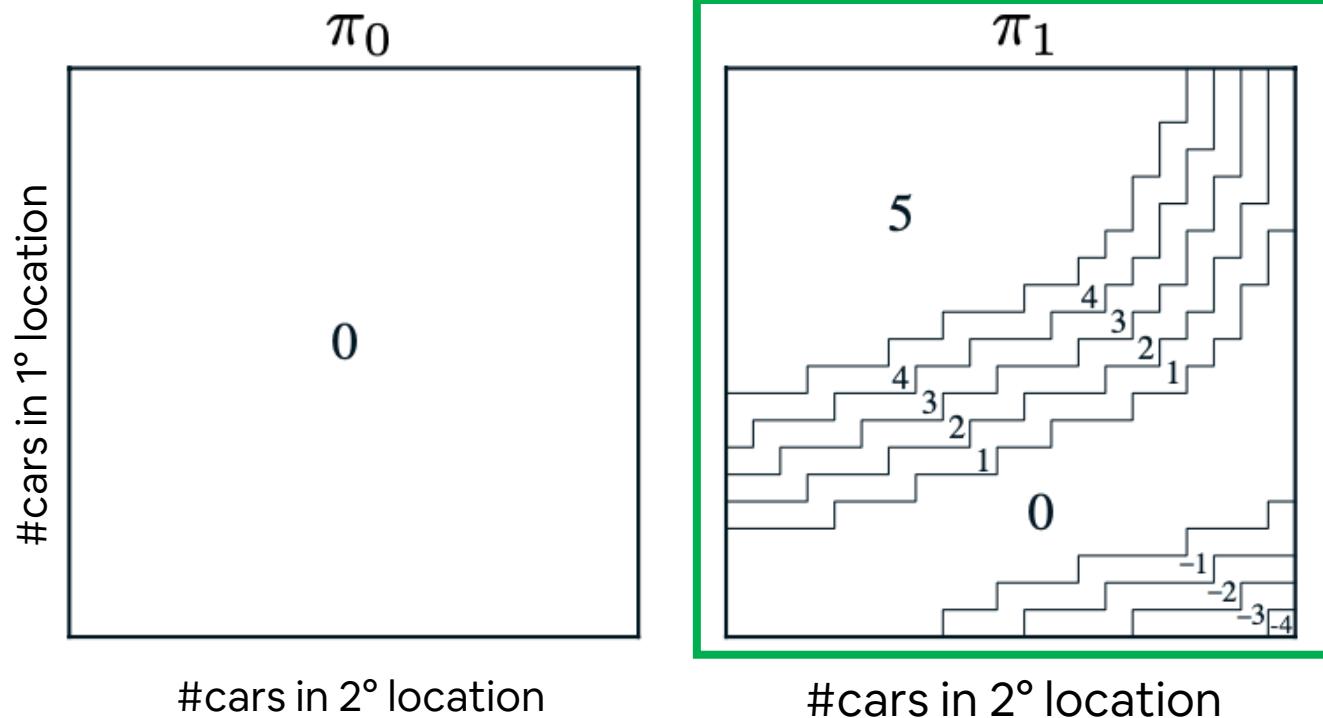
- We consider a discount rate $\gamma = 0.9$
- Time steps are days
- Actions are the net numbers of cars moved
- Jack can move the cars overnight for \$2 per car moved
- The number n of cars requested and returned are Poisson random variables
- 1° location: $\lambda = 3$ rental requests, $\lambda = 3$ 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

ions for a nationwide car

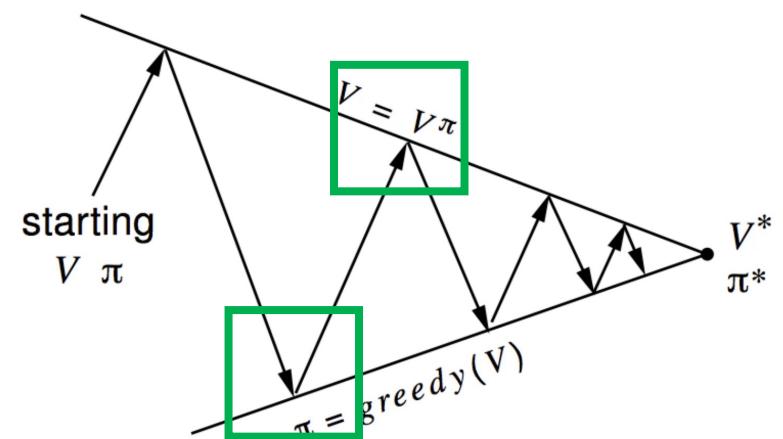
at one location and if he has the car for \$10



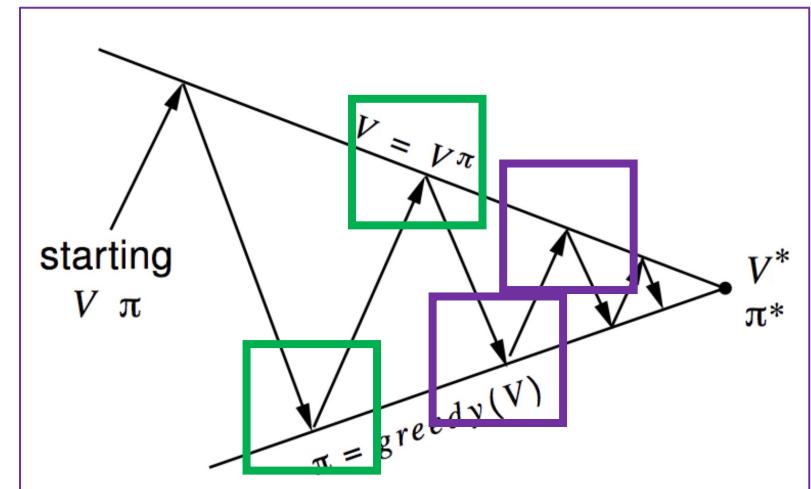
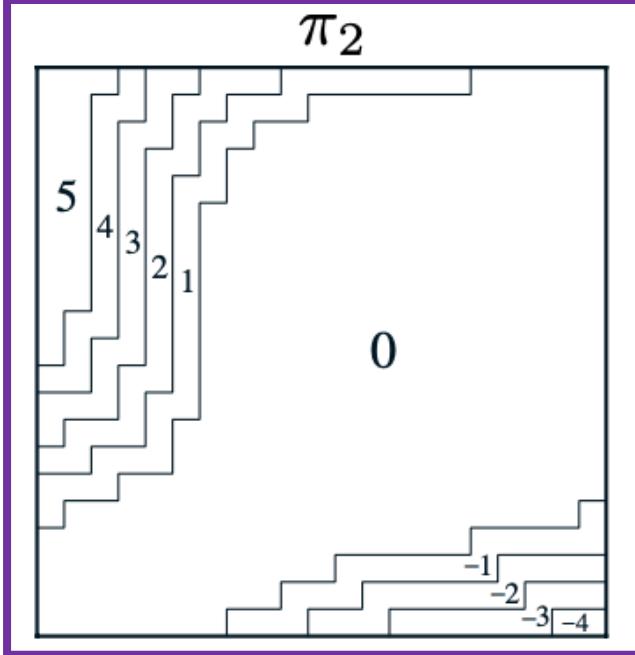
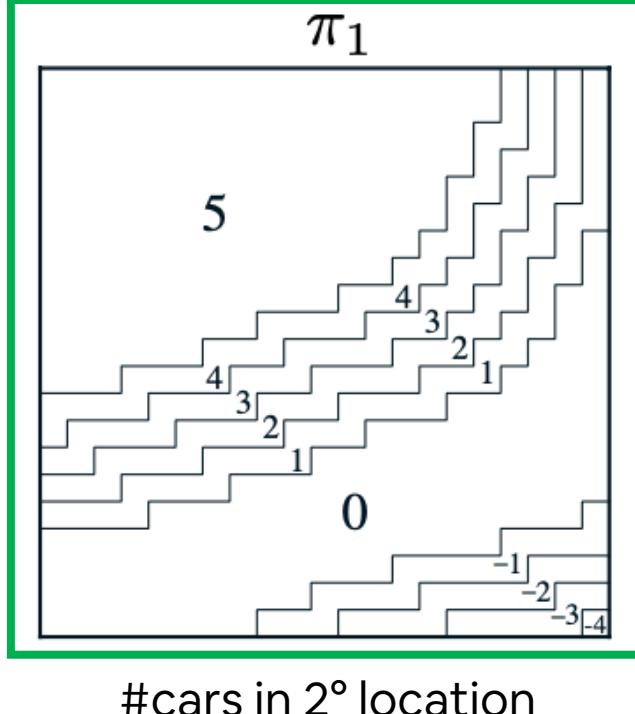
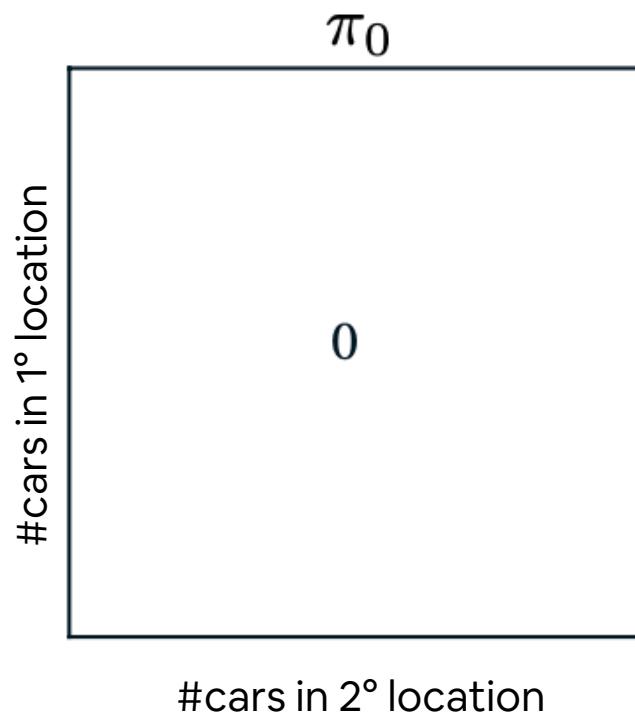
We start with the policy π_0 that moves zero cars. We then apply policy iteration



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

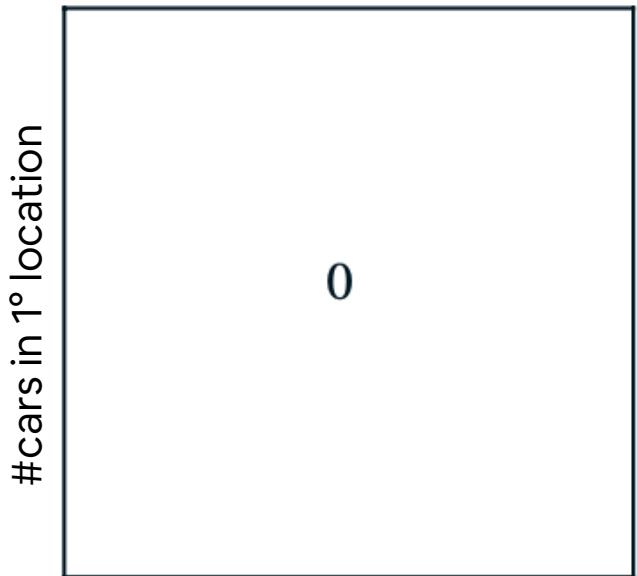


We start with the policy π_0 that moves zero cars. We then apply policy iteration

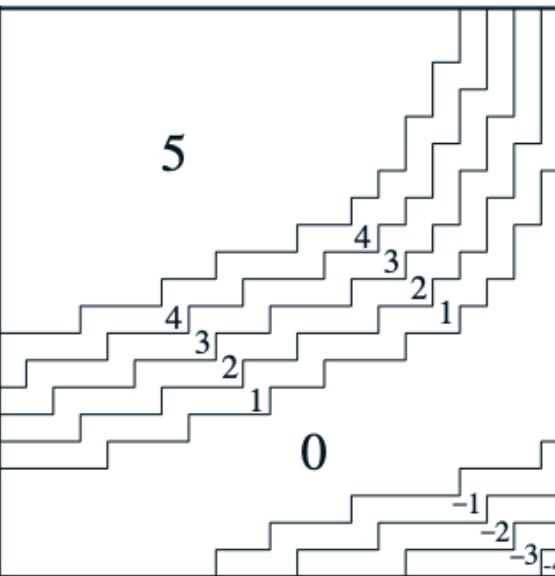


We start with the policy π_0 that moves zero cars. We then apply policy iteration

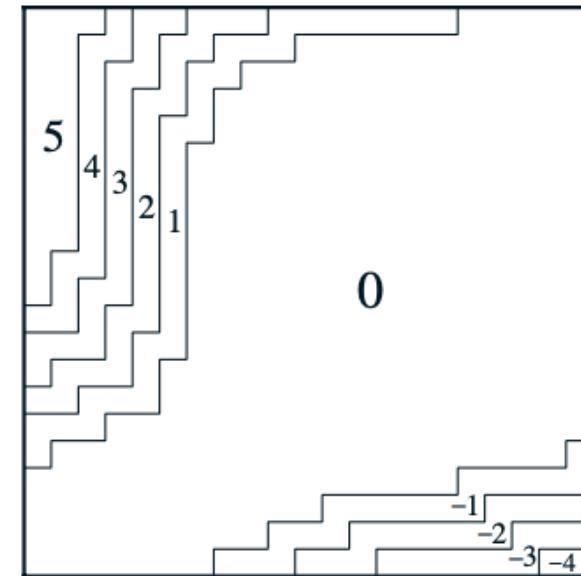
π_0



π_1

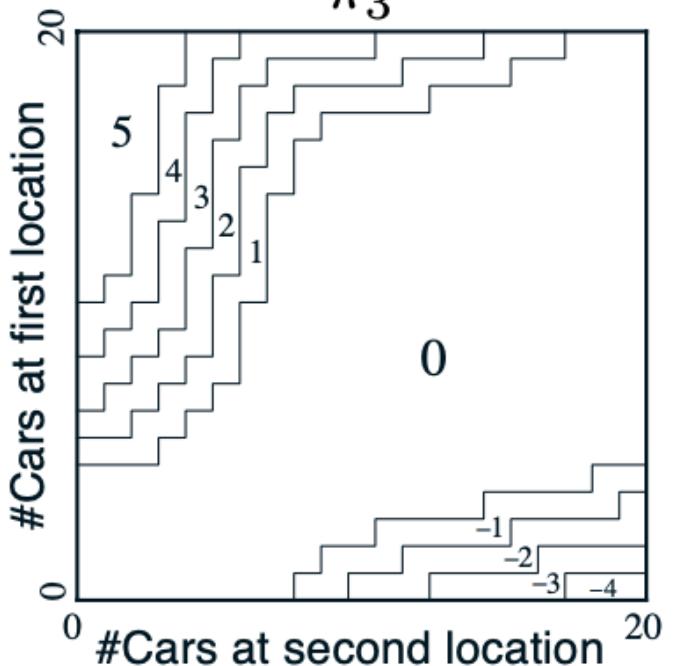


π_2



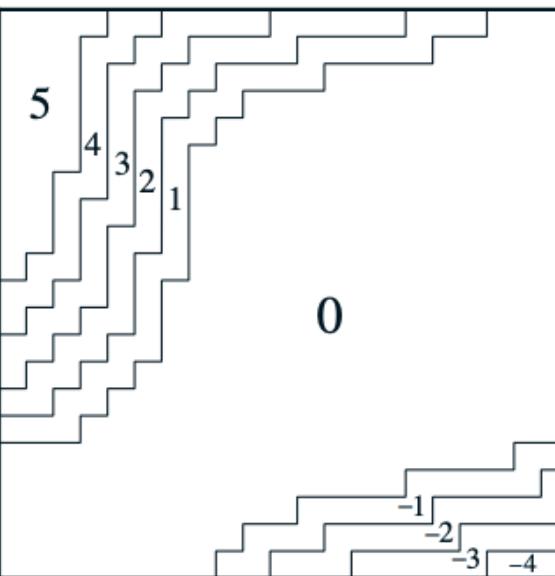
#cars in 2° location

π_3

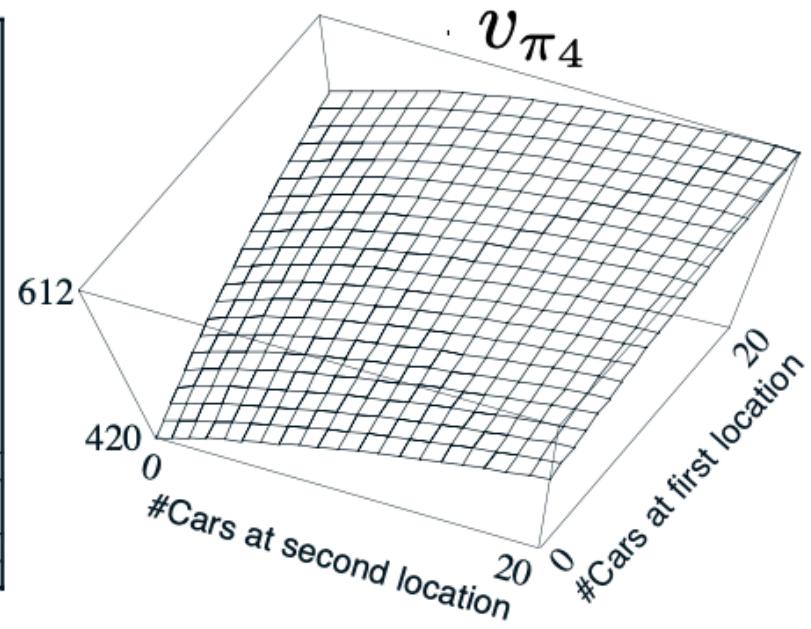


#cars in 2° location

π_4



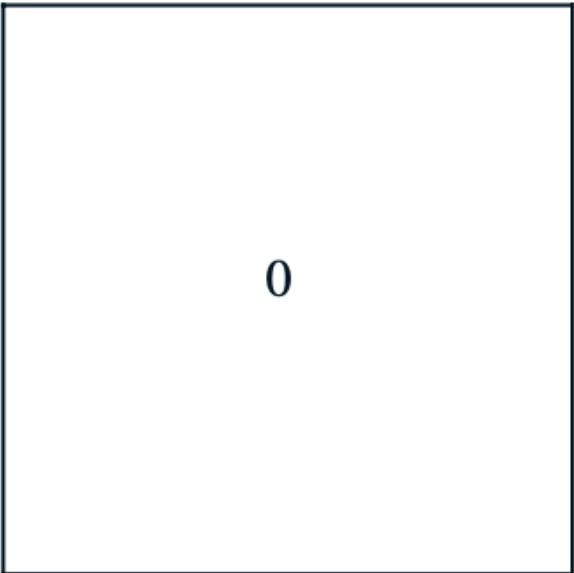
v_{π_4}



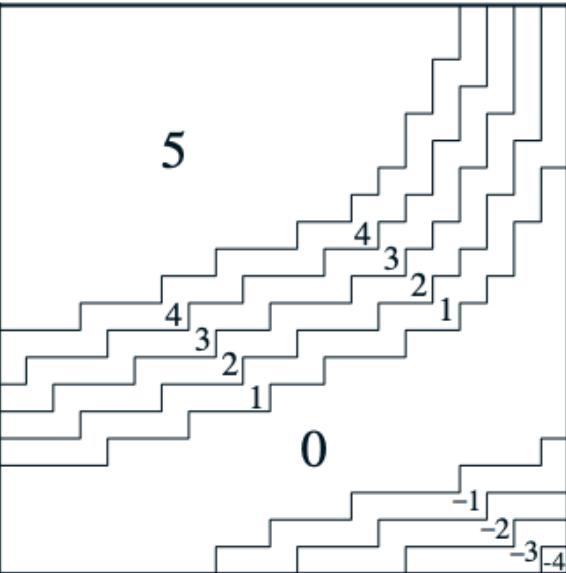
We start with the policy π_0 that moves zero cars. We then apply policy iteration

More on this on lab #3

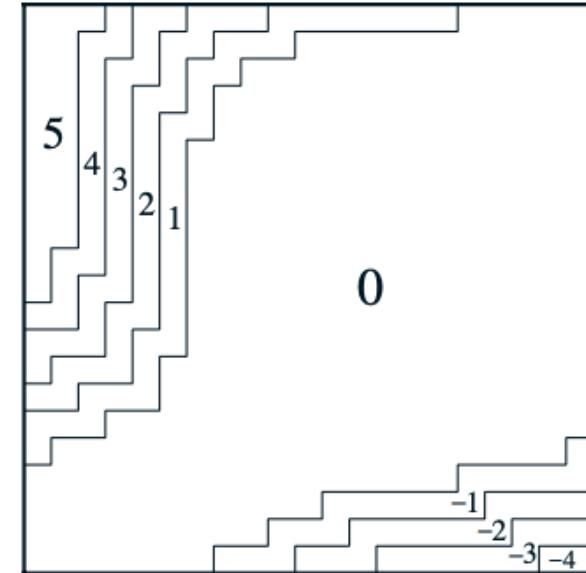
π_0



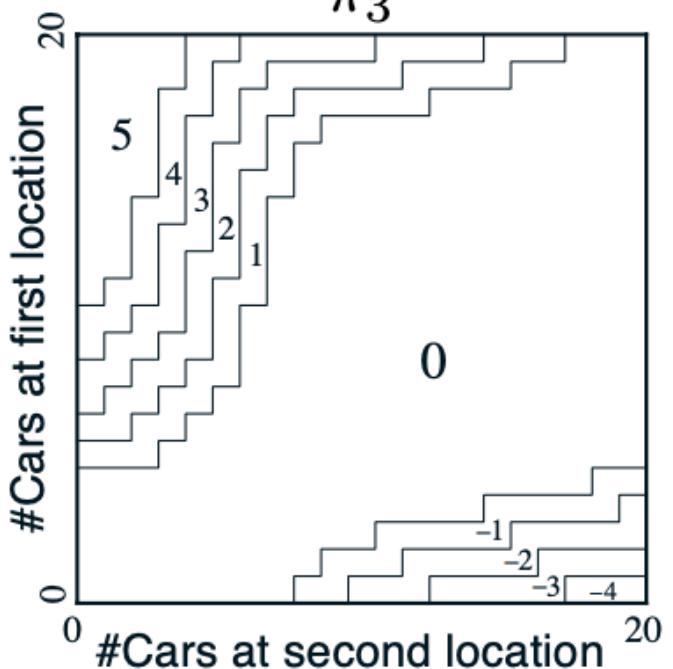
π_1



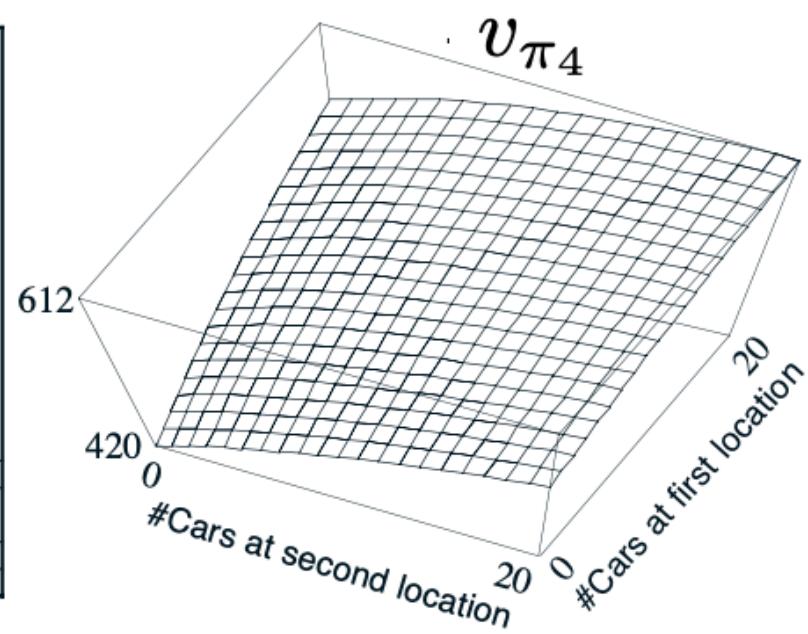
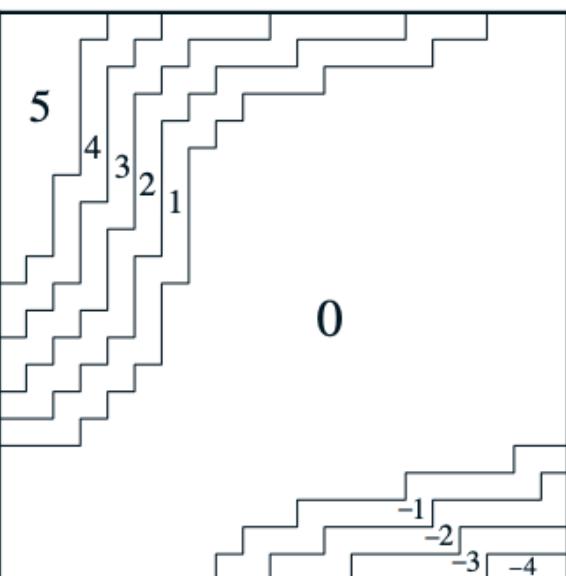
π_2



π_3



π_4



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/>



The screenshot shows the homepage of the MOST (Centro Nazionale per la Mobilità Sostenibile) website. At the top, there is a navigation bar with links for HOME, MOST, MEDIA, INITIATIVES, CONTACT, and IT/EN. There is also a GUEST AREA button. Below the navigation bar, there are logos for the European Union (NextGenerationEU), the Italian Ministry of University and Research, and Italidomani. The main banner features a blue-toned cityscape with a road and the text "WE LOOK TO THE FUTURE" and "A MORE SUSTAINABLE AND DIGITAL MOBILITY".

Control: Policy Iteration (Recap)

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 $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

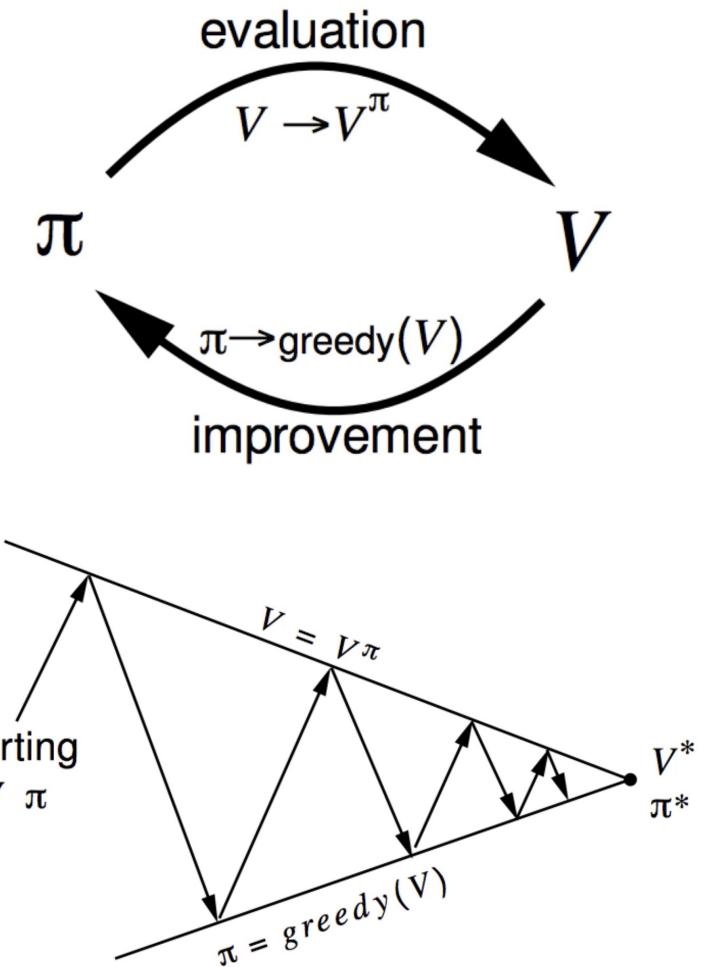
For each $s \in \mathcal{S}$:

$$\text{old-action} \leftarrow \pi(s)$$

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

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2



Control: Policy Iteration (Recap)

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 $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

This step can be quite slow... do we need the exact value of v_π ?

3. Policy Improvement

policy-stable \leftarrow true

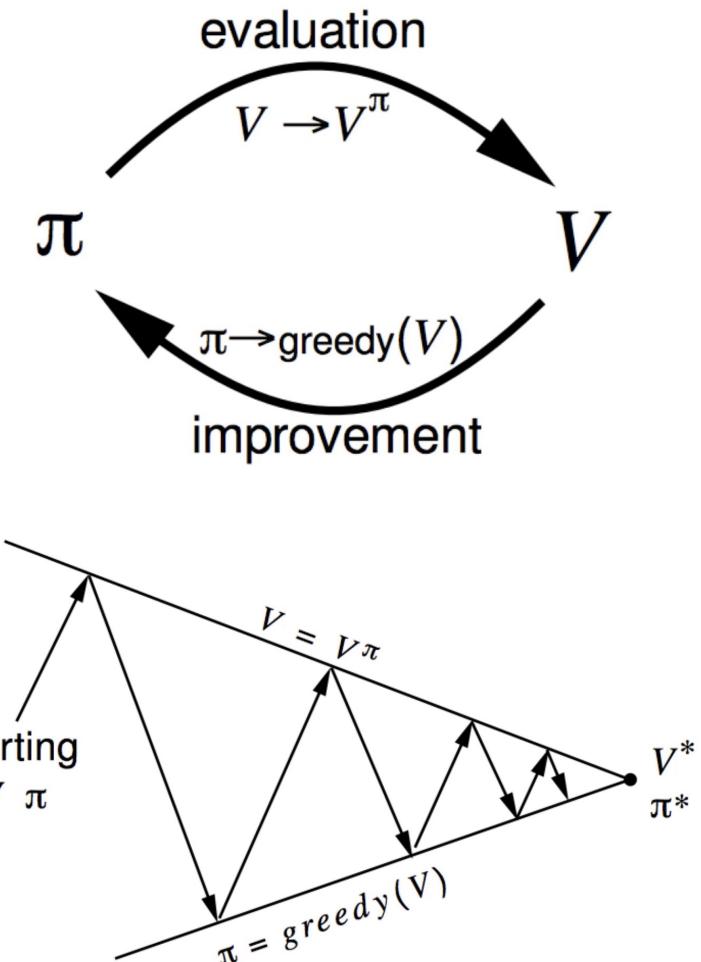
For each $s \in \mathcal{S}$:

$$\text{old-action} \leftarrow \pi(s)$$

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

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2



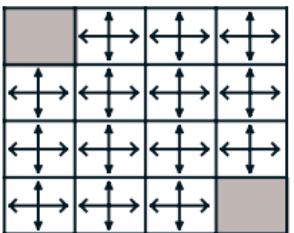
Recap

v_k for the random policy

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

$k = 0$

greedy policy
w.r.t. v_k



random policy

$k = 3$

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

$k = 1$

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

$k = 2$

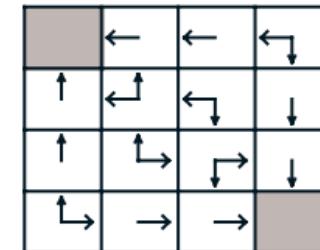
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 10$

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

$k = \infty$

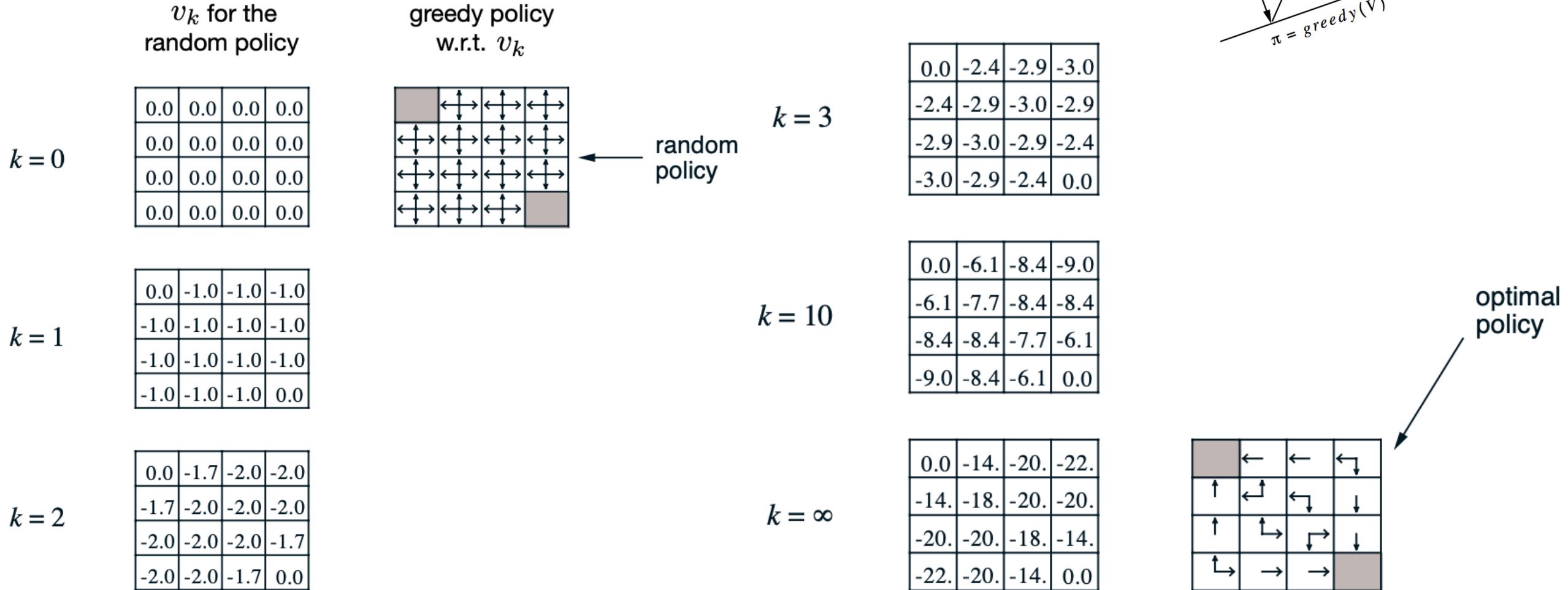
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



optimal policy

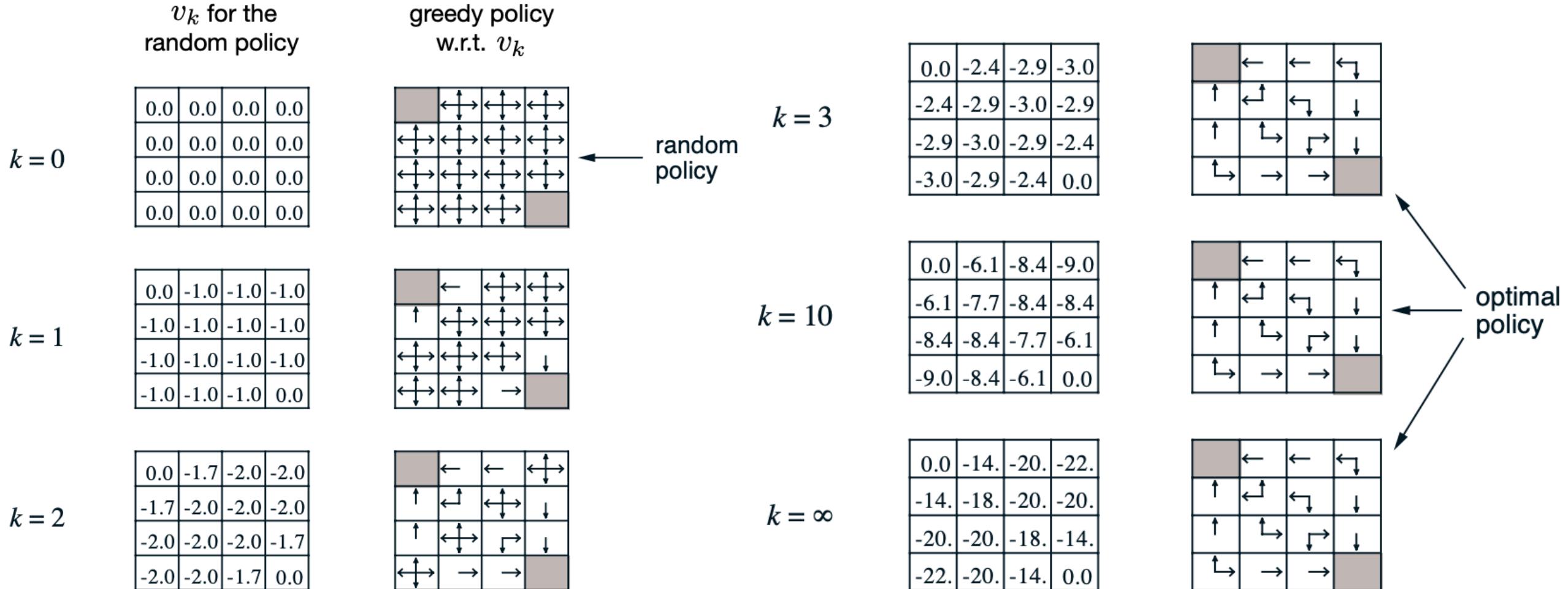
Just one iteration in this case:
evaluate the random policy -> gives us info->optimal policy

Recap



knowing the real value of v_{π} is not necessary to obtain the optimal policy

Recap



Control: Policy Iteration (Recap)

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 $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

This step can be quite slow... do we need the exact value of v_π ?

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

$$old-action \leftarrow \pi(s)$$

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

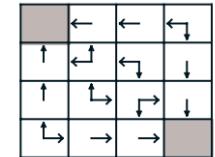
If $old-action \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

According to Policy Iteration, I should have to wait until having the true v_π

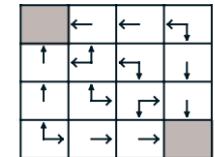
$k = 3$

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



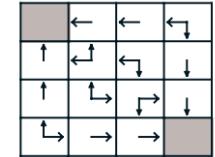
$k = 10$

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



the idea is maybe its not necessary if we already have the optimal policy

Control: Policy Iteration (Recap)

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 $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

This step can be quite slow... do we need the exact value of v_π ?

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

$$\text{old-action} \leftarrow \pi(s)$$

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

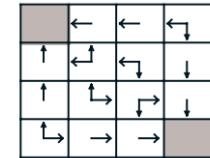
If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

According to Policy Iteration, I should have to wait until having the true v_π

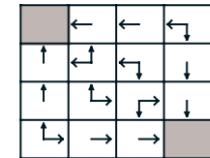
$k = 3$

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



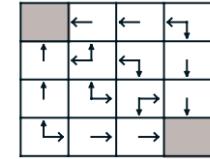
$k = 10$

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



When dealing with a RL problem we need to take into serious account efficiency!

Control: Policy Iteration (Recap)

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 $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

This step can be quite slow... do we need the exact value of v_π ?

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

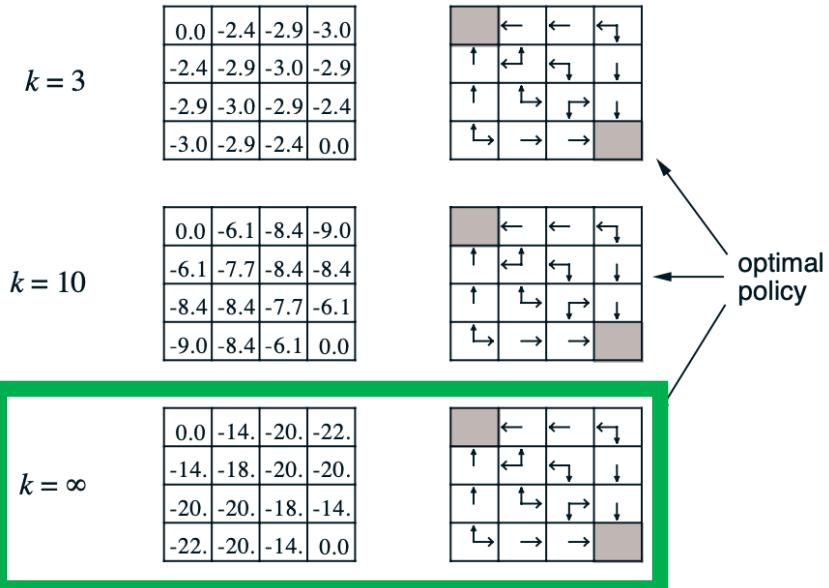
$$\text{old-action} \leftarrow \pi(s)$$

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

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

According to Policy Iteration, I should have to wait until having the true v_π



We can truncate the policy evaluation step... in reality we can do that in many ways without losing the convergence guarantees of policy iteration!

Control: Value Iteration

We consider the case where the evaluation step is stopped after one update! -> This approach is called Value Iteration

Control: Value Iteration

We consider the case where the evaluation step is stopped after one update! -> This approach is called Value Iteration

The approach can be re-written as:

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

for all states. We are combining policy evaluation and update in a single step!

Control: Value Iteration

We consider the case where the evaluation step is stopped after one update! -> This approach is called Value Iteration

The approach can be re-written as:

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

for all states. We are combining policy evaluation and update in a single step!

We can show (but we won't) that this approach would reach v^*

Control: Value Iteration

We consider the case where the evaluation step is stopped after one update! -> This approach is called Value Iteration

The approach can be re-written as:

$$v_{k+1}(s) \doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t=s, A_t=a]$$

This is an update rule!

$$= \max_a \sum_{s',r} p(s',r \mid s,a) [r + \gamma v_k(s')],$$

for all states. We are combining policy evaluation and update in a single step!

We can show (but we won't) that this approach would reach v^*

Policy Evaluation (Prediction) vs. Value Iteration (Control)

Policy evaluation uses the Bellman Expectation Equation as **an update rule**

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\pi}(s')]$$



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

Value iteration uses the Bellman Optimality Equation instead!

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_*(s')]$$



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

Policy Evaluation (Prediction) vs. Value Iteration (Control)

Policy evaluation uses the Bellman Expectation Equation as **an update rule**

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\pi}(s')]$$



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

Value iteration uses the Bellman Optimality Equation instead!

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_*(s')]$$



Instead of averaging over all actions, we just consider the ‘best’

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

Control: Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

```
|   Δ ← 0
|   Loop for each  $s \in \mathcal{S}$ :
|      $v \leftarrow V(s)$ 
|      $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
|     Δ ← max(Δ, |v - V(s)|)
```

until $\Delta < \theta$

Initialization

Output a deterministic policy, $\pi \approx \pi_*$, such that
 $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

Control: Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

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

Loop:

| $\Delta \leftarrow 0$

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

| | $v \leftarrow V(s)$

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

Updates: we are also considering here the ‘in place’ implementation

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

until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
 $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

Control: Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

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

Loop:

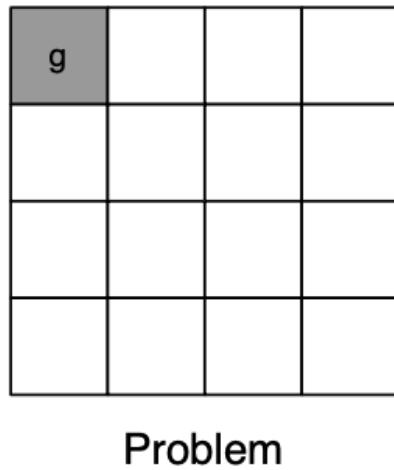
```
|  $\Delta \leftarrow 0$ 
| Loop for each  $s \in \mathcal{S}$ :
|    $v \leftarrow V(s)$ 
|    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
|    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
```

until $\Delta < \theta$

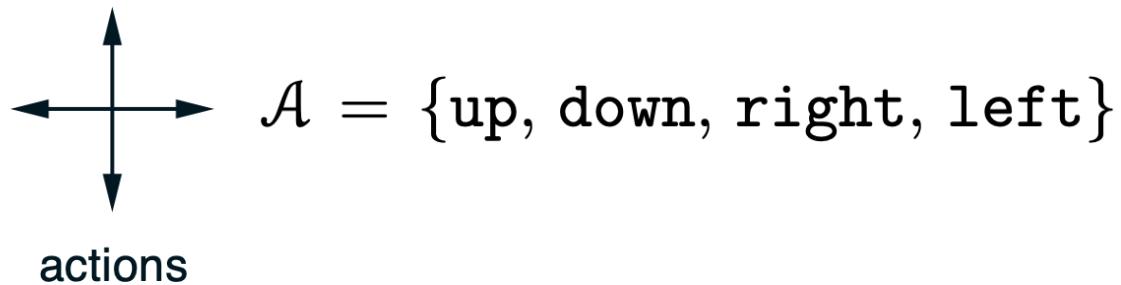
Termination criteria

Output a deterministic policy, $\pi \approx \pi_*$, such that
 $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

Control: Value Iteration on Small Gridworld



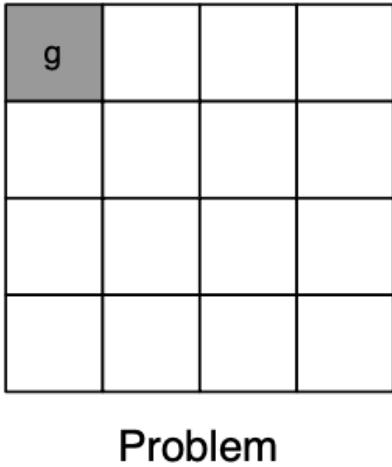
Nonterminal states: white box
Terminal states: grey boxes



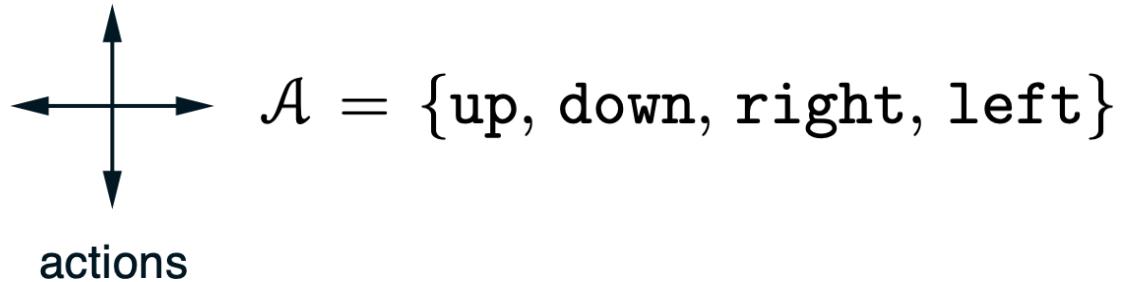
$$R_t = -1$$

on all transitions

Control: Value Iteration on Small Gridworld



Nonterminal states: white box
Terminal states: grey boxes



The book has another example ‘the gambler problem’: take a look!

$$R_t = -1$$

on all transitions

g			

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Problem

v_1

I start considering a random uniform policy

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

g				

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

v_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

v_2

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

g				

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Problem

v_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

v_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

v_3

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

g				

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

v_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

v_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

v_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

v_4

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

g				

This may work also
for problems with
non-terminal states!

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Problem

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

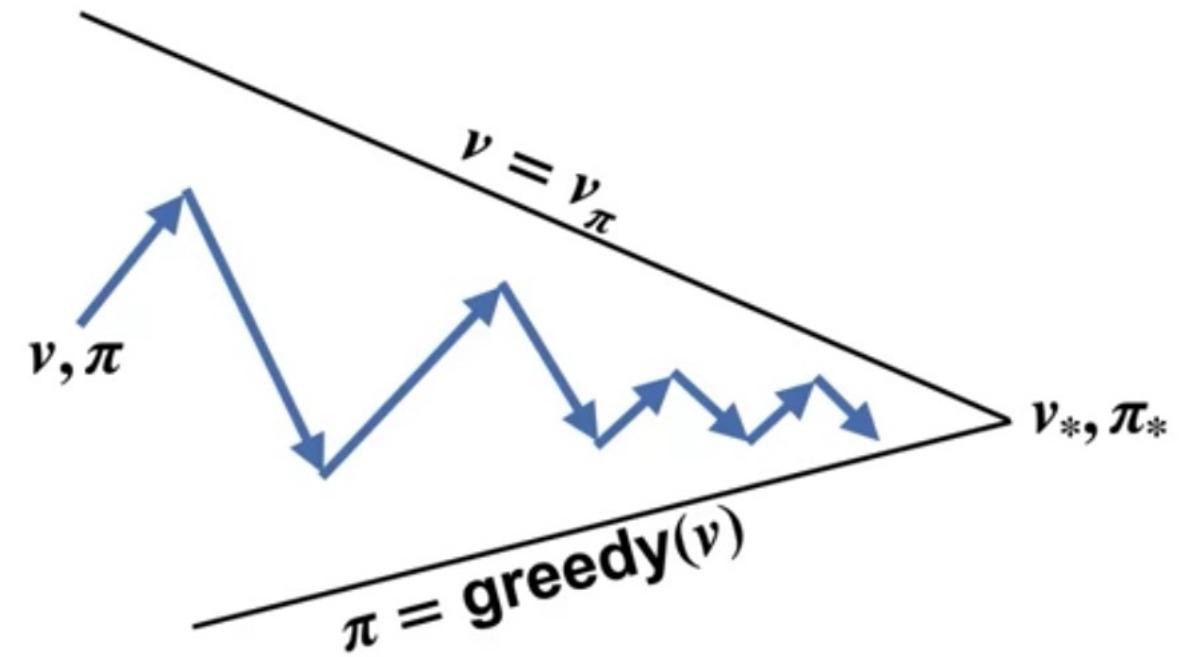
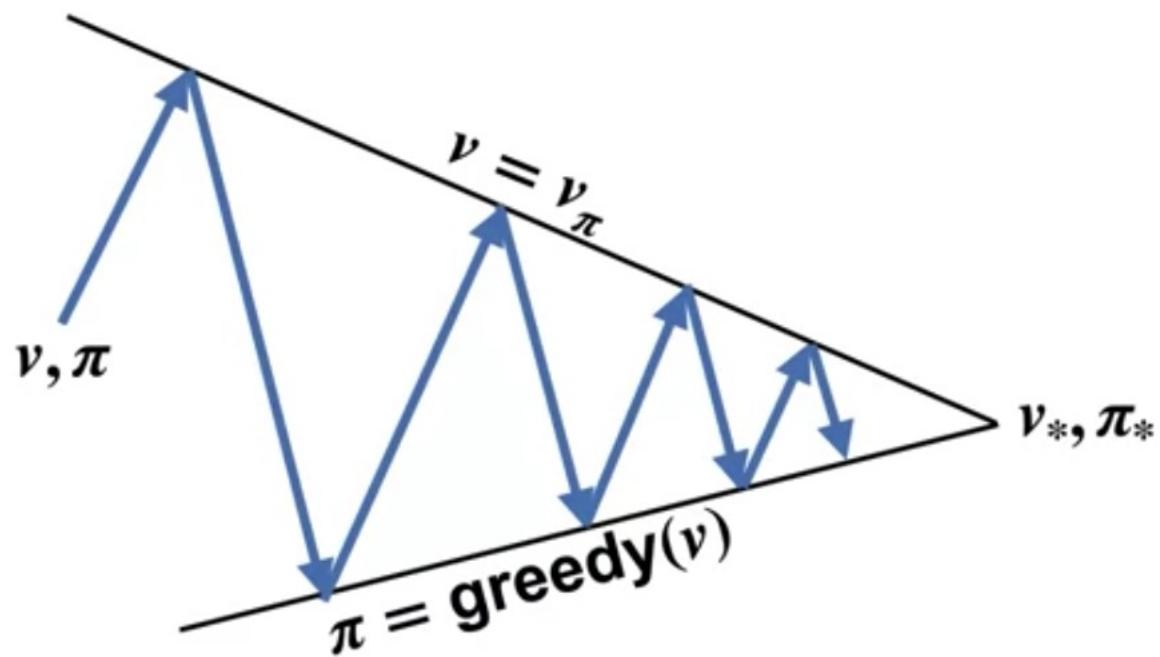
V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Control:

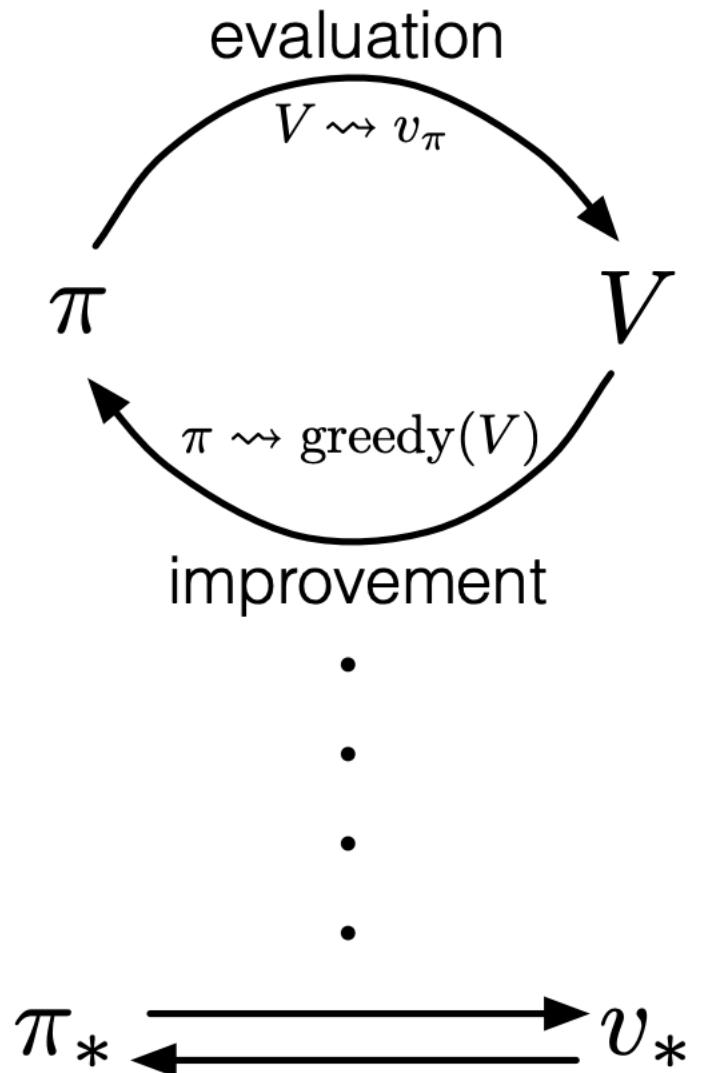
Policy Iteration vs. Value Iteration



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

Generalized Policy Iteration (GPI)

- We may end up with strategies that are hybrid version of Policy Iteration+Value Iteration
- GPI 'unifies' classical DP methods and include all the approaches where the procedure of evaluating and improving a policy interact



(Synchronous) DP Algorithms

Summary

Problem	Bellman Equation	Algorithm
Prediction	Bellman expectation equation	Iterative Policy Evaluation
Control	Bellman expectation equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman optimality equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v^*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q^*(s, a)$, but in that case the complexity would be $O(m^2n^2)$ per iteration

(Synchronous) DP Algorithms

Summary

Problem	Bellman Equation	Algorithm
Prediction	Bellman expectation equation	Iterative Policy Evaluation
Control	Bellman expectation equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman optimality equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v^*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q^*(s, a)$, but in that case the complexity would be $O(m^2n^2)$ per iteration
- How can I speed things up? Asynchronous DP

Asynchronous DP

- All the seen approaches have always a ‘**for all state**’ procedure: for such reason they are called **synchronous** (all states are backed up in parallel)

Asynchronous DP

- All the seen approaches have always a ‘**for all state**’ procedure: for such reason they are called **synchronous** (all states are backed up in parallel)
- **Asynchronous DP** methods give us the freedom to update states in any order (backs up states individually, in any order)

Asynchronous DP

- All the seen approaches have always a ‘**for all state**’ procedure: for such reason they are called **synchronous** (all states are backed up in parallel)
- **Asynchronous DP** methods give us the freedom to update states in any order (backs up states individually, in any order)
- In asynchronous DP for each selected state, apply the appropriate backup: in other words, they asynchronous DP methods are **in-place** approaches that are not organized in terms of systematic sweeps of the state set

Asynchronous DP

- All the seen approaches have always a ‘**for all state**’ procedure: for such reason they are called **synchronous** (all states are backed up in parallel)
- **Asynchronous DP** methods give us the freedom to update states in any order (backs up states individually, in any order)
- In asynchronous DP for each selected state, apply the appropriate backup: in other words, they asynchronous DP methods are **in-place** approaches that are not organized in terms of systematic sweeps of the state set
- Asynchronous DP methods can significantly reduce computation: If the state set is very large, then even a single sweep can be prohibitively expensive!

Asynchronous DP Example: Policy Evaluation

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 $V(\text{terminal}) = 0$

Loop:

$$\Delta \leftarrow 0$$

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

$$v \leftarrow V(s) \quad V'(s) \leftarrow V(s)$$

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

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

until $\Delta < \theta$

Synchronous update implementation
I am using a buffer to have systematic,
ordered updates

Asynchronous DP Example: Policy Evaluation

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 $V(\text{terminal}) = 0$

Loop:

$$\Delta \leftarrow 0$$

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

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$

‘In place’ update implementation

Asynchronous DP

- Asynchronous DP approaches update the values of states in any order whatsoever, using whatever values of other states happen to be available.

Asynchronous DP

- Asynchronous DP approaches update the values of states in any order whatsoever, using whatever values of other states happen to be available.
- Asynchronous DP methods still converge if all states continue to be selected

Asynchronous DP

- Asynchronous DP approaches update the values of states in any order whatsoever, using whatever values of other states happen to be available.
- Asynchronous DP methods still converge if all states continue to be selected
- Other examples:
 1. We can select the states to which we apply updates to improve the algorithm's rate of progress in a particular state.

Asynchronous DP

- Asynchronous DP approaches update the values of states in any order whatsoever, using whatever values of other states happen to be available.
- Asynchronous DP methods still converge if all states continue to be selected
- Other examples:
 1. We can select the states to which we apply updates to improve the algorithm's rate of progress in a particular state.
 2. Or we might even try to skip updating some states entirely if they are not relevant to optimal behavior

Asynchronous DP

- Asynchronous DP approaches update the values of states in any order whatsoever, using whatever values of other states happen to be available.
- Asynchronous DP methods still converge if all states continue to be selected
- Other examples:
 1. We can select the states to which we apply updates to improve the algorithm's rate of progress in a particular state.
 2. Or we might even try to skip updating some states entirely if they are not relevant to optimal behavior
 3. We may collect trajectories and see the states that are used by a policy and just update those

Asynchronous DP

- Asynchronous DP approaches update the values of states in any order whatsoever, using whatever values of other states happen to be available.
- Asynchronous DP methods still converge if all states continue to be selected
- Other examples:
 1. We can select the states to which we apply updates to improve the algorithm's rate of progress in a particular state.
 2. Or we might even try to skip updating some states entirely if they are not relevant to optimal behavior
 3. We may collect trajectories and see the states that are used by a policy and just update those
 4. [Prioritized Sweeping – Chapter 8] Or we can try to order the updates to let value information propagate from state to state in an efficient way: some states may not need their values updated as often as others.

An example of Asynchronous DP method: Prioritized Sweeping (chapter 8)

- Use magnitude of Bellman error to guide state selection, e.g.

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

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

Dynamic Programming: Exam

- All the content of Chapter 4 are Exam material!
- Pay particular attention to the algorithms!

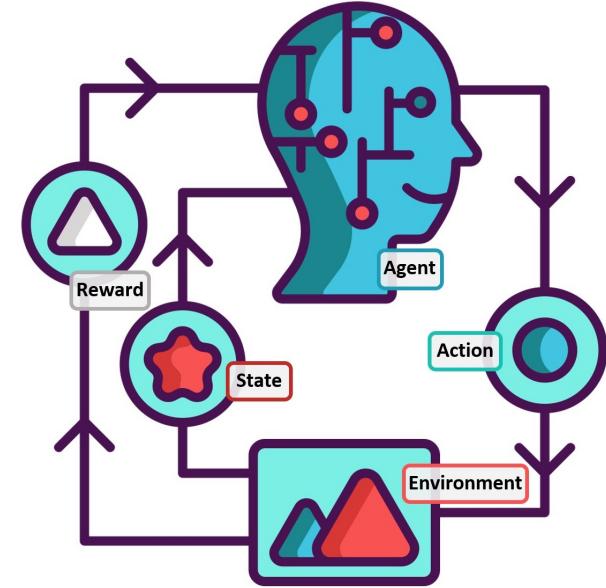
Monte Carlo Methods (Chapter 5)



Some problems that can be formalized in a RL fashion (from Lecture #01)

- Autonomous agents (self-driving cars, drones, robots...)
- Games
- HVAC (Heating, Ventilating, Air Conditioning) energy optimization
- Trading and Portfolio management
- Online advertising & Recommendation systems (news, items, ...)
- Healthcare, biology

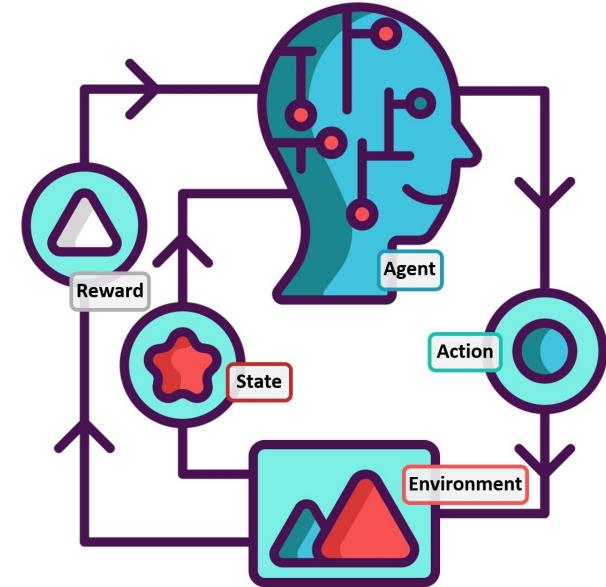
....



Some problems that can be formalized in a RL fashion (from Lecture #01)

- Autonomous agents (self-driving cars, drones, robots...)
- Games
- HVAC (Heating, Ventilating, Air Conditioning) energy optimization
- Trading and Portfolio management
- Online advertising & Recommendation systems (news, items, ...)
- Healthcare, biology

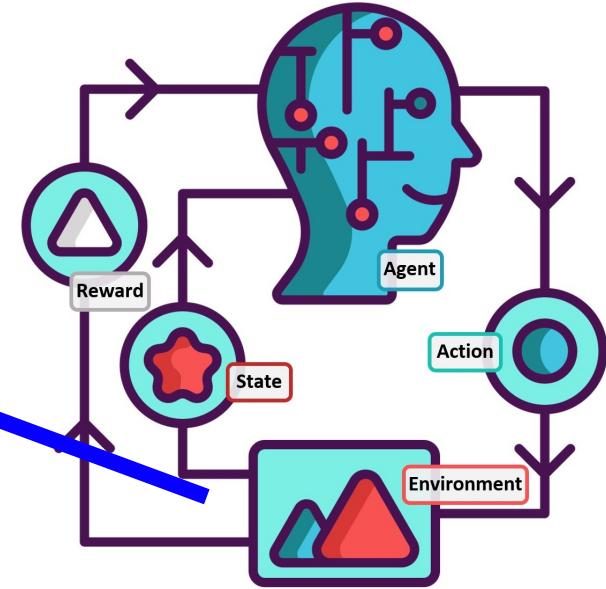
All of these problems can be formalized as MDPs ($(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$), but defining an MDP may be impossible!
We can have an unknown environment behaviour!



Some problems that can be formalized in a RL fashion (from Lecture #01)

- Autonomous agents (self-driving cars, drones, robots...)
- Games
- HVAC (Heating, Ventilating, Air Conditioning) energy optimization
- Trading and Portfolio management
- Online advertising & Recommendation systems (news, items, ...)
- Healthcare, biology

All of these problems can be formalized as MDPs ($(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$), but defining an MDP may be impossible! We can have an unknown environment behaviour!



Monte Carlo Methods

- Monte Carlo (MC) Methods are simple, but effective, approaches to deal with the ‘full’ RL problem (without knowledge of the MDP’s \mathcal{P} \mathcal{R})
- MC methods only require experience: sample sequences of states, actions and rewards (real world data gathered by interacting with the environment)
- MC will be our first ‘model-free’ approach (for both prediction and control)
- With MC methods we require episodic tasks

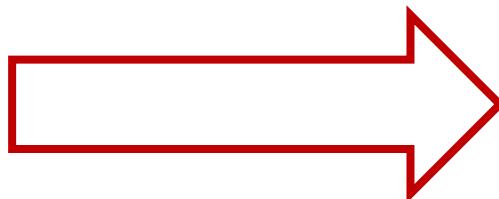
Monte Carlo Methods

- Monte Carlo (MC) Methods are simple, but effective, approaches to deal with the ‘full’ RL problem (without knowledge of the MDP’s \mathcal{P} \mathcal{R})
- MC methods only require experience: sample sequences of states, actions and rewards (real world data gathered by interacting with the environment)
- MC will be our first ‘model-free’ approach (for both prediction and control)
- With MC methods we require episodic tasks

We will only look at prediction today!
Control on next lecture!

Pay attention, from now onwards in this course:

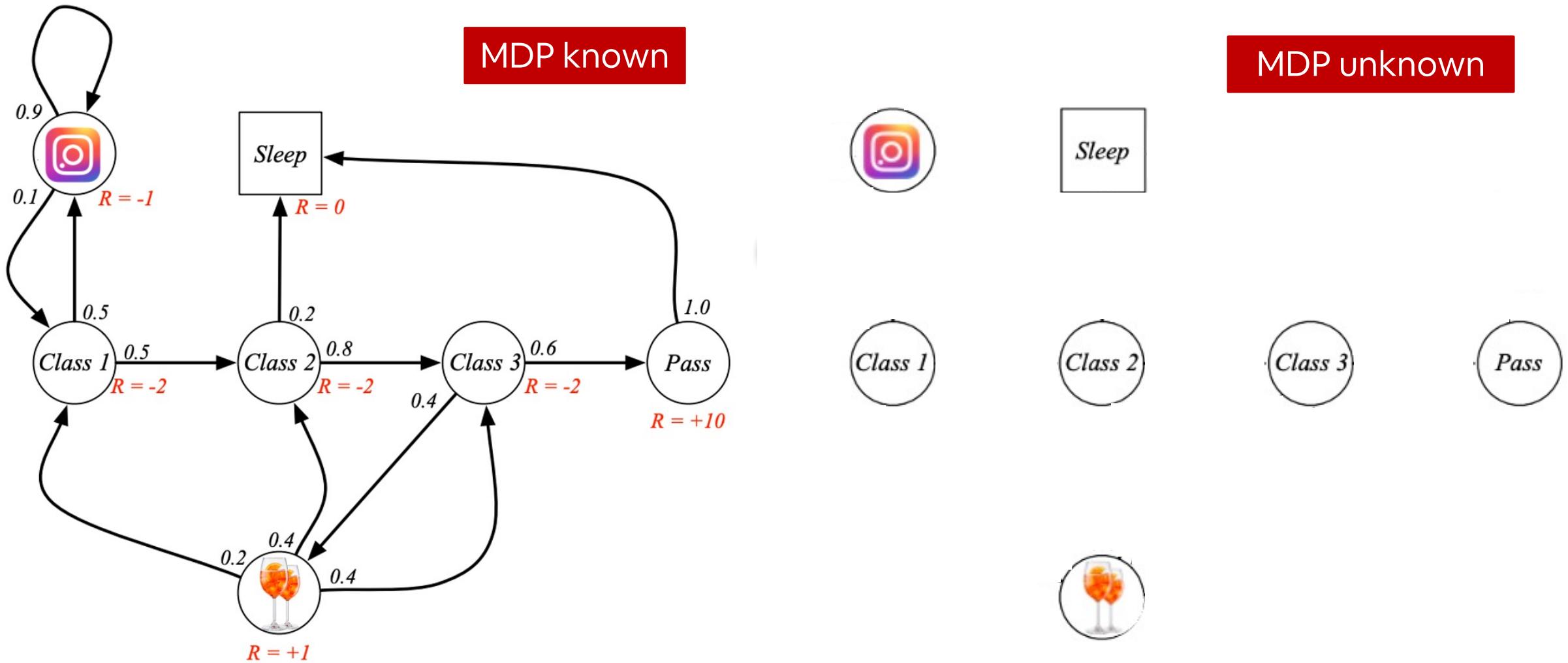
\mathcal{P}, \mathcal{R}
known



Data,
experience

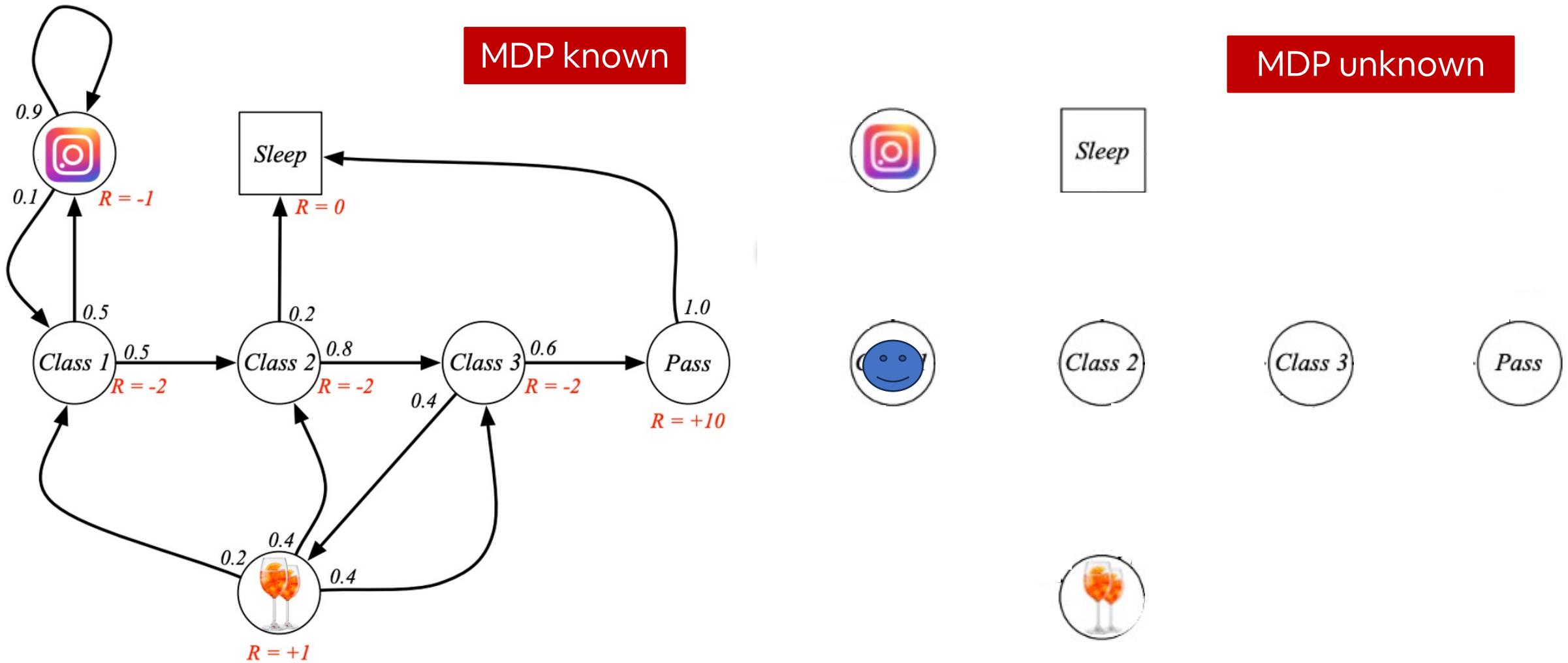
\mathcal{P}, \mathcal{R}
known

Data,
experience



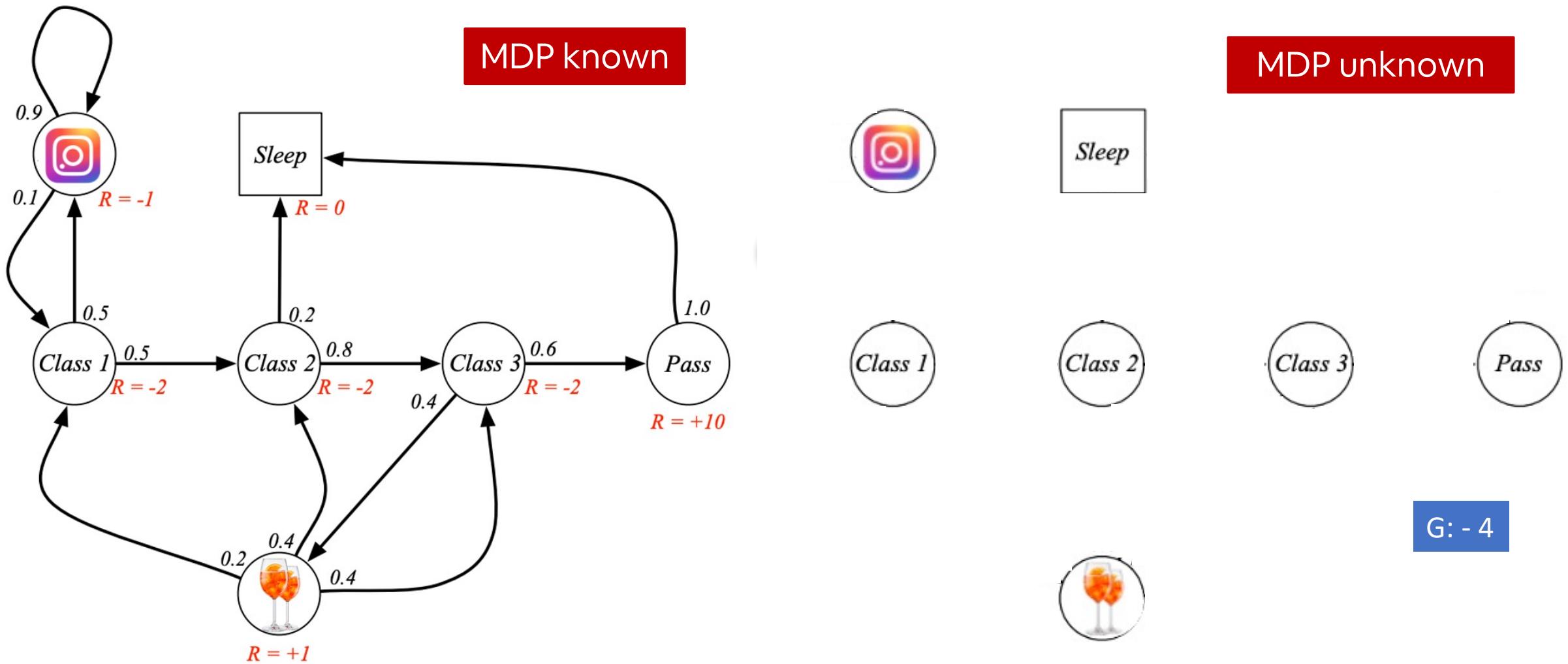
\mathcal{P}, \mathcal{R}
known

Data,
experience



\mathcal{P}, \mathcal{R}
known

Data,
experience



Monte Carlo Methods

- The term ‘Monte Carlo’ is often used more broadly for any estimation method that relies on repeated random sampling
- For example: estimating the average sum of 12 dice rolls

Sum of Faces	Probability
12	?
13	?
14	?
...	
71	?
72	?

Samples

39 38 49 36 33 47 37 49 41 40 44 44

35 43 47 28 46 52 47 45 43 43 42 43

40 32 29 48 43 49 48 39 42 35 44 48

36 32 41 49 34 54 37 42 37 38 42 50

39 37 45 49 44 34 38 50 35 36 42 45

Monte Carlo Methods

- The term ‘Monte Carlo’ is often used more broadly for any estimation method that relies on repeated random sampling
- For example: estimating the average sum of 12 dice rolls

Sum of Faces	Probability
12	?
13	?
14	?
...	
71	?
72	?

Samples

39 38 49 36 33
35 43 47 28 46 52 47 45 43 43 42 43
40 32 29 48 43 49 48 39 42 35 44 48
36 32 41 49 34 54 37 42 37 38 42 50
39 37 45 49 44 34 38 50 35 36 42 45

Average = 41.57

MC methods estimate expectations by averaging over a large number of random samples

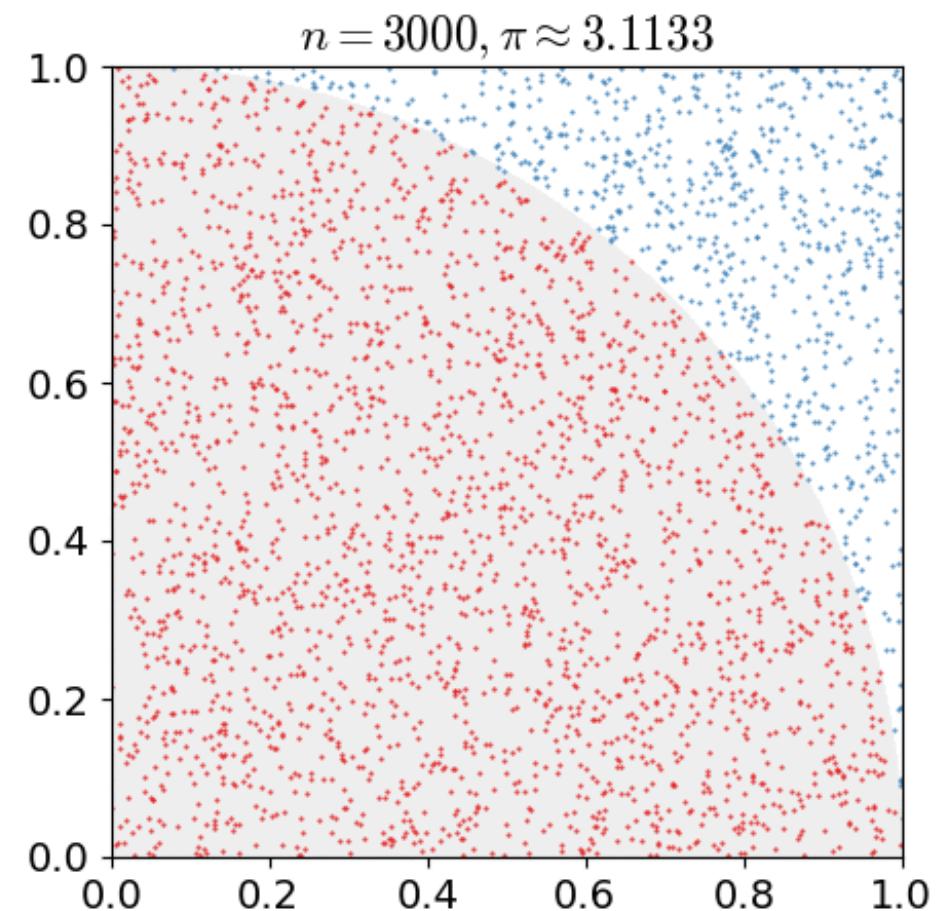
Expectations in RL

- In RL we are looking for estimation state value functions (and action-value functions) that are defined as expectations

$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s]$$

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

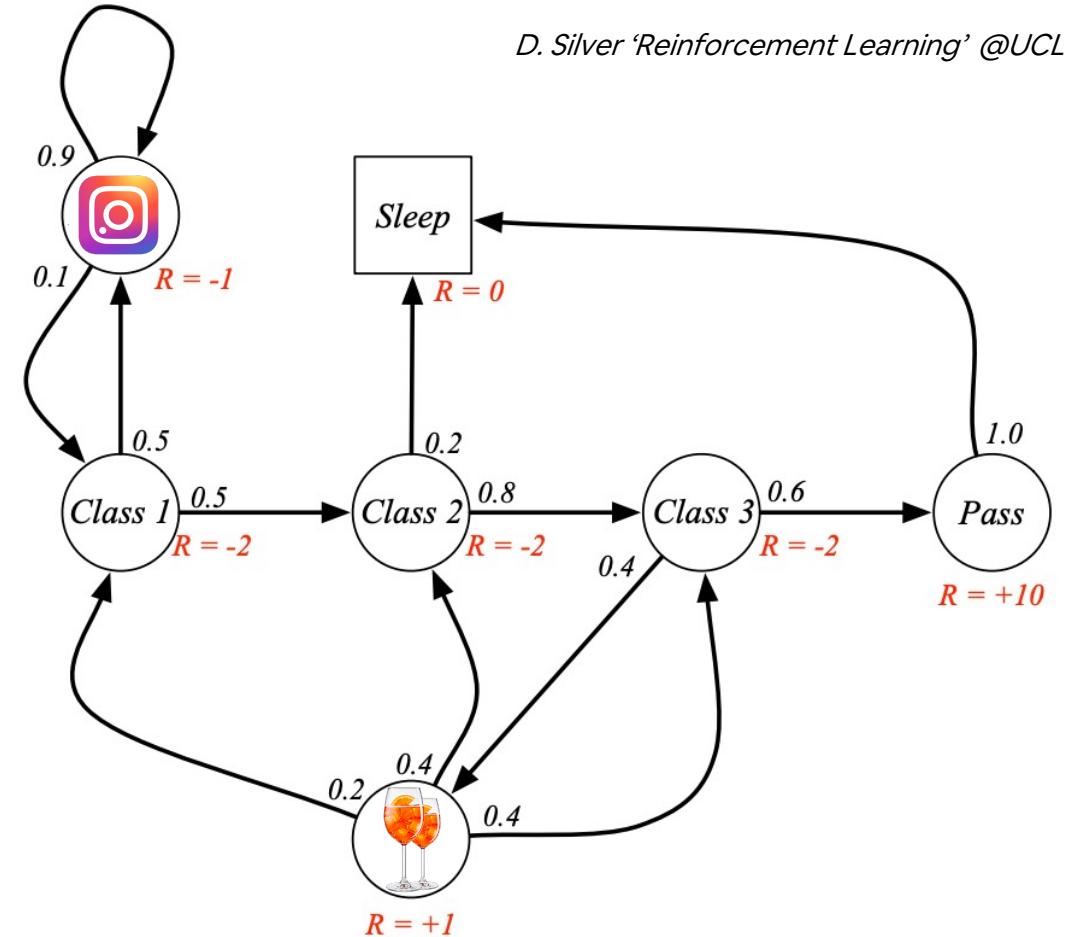
- If we have episodes, **we can see returns (we have data)!**
- We can approximate the ‘true’ expected returns with the average of the returns seen if we get many data collected!



Prediction: Let's see an example!

In the student MDP (with $\gamma = 1$), we draw some samples: following the depicted policy π
[#1] C1(-2)C2(-2)C3(+10)Pass(0)Sleep

Let's estimate the value function for state C1 with MC methods:



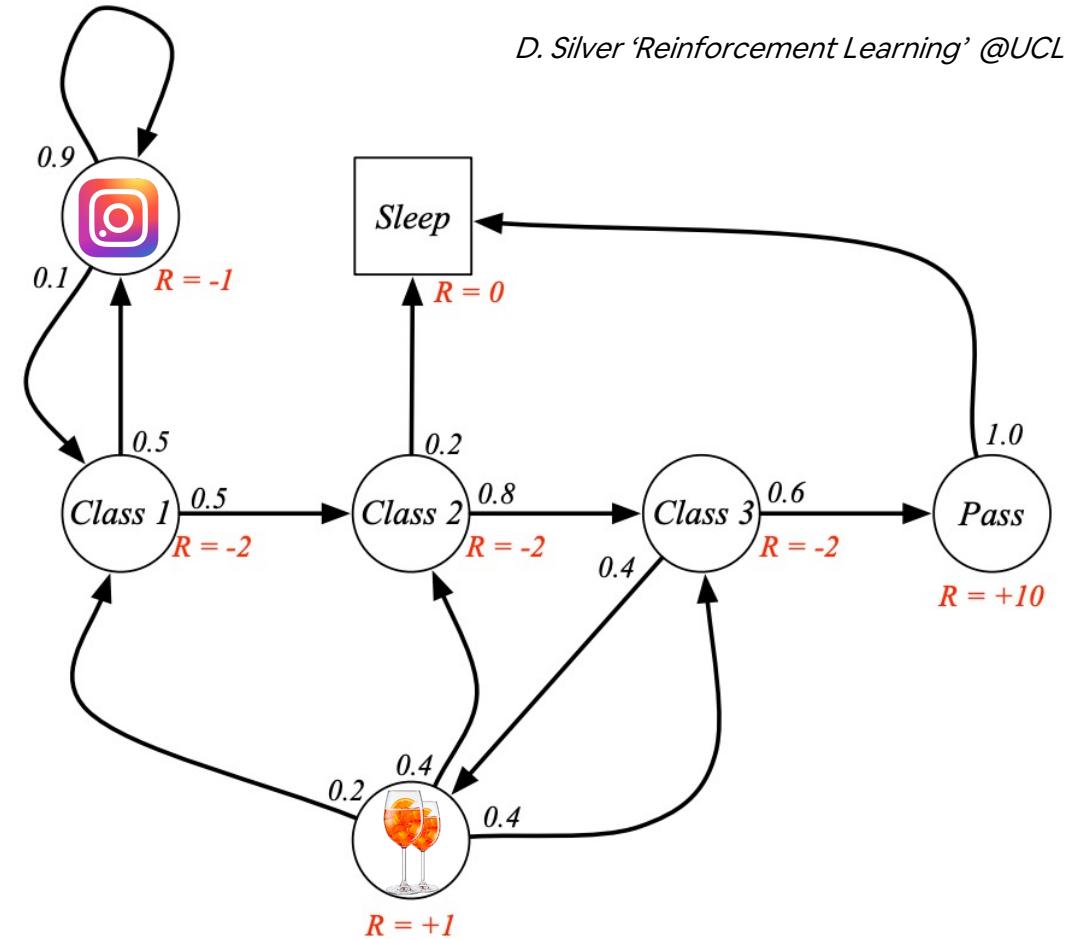
- For simplicity I am showing you a MRP, but the concept holds also for the MDP
- I don't know the MDP (the graph is not available!)

Prediction: Let's see an example!

In the student MDP (with $\gamma = 1$), we draw some **samples**: following the depicted policy π

[#1] C1(-2)C2(-2)C3(-2)Pass(+10)Sleep

Let's estimate the value function for state C1 with MC methods:
[episode #1] +4



Prediction: Let's see an example!

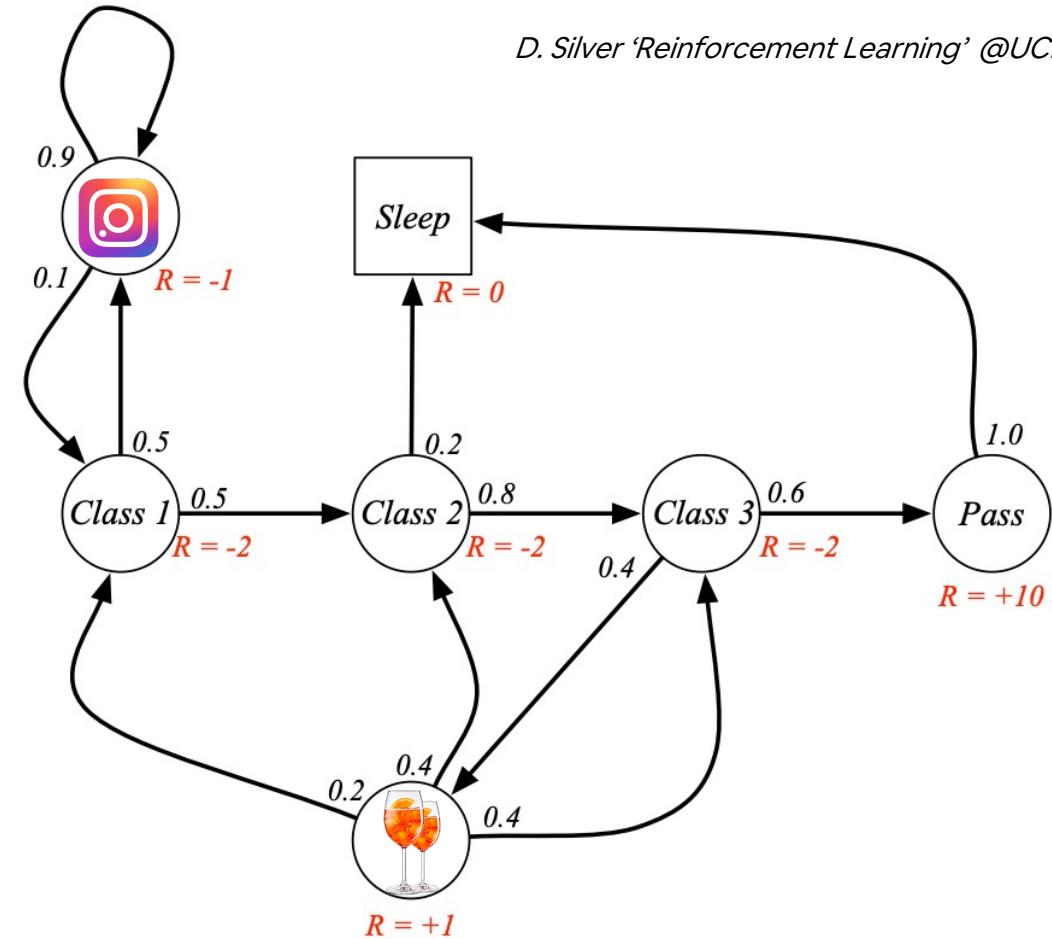
In the student MDP (with $\gamma = 1$), we draw some **samples**: following the depicted policy

π

[#1] C1(-2)C2(-2)C3(-2)Pass(+10)Sleep

[#2] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)Sleep

D. Silver 'Reinforcement Learning' @UCL



Let's estimate the value function for state C1 with MC methods:

[episode #1] +4

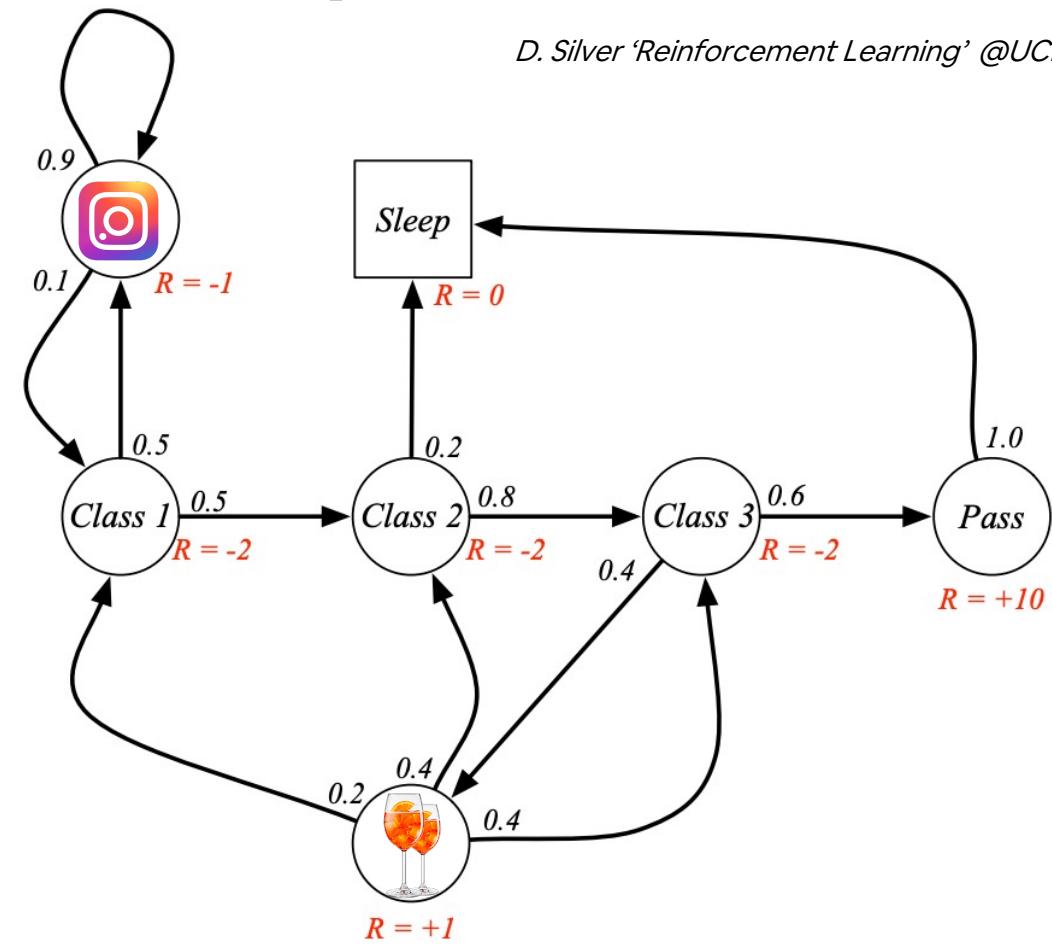
[episode #2] -8, -4

Prediction: Let's see an example!

In the student MDP (with $\gamma = 1$), we draw some **samples**: following the depicted policy π

- [#1] C1(-2)C2(-2)C3(-2)Pass(+10)Sleep
- [#2] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)Sleep
- [#3] C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)C3(-2)Pass(+10)Sleep

D. Silver 'Reinforcement Learning' @UCL



Let's estimate the value function for state C1 with MC methods:

- [episode #1] +4
- [episode #2] -8, -4
- [episode #3] +1

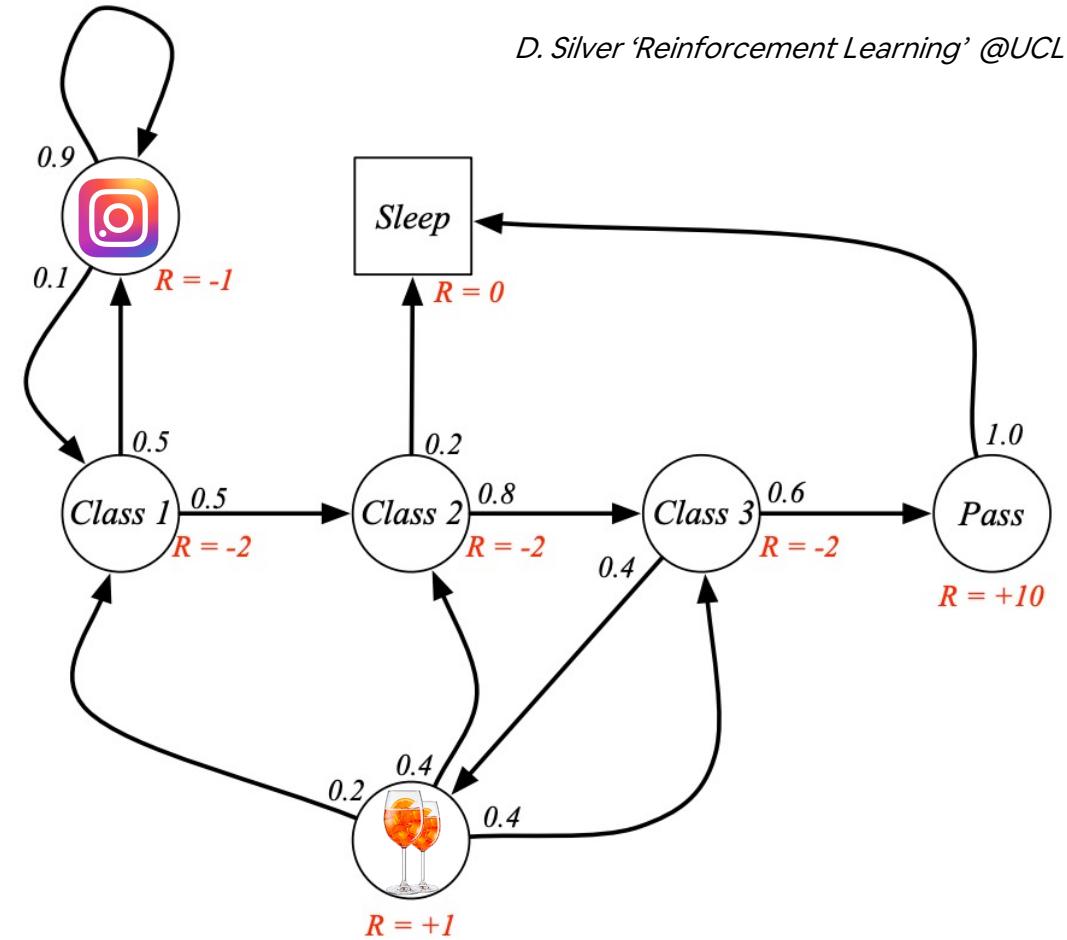
Prediction: Let's see an example!

In the student MDP (with $\gamma = 1$), we draw some **samples**: following the depicted policy π

- [#1] C1(-2)C2(-2)C3(-2)Pass(+10)Sleep
- [#2] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)Sleep
- [#3] C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)C3(-2)Pass(+10)Sleep
- [#4] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C1(-2)IG(-1)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)Sleep

Let's estimate the value function for state C1 with MC methods:

- [episode #1] +4
- [episode #2] -8, -4
- [episode #3] +1
- [episode #4] -20, -16, -11, -7



Prediction: Let's see an example!

In the student MDP (with $\gamma = 1$), we draw some **samples**: following the depicted policy π

- [#1] C1(-2)C2(-2)C3(-2)Pass(+10)Sleep
- [#2] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)Sleep
- [#3] C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)C3(-2)Pass(+10)Sleep
- [#4] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C1(-2)IG(-1)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)Sleep

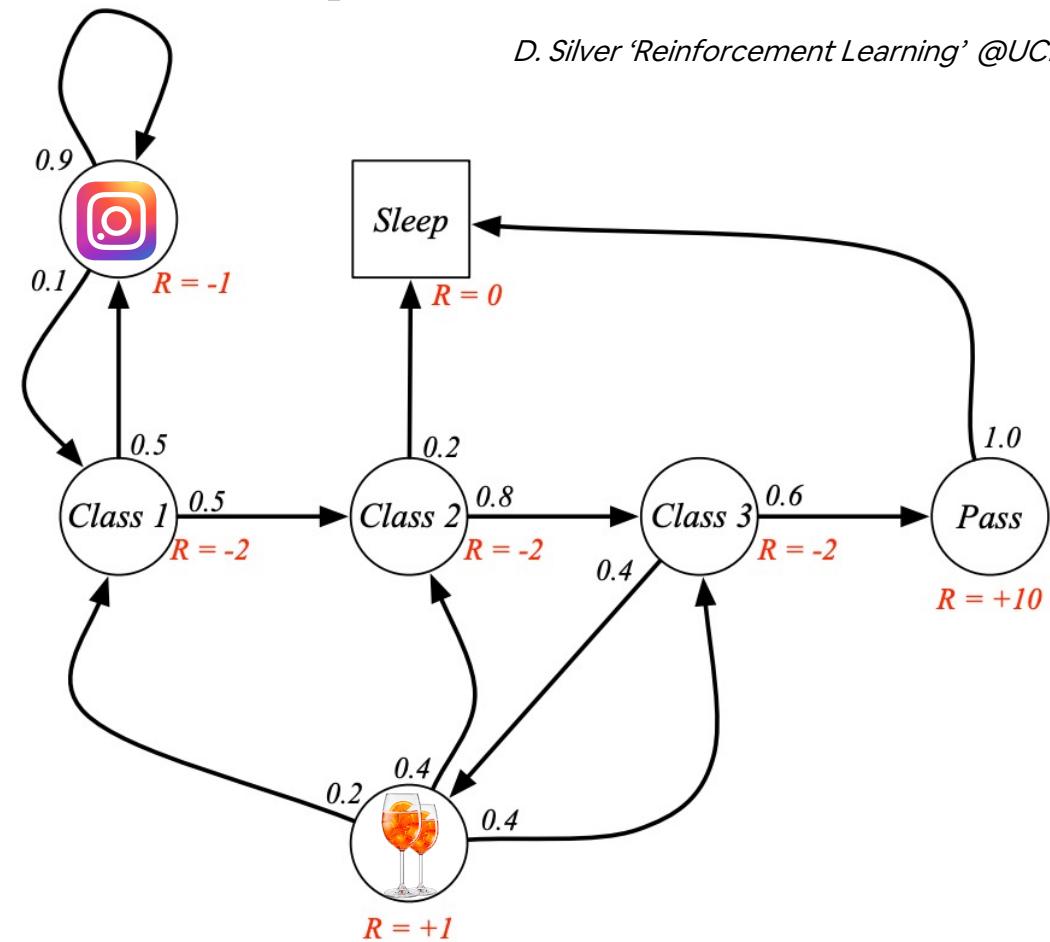
Let's estimate the value function for state C1 with MC methods:

- [episode #1] +4
- [episode #2] -8, -4
- [episode #3] +1

[episode #4] -20, -16, -11, -7

$$v_{\pi} = -7.625 = 0.125(4-8-4+1-20-16-11-7)$$

D. Silver 'Reinforcement Learning' @UCL



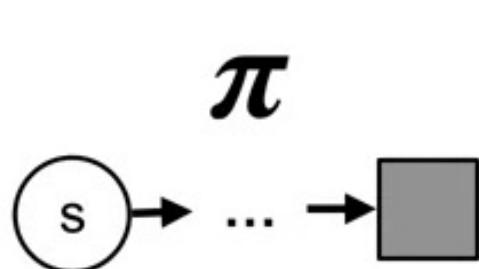
(Monte Carlo Methods vs Bandits)



Sample **rewards**

$$\frac{3 + 5 + 0 + 1}{4} = 1.5$$

Estimated value



Sample **returns**

$$\frac{3 + 5 + 0 + 1}{4} = 1.5$$

Estimated value

Prediction: (First-visit) MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

We need to generate an estimation for each state!

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

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

Prediction: (First-visit) MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

We generate episode/experience

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

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

Prediction: (First-visit) MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

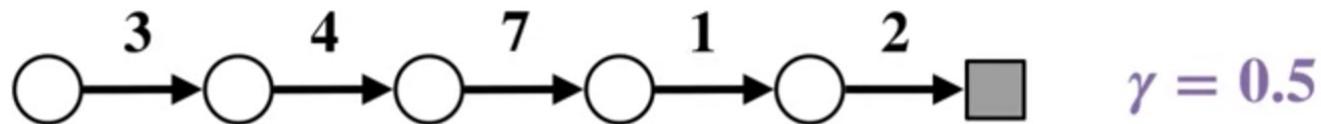
Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

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

We update the return for each of the ‘visited’ states by going backwards from the final time stamp in the episode... why?

Prediction: (First-visit) MC prediction – Computing returns efficiently



10 '+' operations

$$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 R_5 = 7$$

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 = 8$$

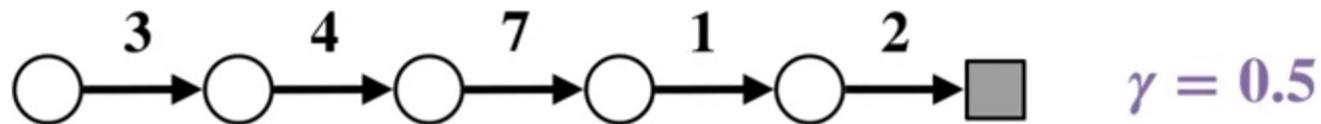
$$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 = 8$$

$$G_3 = R_4 + \gamma R_5 = 2$$

$$G_4 = R_5 = 2$$

$$G_5 = 0$$

Prediction: (First-visit) MC prediction – Computing returns efficiently



10 '+' operations

$$G_0 = R_1 + \gamma G_1$$

$$G_1 = R_2 + \gamma G_2$$

$$G_2 = R_3 + \gamma G_3$$

$$G_3 = R_4 + \gamma G_4$$

$$G_4 = R_5 + \gamma G_5$$

$$G_5 = 0$$

5 '+' operations

$$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 R_5 = 7$$

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 = 8$$

$$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 = 8$$

$$G_3 = R_4 + \gamma R_5 = 2$$

$$G_4 = R_5 = 2$$

$$G_5 = 0$$

Prediction: (First-visit) MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

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

Pay attention: we are just updating
the states that have been visited!

Prediction: (First-visit) MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

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

A more efficient approach is to use incremental updates (we have already seen this)

Incremental Update

$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]$

Prediction: First-visit MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

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

Prediction: Every-visit MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} .

Append G to $Returns(S_t)$

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

Prediction: First-visit vs Every Visit

In the student MDP (with $\gamma = 1$), we draw some samples: following the depicted policy π

- [#1] C1(-2)C2(-2)C3(-2)Pass(+10)Sleep
- [#2] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)Sleep
- [#3] C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)C3(-2)Pass(+10)Sleep
- [#4] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)Sleep

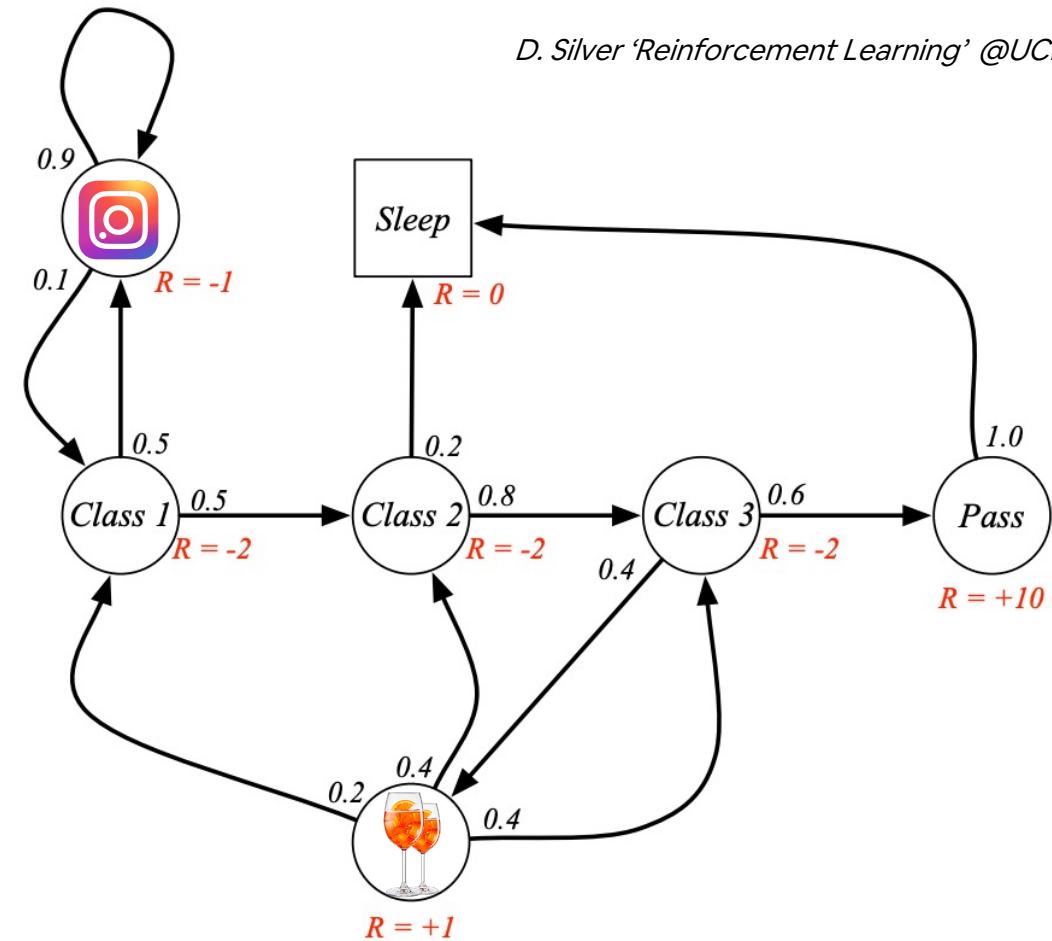
Let's estimate the value function for state C1 with MC methods:

Every-visit estimation $v_\pi = -7.625 = 0.125(4-8-4+1-20-16-11-7)$:

[episode #1] +4 [episode #2] -8, -4

[episode #3] +1 [episode #4] -20, -16, -11, -7

D. Silver 'Reinforcement Learning' @UCL



Prediction: First-visit vs Every Visit

In the student MDP (with $\gamma = 1$), we draw some **samples**: following the depicted policy π

- [#1] C1(-2)C2(-2)C3(-2)Pass(+10)Sleep
- [#2] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)Sleep
- [#3] C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)C3(-2)Pass(+10)Sleep
- [#4] C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C1(-2)IG(-1)IG(-1)C1(-2)C2(-2)C3(-2)Spritz(+1)C2(-2)Sleep

Let's estimate the value function for state C1 with MC methods:

Every-visit estimation $v_{\pi} = -7.625 = 0.125(4-8-4+1-20-16-11-7)$:

[episode #1] +4 [episode #2] -8, -4

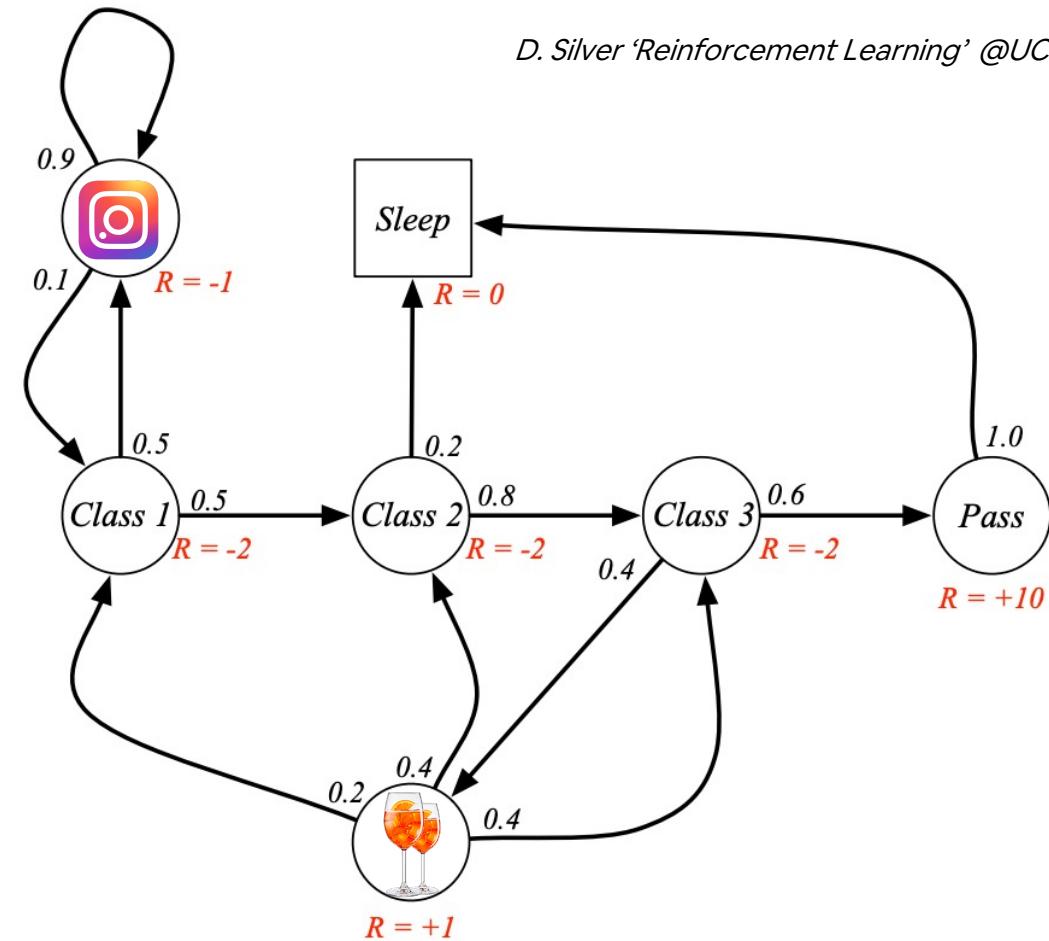
[episode #3] +1 [episode #4] -20, -16, -11, -7

First-visit estimation $v_{\pi} = -5.75 = 0.25(4-8+1-20)$:

[return #1] +4 [return #2] -8

[return #3] +1 [return #4] -20

D. Silver 'Reinforcement Learning' @UCL



Prediction: Every-visit MC prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

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

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots,$

$G \leftarrow \gamma G + R_{t+1}$

~~Unless S_t appears in S_0, S_1, \dots, S_{t-1} .~~

Append G to $Returns(S_t)$

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

Every-visit MC don't consider the check for S_t having occurred earlier in the episode.

We will focus on first-visit version, but later in the course every-visit version will be more relevant

Prediction: MC prediction - Blackjack



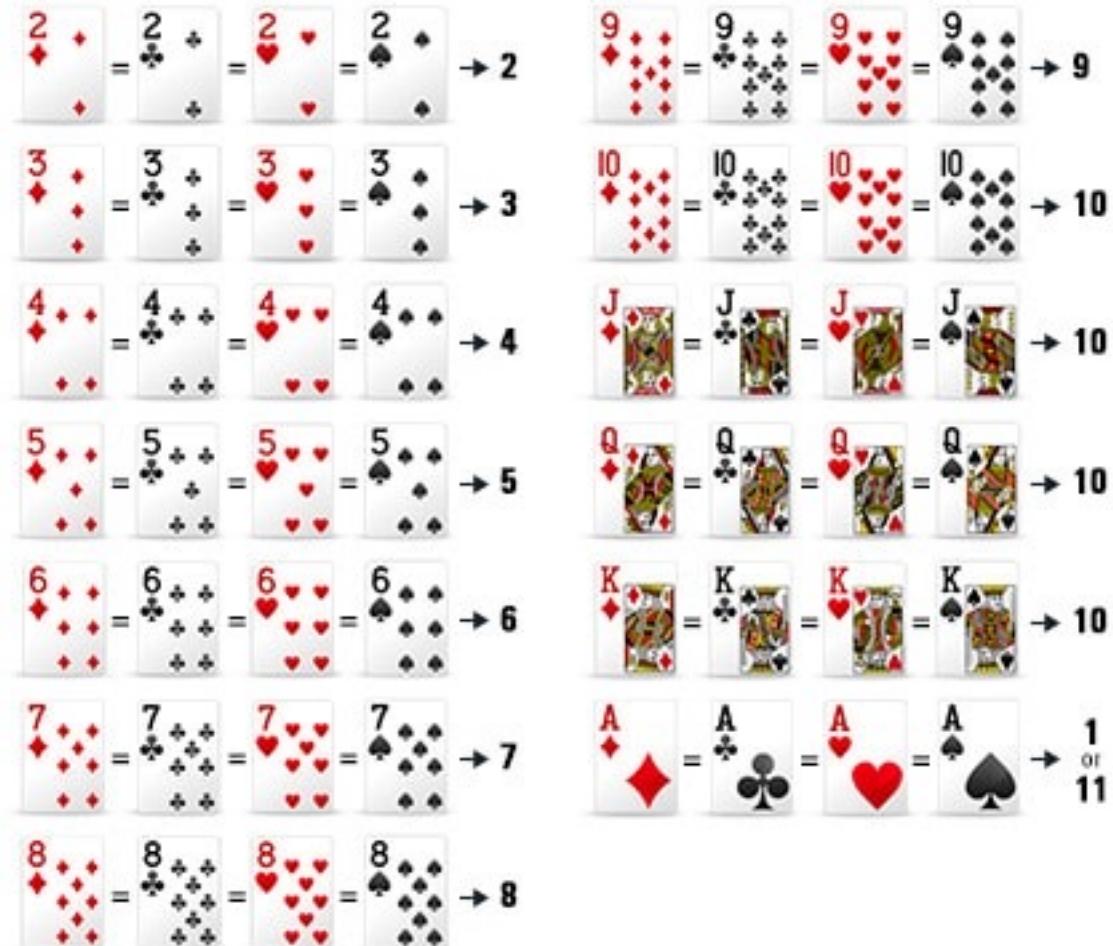
Prediction: MC prediction - Blackjack



Pay attention: we are going to use Blackjack for the third Lab!

Prediction: MC prediction - Blackjack

- Card game played against a dealer
- Played with classic 52 deck
- Goal: collect cards so that their sum is as large as possible without exceeding 21



Prediction: MC prediction - Blackjack

- Game (episode) begins with 2 cards to both the player and the dealer
- One of the card of the dealer is faced down
- If the player starts with 21, he/she immediately win (unless the dealer has also 21, then it is a draw)



Prediction: MC prediction - Blackjack

- Game (episode) begins with 2 cards to both the player and the dealer
- One of the card of the dealer is faced down
- If the player starts with 21, he/she immediately win (unless the dealer has also 21, then it is a draw)
- Without 21 the player can request cards (**hit**): one card at each request is given



Prediction: MC prediction - Blackjack

- Game (episode) begins with 2 cards to both the player and the dealer
- One of the card of the dealer is faced down
- If the player starts with 21, he/she immediately win (unless the dealer has also 21, then it is a draw)
- Without 21 the player can request cards (**hit**): one card at each request is given
- If 21 is passed, the player lose



Prediction: MC prediction - Blackjack

- If the player stops requesting card (**stick**) then it is the dealer turn
- In our simplified case, the dealer has a fixed strategy: he sticks on any sum of 17 or greater, hits otherwise



Prediction: MC prediction - Blackjack

- If the player stops requesting card (**stick**) then it is the dealer turn
- In our simplified case, the dealer has a fixed strategy: he sticks on any sum of 17 or greater, hits otherwise
- We also consider a deck with replacement (no counting cards ☺)
- Formalization as an episodic task with rewards +1/-1/0 for winning/losing/drawing
- Undiscounted MDP



Prediction: MC prediction - Blackjack

- If the player stops requesting card (**stick**) then it is the dealer turn
- In our simplified case, the dealer has a fixed strategy: he sticks on any sum of 17 or greater, hits otherwise
- We also consider a deck with replacement (no counting cards ☺)
- Formalization as an episodic task with rewards +1/-1/0 for winning/losing/drawing
- Undiscounted MDP

Which can be the states?
Which can be the actions?



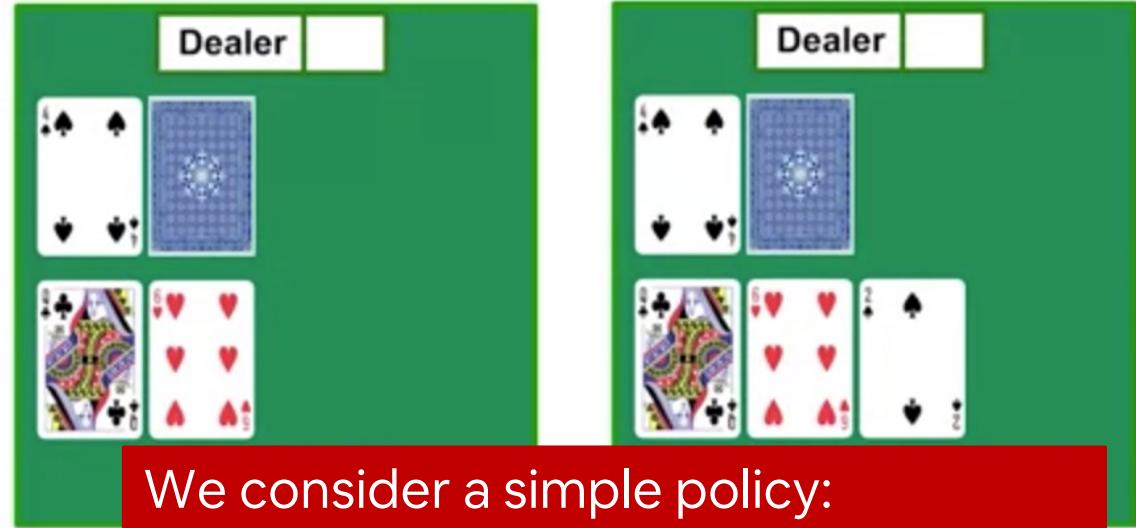
Prediction: MC prediction - Blackjack

- If the player stops requesting card (**stick**) then it is the dealer turn
- In our simplified case, the dealer has a fixed strategy: he sticks on any sum of 17 or greater, hits otherwise
- We also consider a deck with replacement (no counting cards ☺)
- Formalization as an episodic task with rewards +1/-1/0 for winning/losing/drawing
- Undiscounted MDP, with 2 actions (hit and stick)
- For the player (agent) this is a problem with 200 states, depending on
 1. Having or not a ‘usable’ ace (can be considered both as 1 or 11);
 2. The player current sum (12-21);
 3. The dealer’s shown card (Ace-10)



Prediction: MC prediction - Blackjack

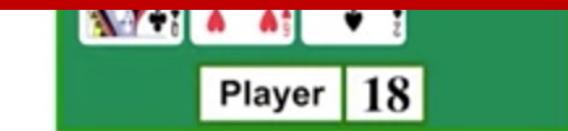
- If the player stops requesting card (**stick**) then it is the dealer turn
- In our simplified case, the dealer has a fixed strategy: he sticks on any sum of 17 or greater, hits otherwise
- We also consider a deck with replacement (no counting cards ☺)
- Formalization as an episodic task with rewards +1/-1/0 for winning/losing/drawing
- Undiscounted MDP, with 2 actions (hit and stick)
- For the player (agent) this is a problem with 200 states, depending on
 1. Having or not a ‘usable’ ace (can be considered both as 1 or 11);
 2. The player current sum (12-21);
 3. The dealer’s shown card (Ace-10)



We consider a simple policy:

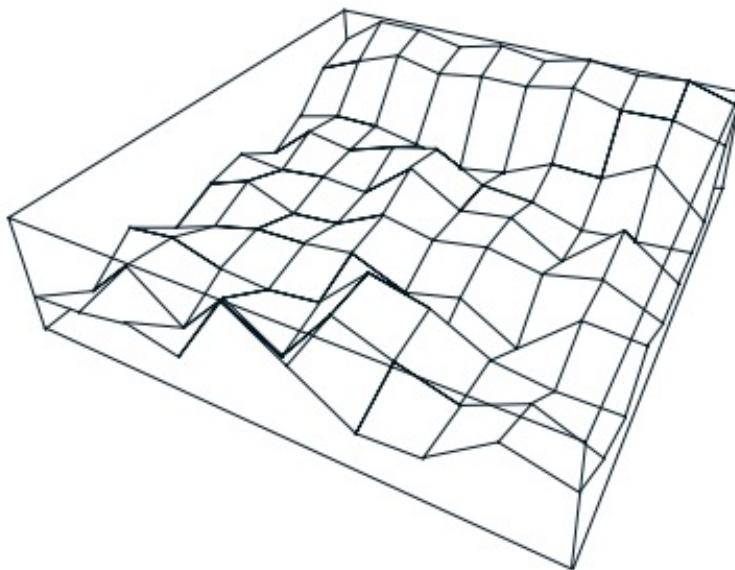
- The players stick if he/she has 20/21
- Hit otherwise

How good is this strategy? Let's play the game and compute with MC the value functions

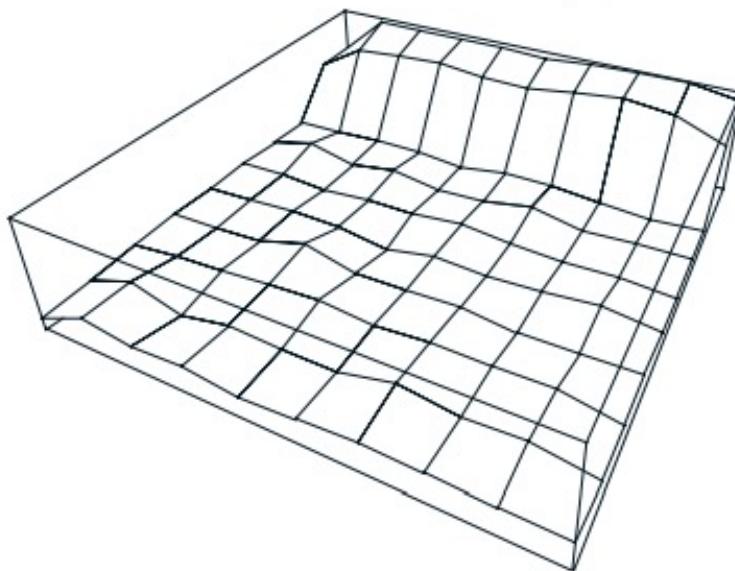


After 10,000 episodes

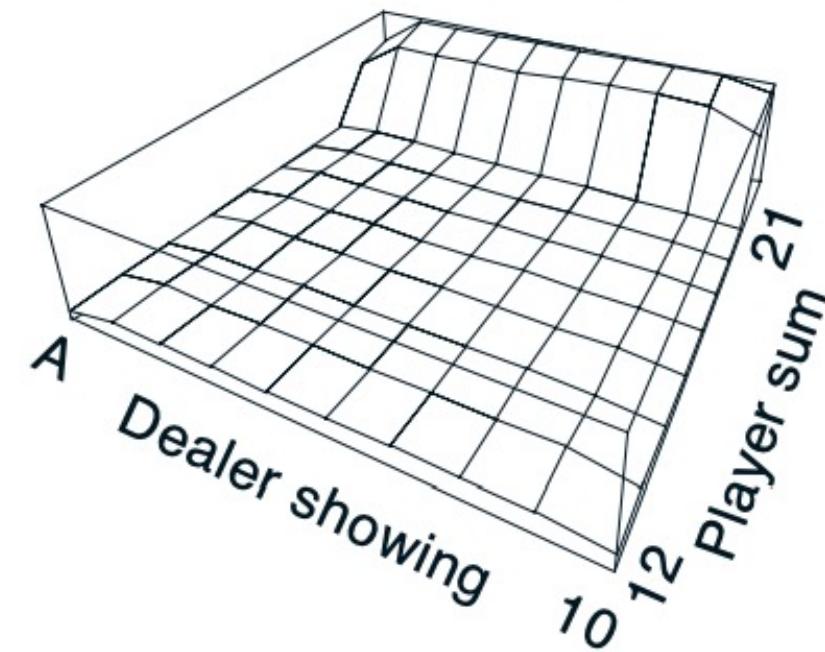
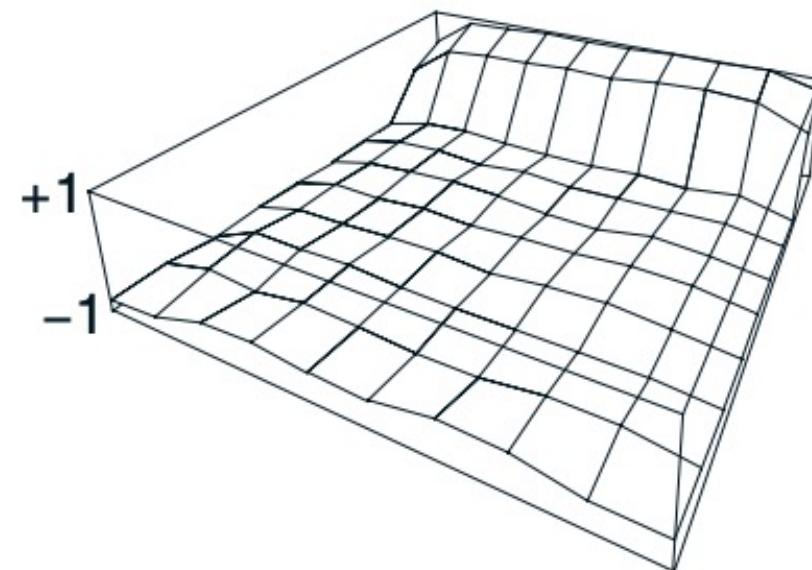
Usable
ace



No
usable
ace



After 500,000 episodes



MC learning: final comments

- The same algorithm can be used for estimating **q-value** functions
- MC learns directly from experience: there's **no need to keep a large model of the environment.**
- MC methods can **estimate the value of an individual state independently of the values of any other states.** (In dynamic programming, the value of each state depends on the values of other states!)
- The **computation** needed to update the value of each state along the way doesn't depend in any way on the size of the MDP (which was the case for DP). Rather, it **depends on the length of the episode!**

Credits

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

**Thank you!
Questions?**

Lecture #06
Dynamic Programming for
MDPs & Monte Carlo

Gian Antonio Susto

