Dynamic Programming

Machine Learning

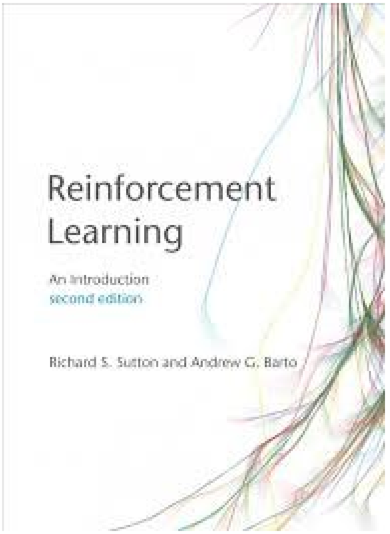
Daniele Loiacono

POLITECNICO
MILANO 1863

Outline and References

- Outline
  - Policy Evaluation
  - Policy Improvement
  - Policy Iteration
  - Generalized Policy Iteration
  - Efficiency of DP

- References
  - Reinforcement Learning: An Introduction [RL Chapter 4]
  - Fundamentals of Reinforcement Learning (Coursera)



Why Dynamic Programming?

- To solve an MDP we need to find an optimal policy
- Unfortunately we cannot use a brute-force approach:
  - $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies to evaluate
  - $|\mathcal{S}|$ linear equations to solve for each policy
- Dynamic Programming (DP) is a method that allow to solve a complex problem by breaking it down into simpler sub-problems in a **recursive** manner
- We will see how to use DP to solve an MDP thanks to the Bellman Equations

Policy Evaluation

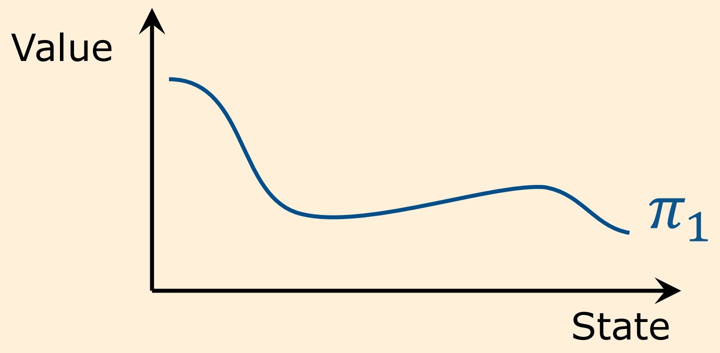
$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi}(s') \right)$$

$$\pi, r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow V_{\pi}$$

Policy Improvement

Bellman Eqs (V_{π}, V^*, Q_{π}, Q^*)

$$r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow \pi^*$$



Why Dynamic Programming?

- To solve an MDP we need to find an optimal policy
- Unfortunately we cannot use a brute-force approach:
  - $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies to evaluate
  - $|\mathcal{S}|$ linear equations to solve for each policy
- Dynamic Programming (DP) is a method that allow to solve a complex problem by breaking it down into simpler sub-problems in a **recursive** manner
- We will see how to use DP to solve an MDP thanks to the Bellman Equations

Policy Evaluation

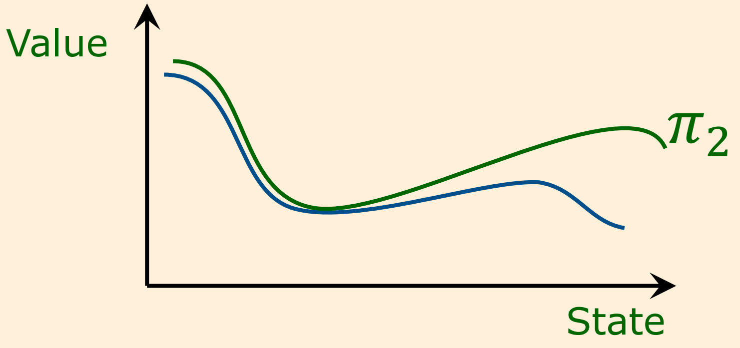
$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

$$\pi, r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow V_\pi$$

Policy Improvement

Bellman Eqs (V_π, V^*, Q_π, Q^*)

$$r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow \pi^*$$



Why Dynamic Programming?

- To solve an MDP we need to find an optimal policy
- Unfortunately we cannot use a brute-force approach:
  - $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies to evaluate
  - $|\mathcal{S}|$ linear equations to solve for each policy
- Dynamic Programming (DP) is a method that allow to solve a complex problem by breaking it down into simpler sub-problems in a **recursive** manner
- We will see how to use DP to solve an MDP thanks to the Bellman Equations

Policy Evaluation

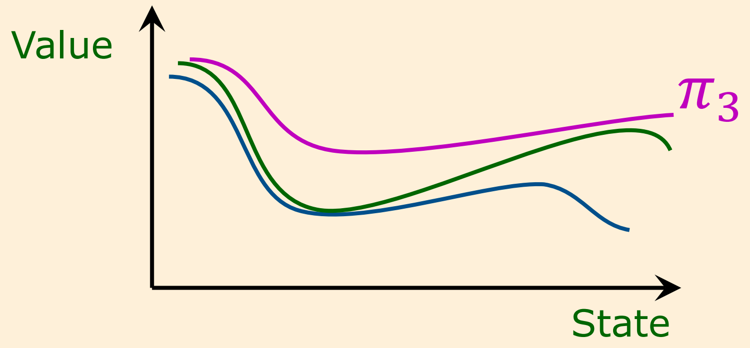
$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi}(s') \right)$$

$$\pi, r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow V_{\pi}$$

Policy Improvement

Bellman Eqs (V_{π}, V^*, Q_{π}, Q^*)

$$r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow \pi^*$$



Why Dynamic Programming?

- To solve an MDP we need to find an optimal policy
- Unfortunately we cannot use a brute-force approach:
  - $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies to evaluate
  - $|\mathcal{S}|$ linear equations to solve for each policy
- Dynamic Programming (DP) is a method that allow to solve a complex problem by breaking it down into simpler sub-problems in a **recursive** manner
- We will see how to use DP to solve an MDP thanks to the Bellman Equations

Policy Evaluation

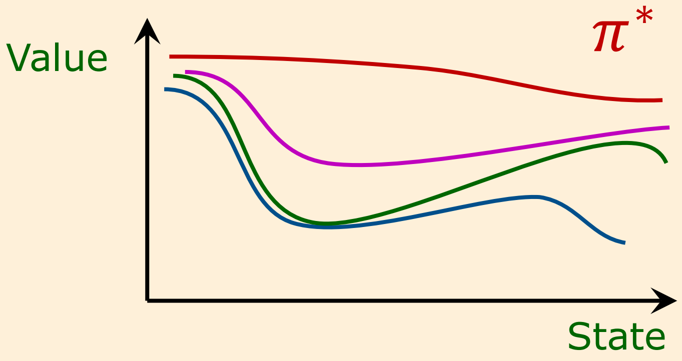
$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

$\pi, r, p, \gamma \longrightarrow$ $\boxed{\text{DP}}$ $\longrightarrow V_\pi$

Policy Improvement

Bellman Eqs (V_π, V^*, Q_π, Q^*)

$r, p, \gamma \longrightarrow$ $\boxed{\text{DP}}$ $\longrightarrow \pi^*$



Policy Evaluation

Iterative Policy Evaluation

- We search the solution of the Bellman expectation equation:

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) V_\pi(s') \right)$$

- DP solves this problem through iterative application of Bellman equation:

$$V_{k+1}(s) \leftarrow \sum_{a\in\mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) V_k(s') \right)$$

  - At each iteration k, the value-function V_k is updated for all state $s \in \mathcal{S}$

$$V_0 \to V_1 \to \cdots \to V_k \to V_{k+1} \to V_\pi$$

sweep

  - It can be proved that V_k converge to V_π as $k \to \infty$ for any V_0

Iterative Policy Evaluation (2)

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

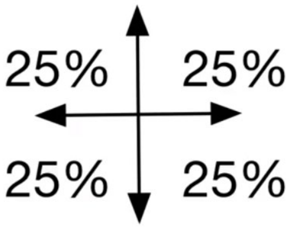
Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

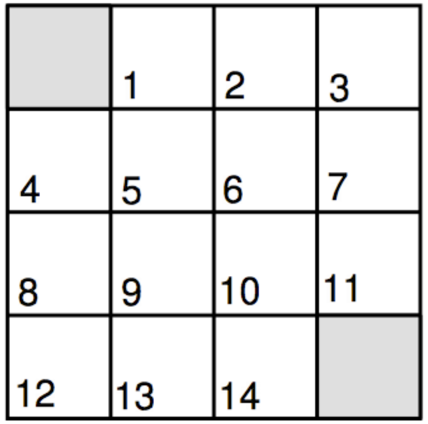
Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

«in place» update

Iterative Policy Evaluation: a Small Gridworld Example

- Let consider gridworld environment with two terminal states, where
  - $\gamma = 1$
  - $r(s,a) = -1 \quad \forall s, a$
  - $\pi(a|s)$





V_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

V_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

V_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

V_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

V_{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

V_∞

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

Policy Improvement

Policy Improvement

- Do you remember how to derive optimal policy from optimal value functions?

$$\pi^*(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\} = \arg\max_a Q^*(s,a)$$

- What happens if we act greedy with respect to non optimal value function?

$$\pi'(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right\} = \arg\max_a Q_\pi(s,a), \quad \forall s \in \mathcal{S}$$

- Is π' different from π ?
  - If not, it means that π is already the optimal policy π^* (as it satisfies the Bellman Optimality equations)
  - Otherwise, is π' better or as good as π?

Policy Improvement Theorem

- For any pair deterministic policies π' and π such that:

$$Q_\pi(s, \pi'(s)) \geq Q_\pi(s, \pi(s)), \quad \forall s \in \mathcal{S}$$

then π' is better or as good as π

$$\pi' \geq \pi$$

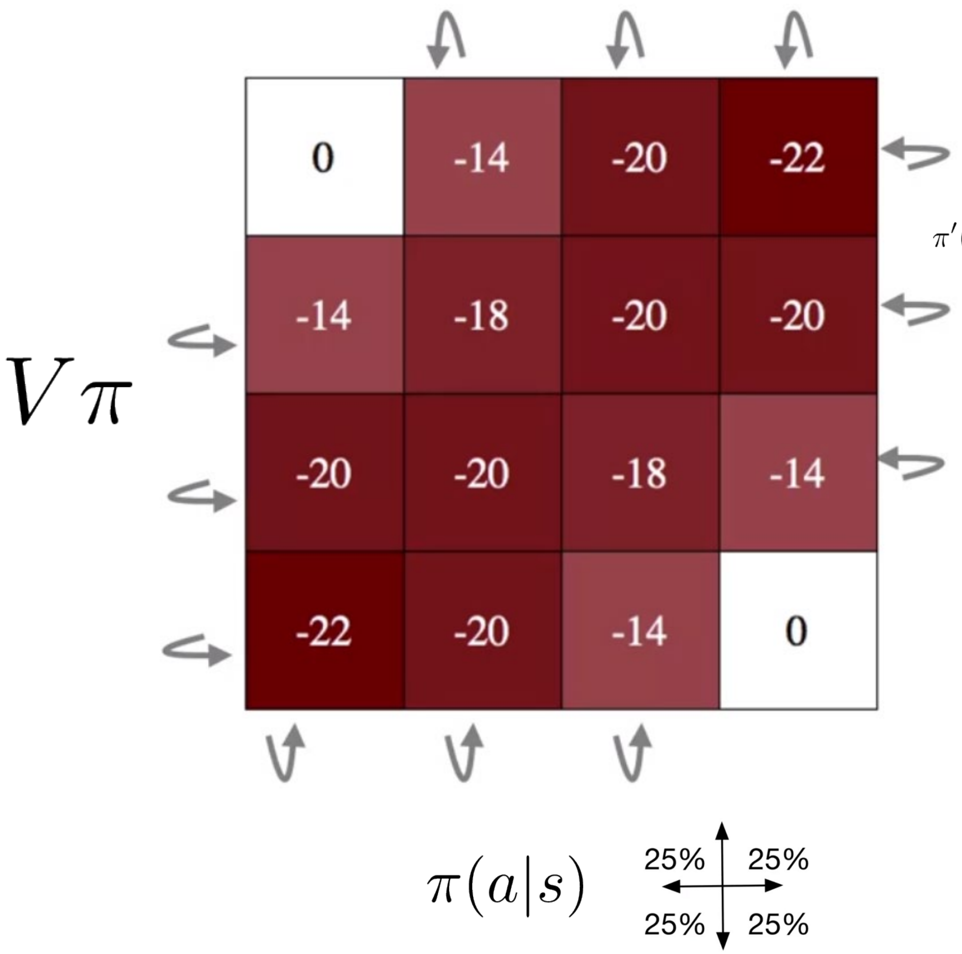
  - If $\exists s \in \mathcal{S}$ s.t. $Q_\pi(s, \pi'(s)) > Q_\pi(s, \pi(s))$ then $\pi' > \pi$

- Proof

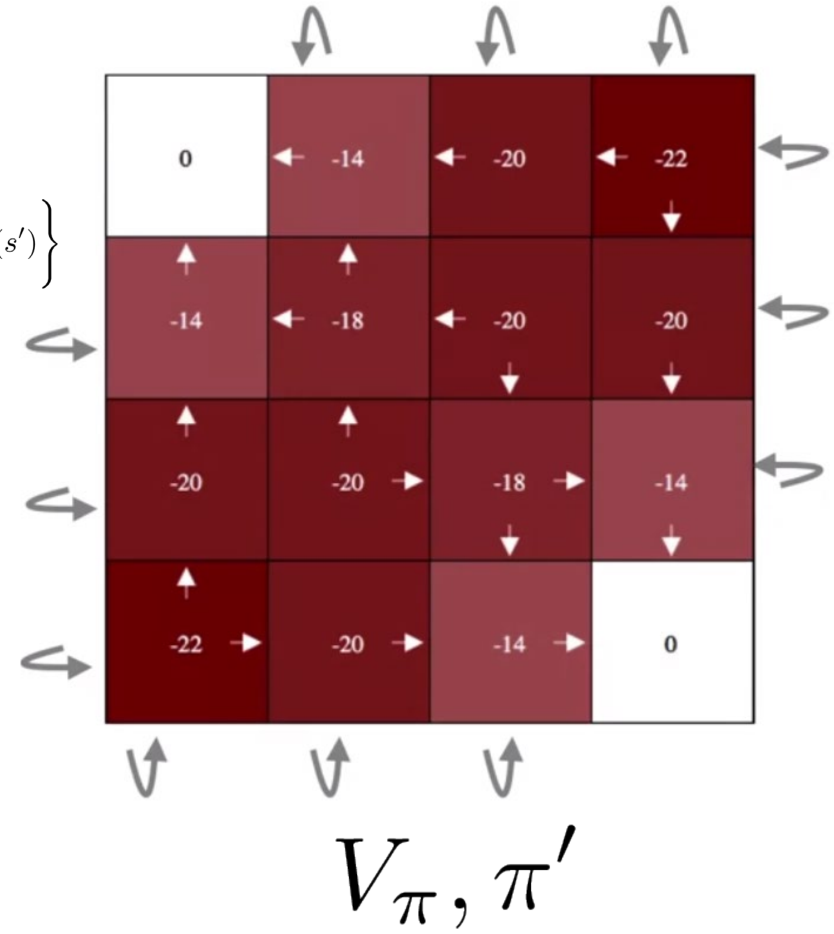
$$\begin{aligned}
V_\pi(s) &\leq Q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_\pi(S_{t+2},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

Policy Improvement: a Small Gridworld Example

- Let's go back to the value function we found iteratively for a random policy in the Small Gridworld example



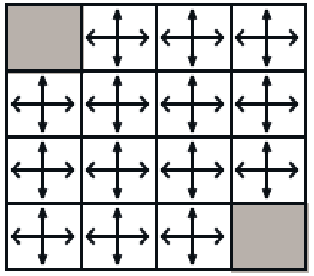
$$\pi'(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right\}$$



Policy Improvement: a Small Gridworld Example

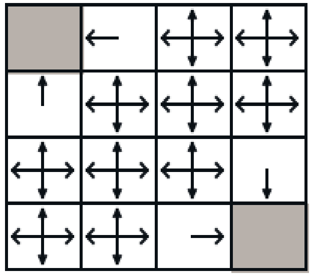
V_0

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



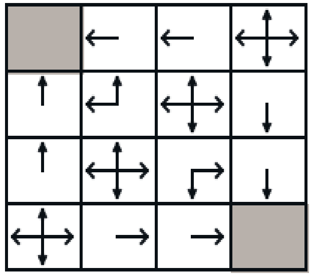
V_1

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



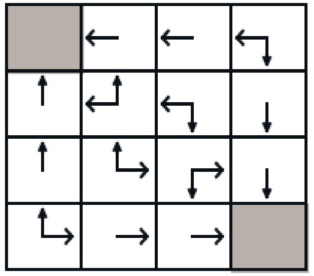
V_2

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



V_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



π^*

Policy Iteration

Policy Iteration

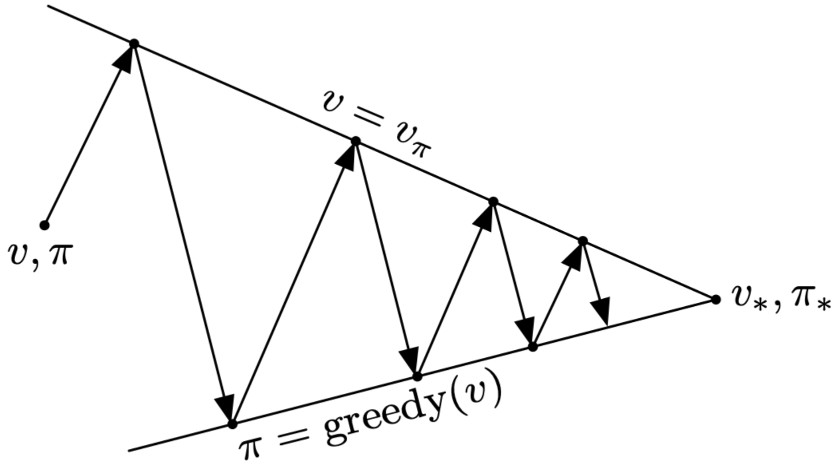
- We can exploit the policy improvement theorem to find the optimal policy:

Policy Evaluation

Policy Improvement

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

$\pi_1 > \pi_0$



Policy Iteration (2)

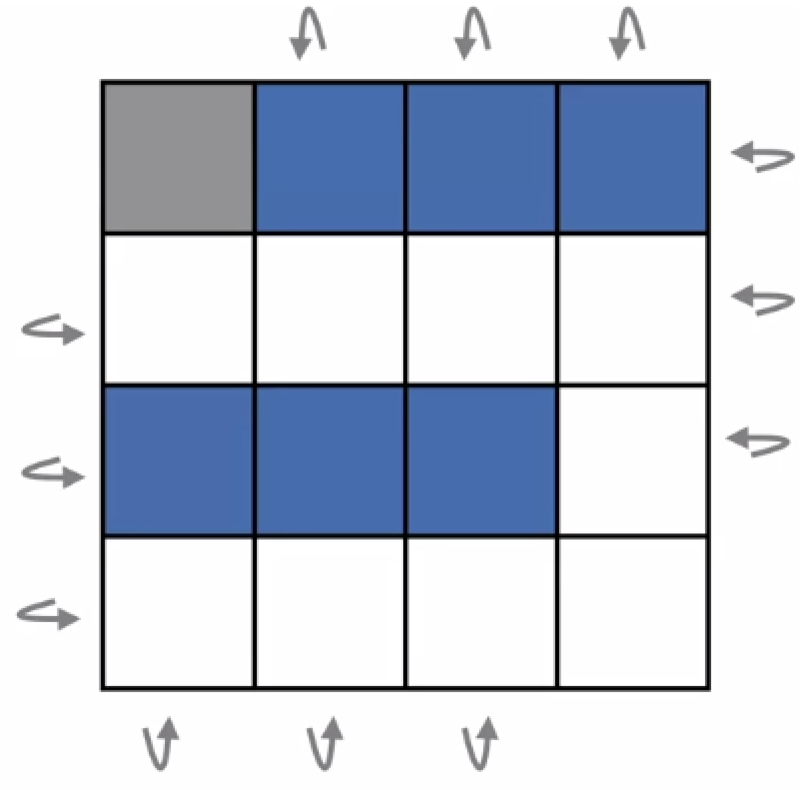
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   $\quad$ 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

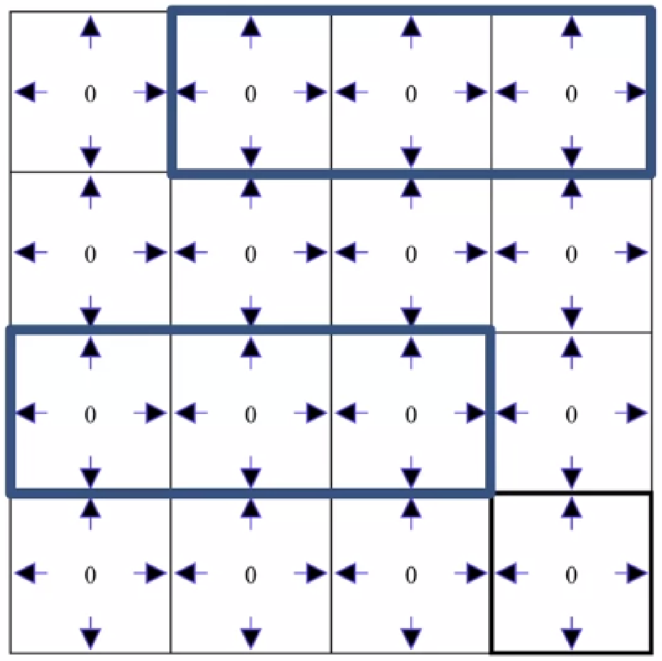
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

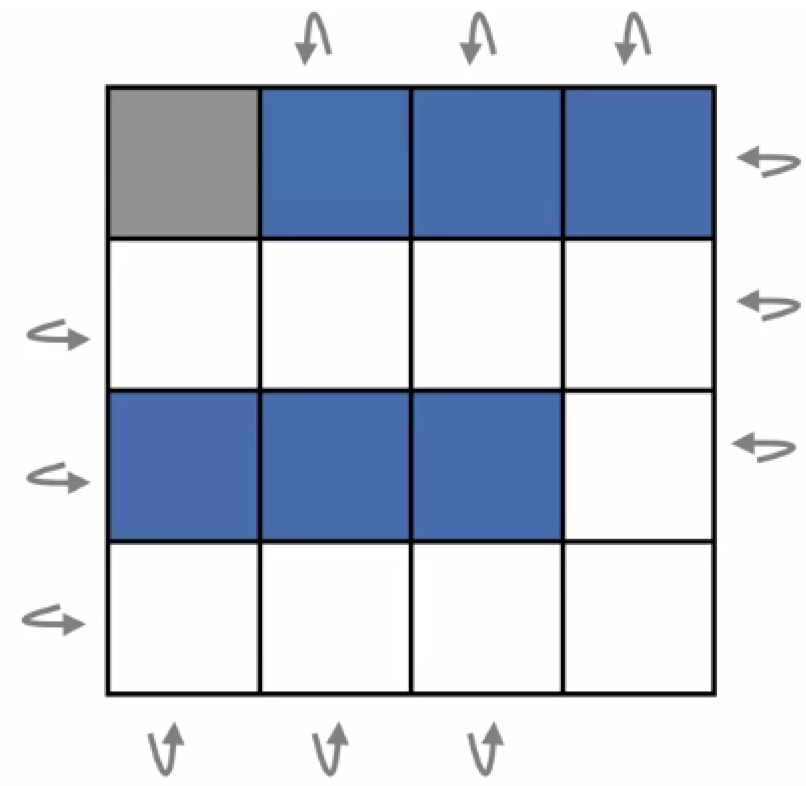
V_π, π'



Evaluation

Improvement

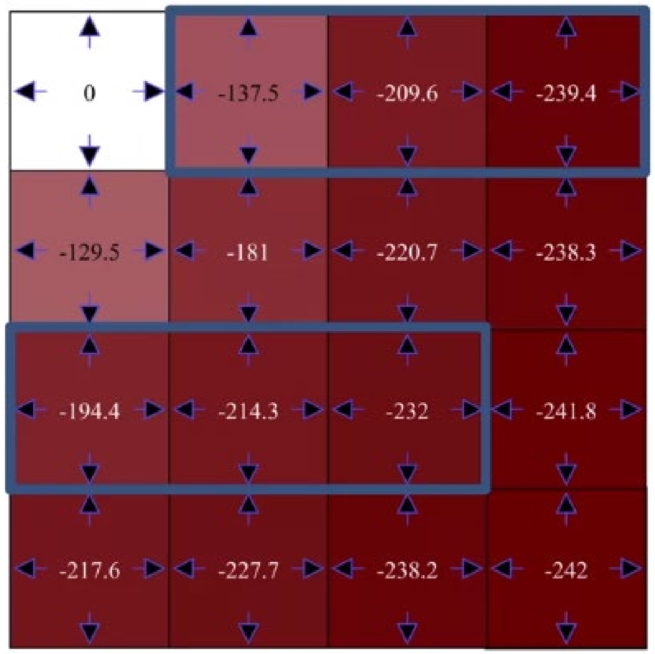
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

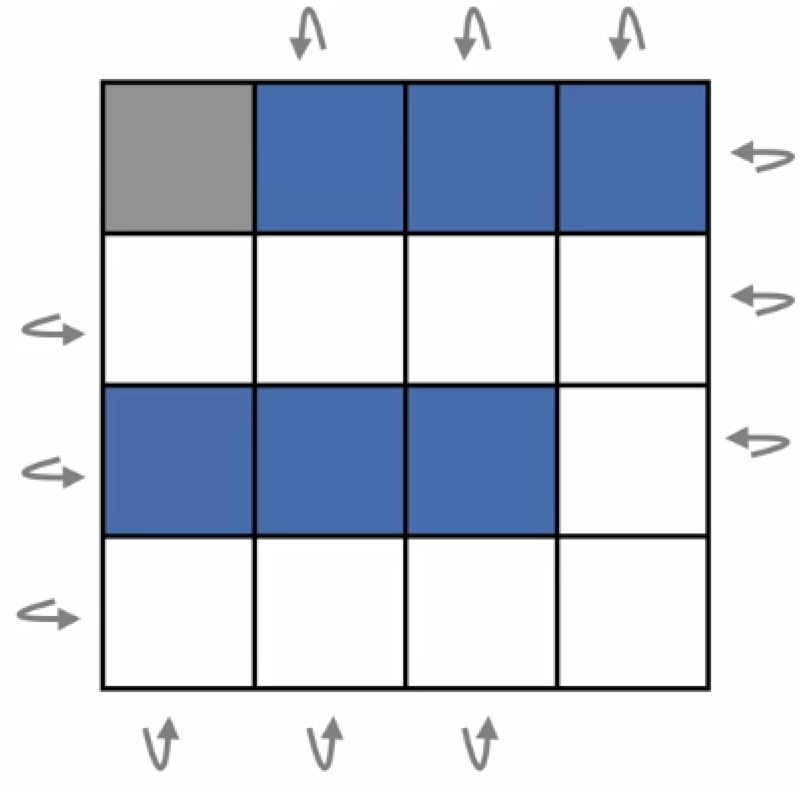
V_π, π'



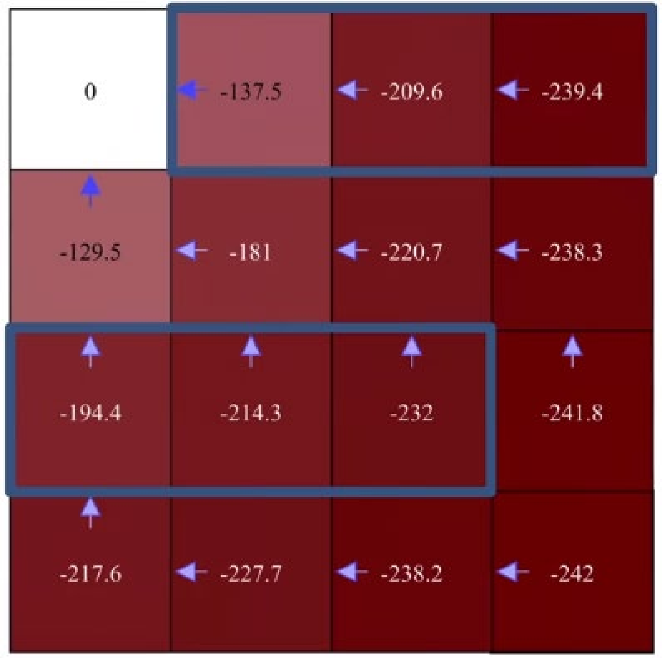
Evaluation

Improvement

Policy Iteration: Example



V_π, π'



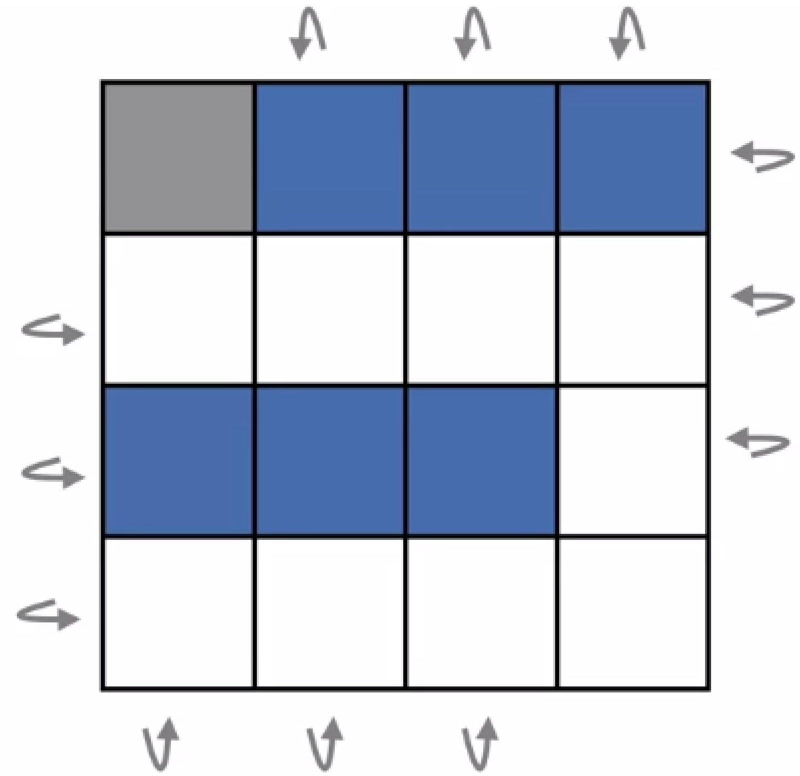
$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

Evaluation

Improvement

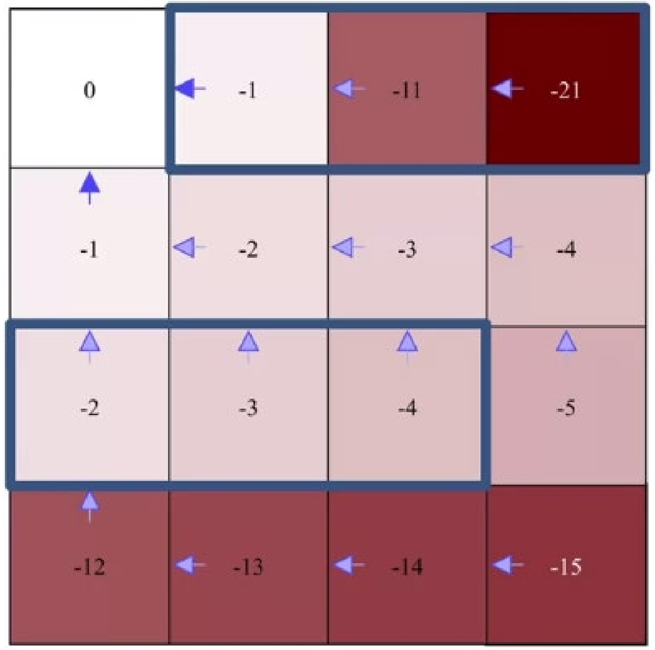
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

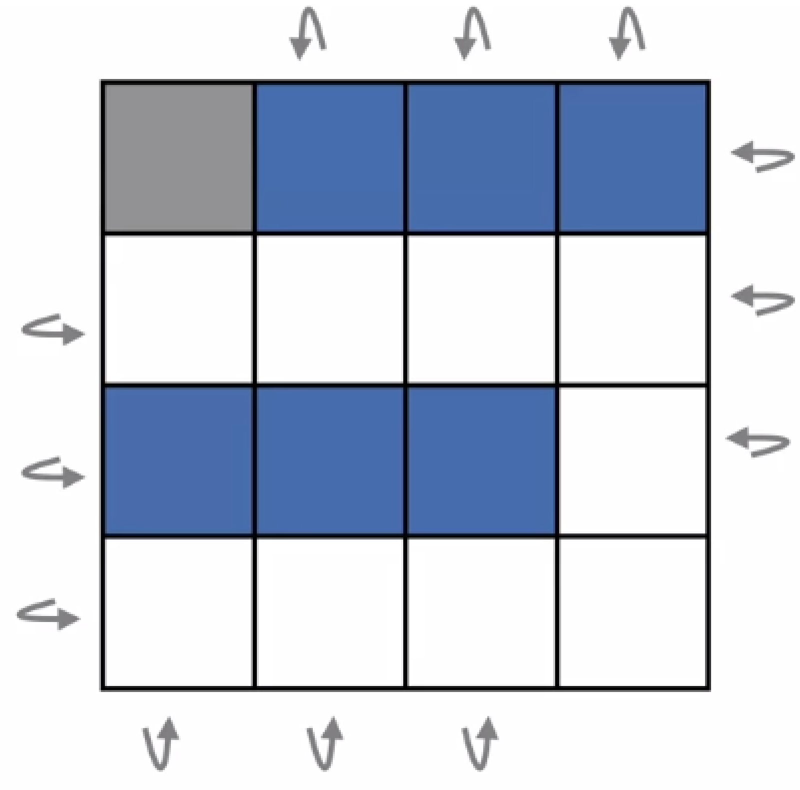
$$V_\pi, \pi'$$



Evaluation

Improvement

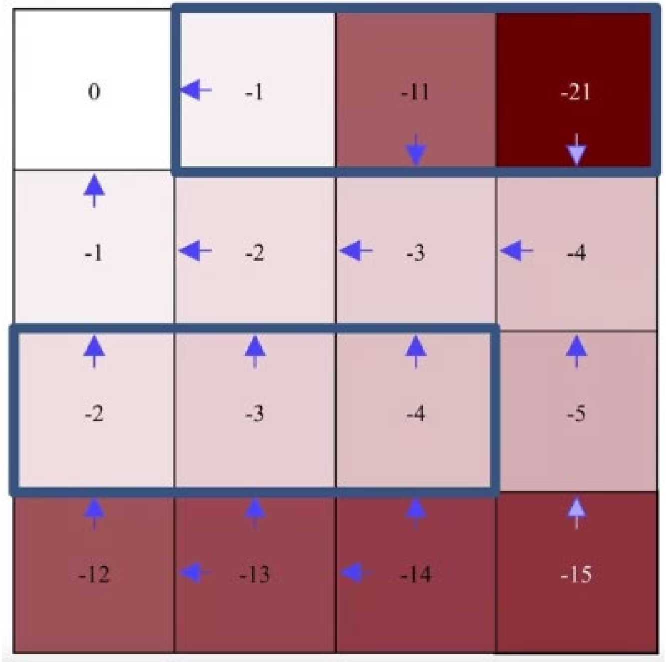
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$\gamma = 1$

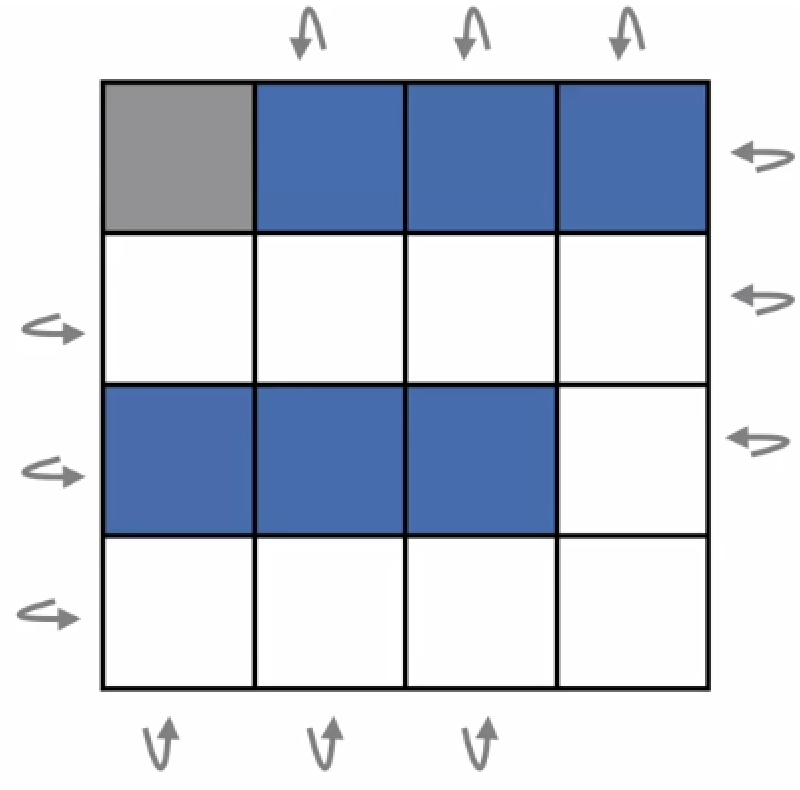
V_π, π'



Evaluation

Improvement

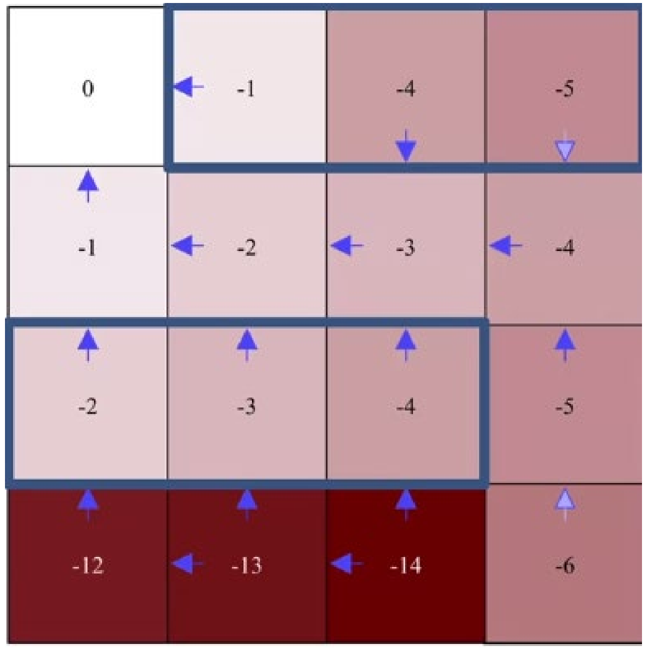
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

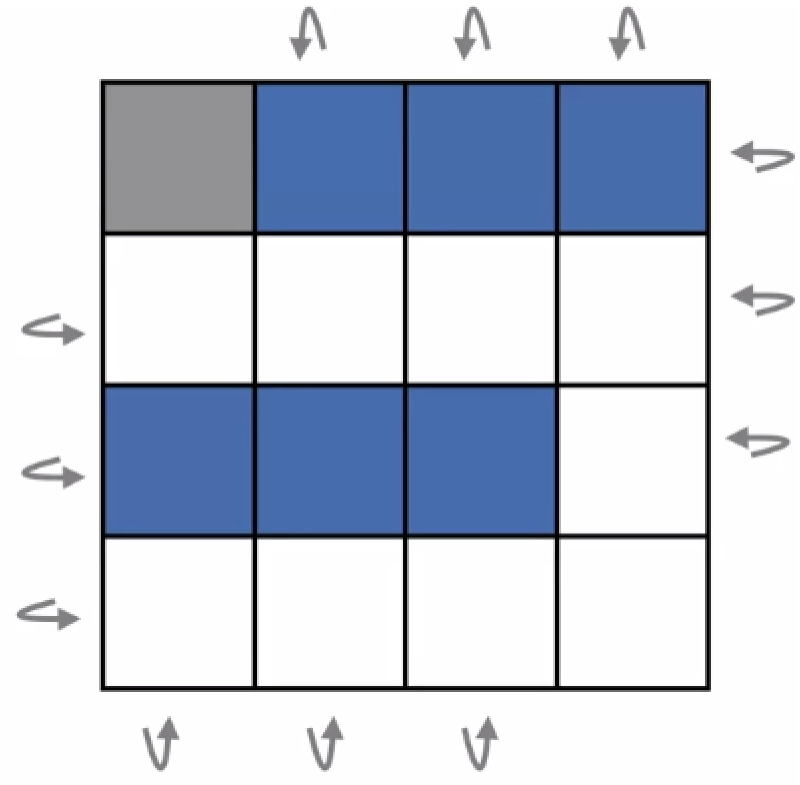
V_π, π'



Evaluation

Improvement

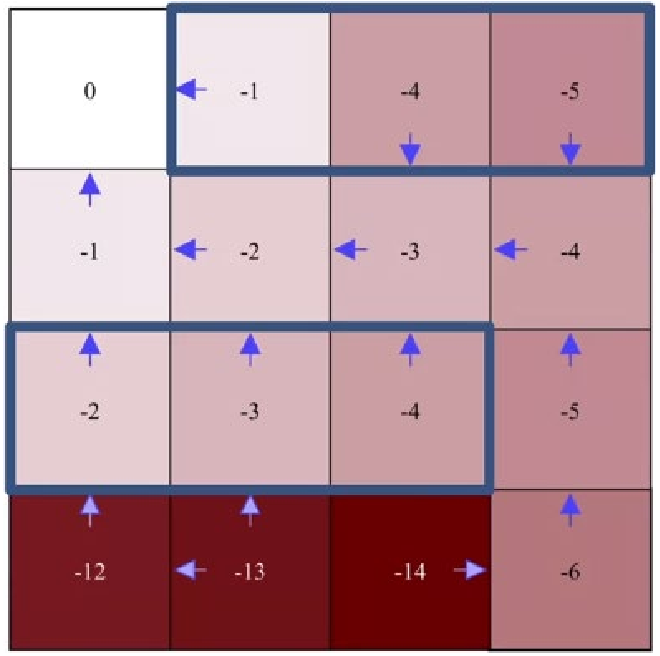
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

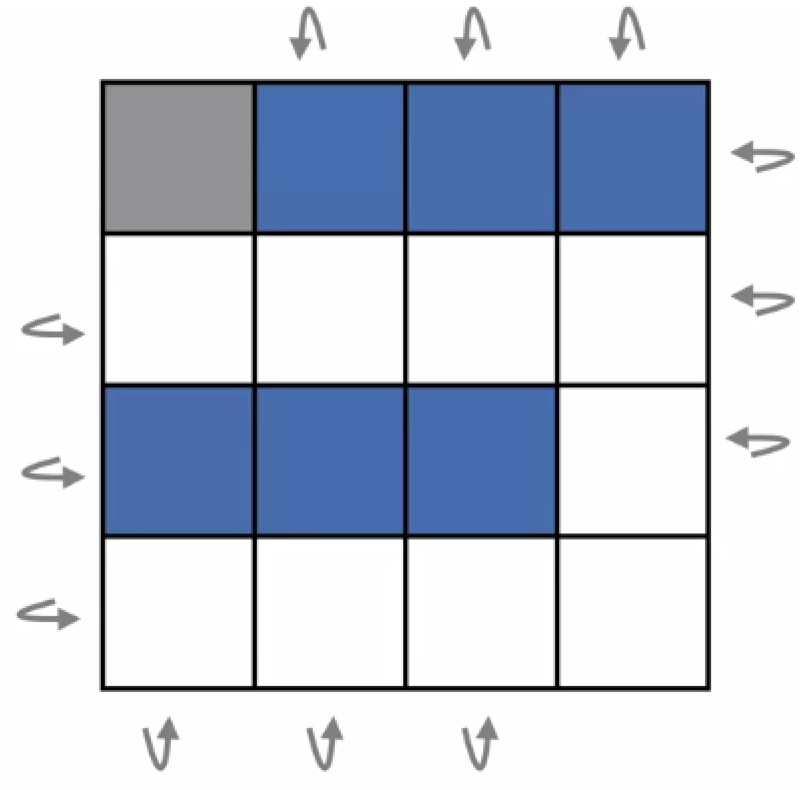
$$V_\pi, \pi'$$



Evaluation

Improvement

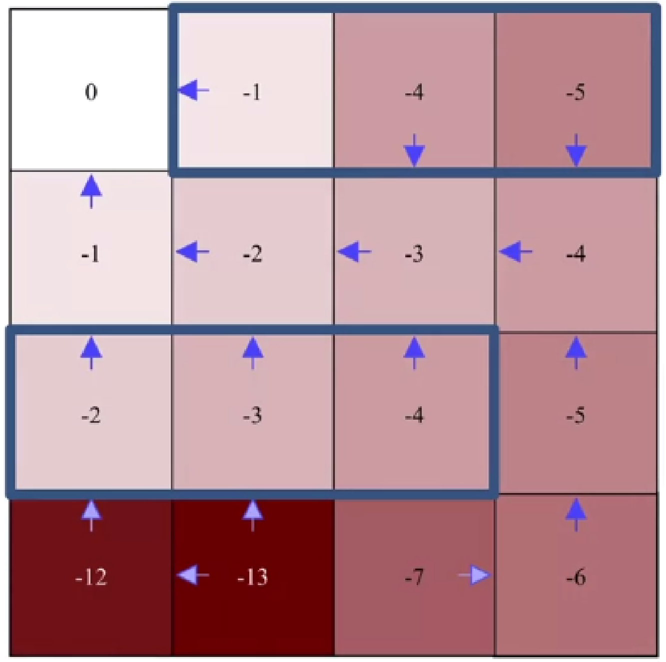
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

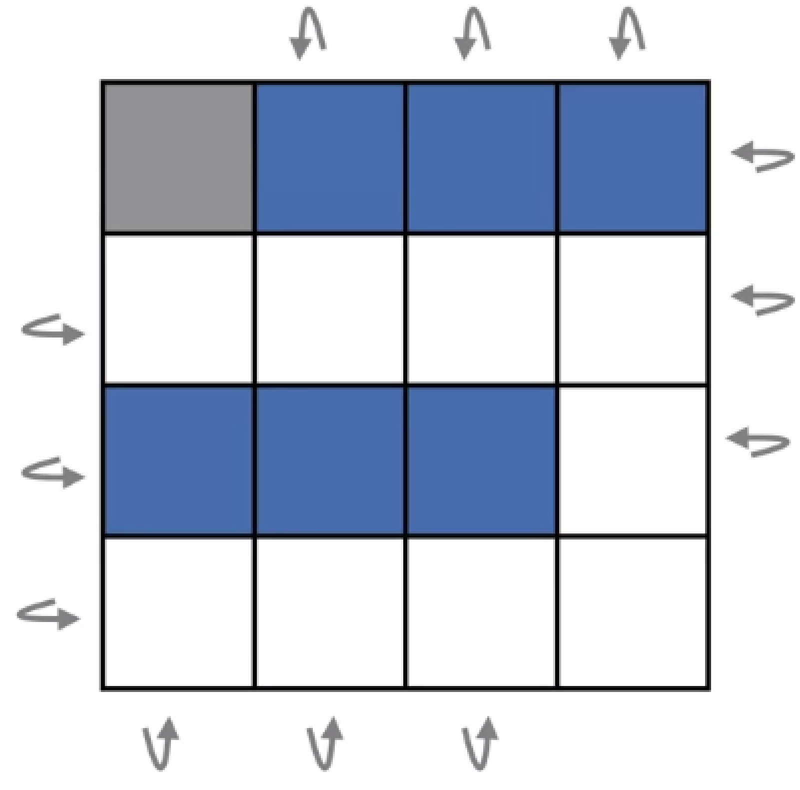
V_π, π'



Evaluation

Improvement

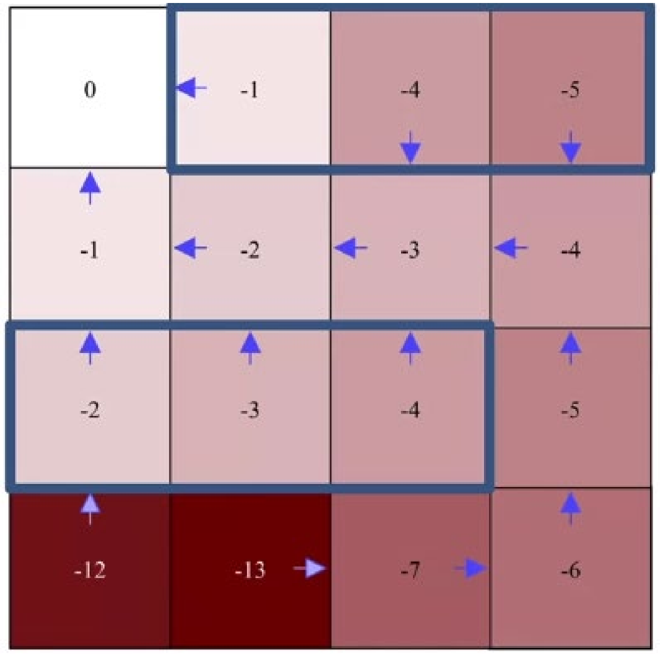
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

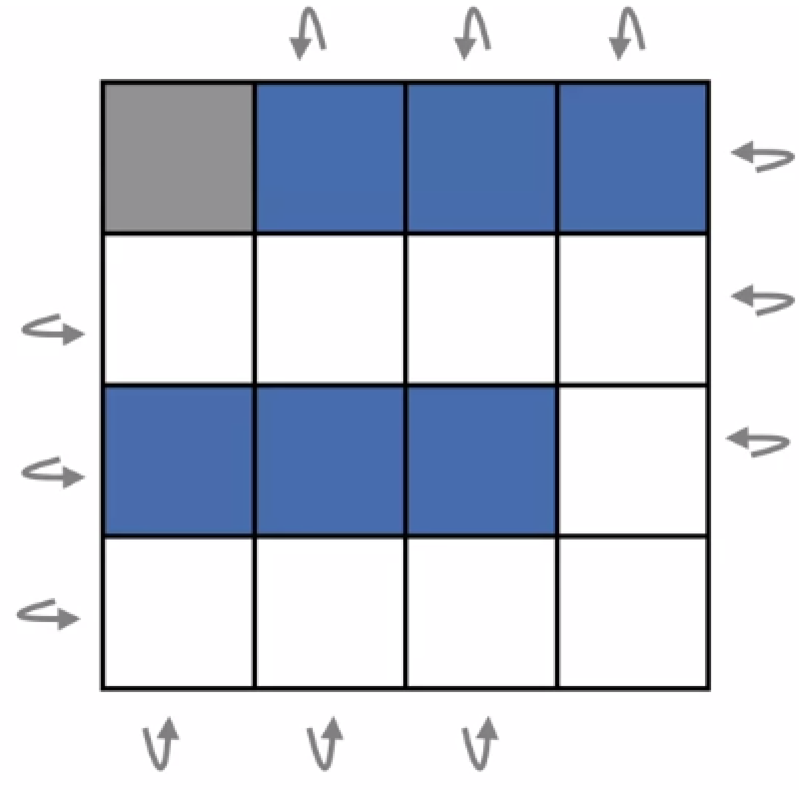
V_π, π'



Evaluation

Improvement

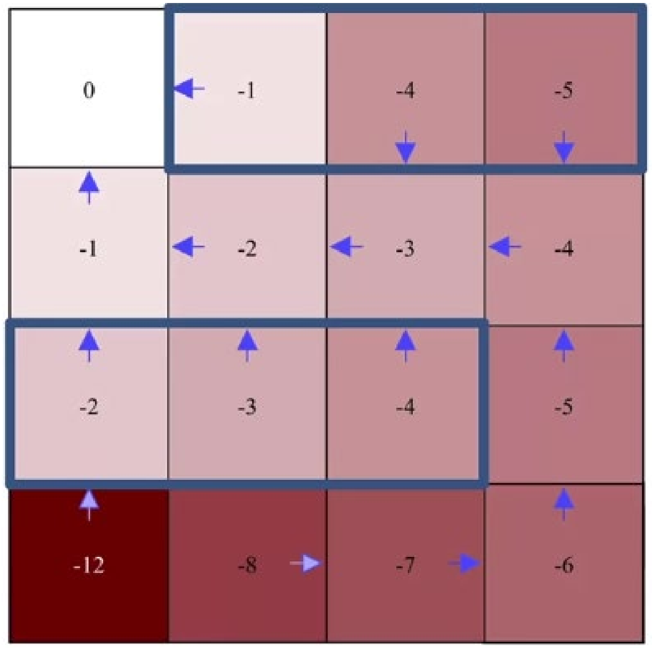
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

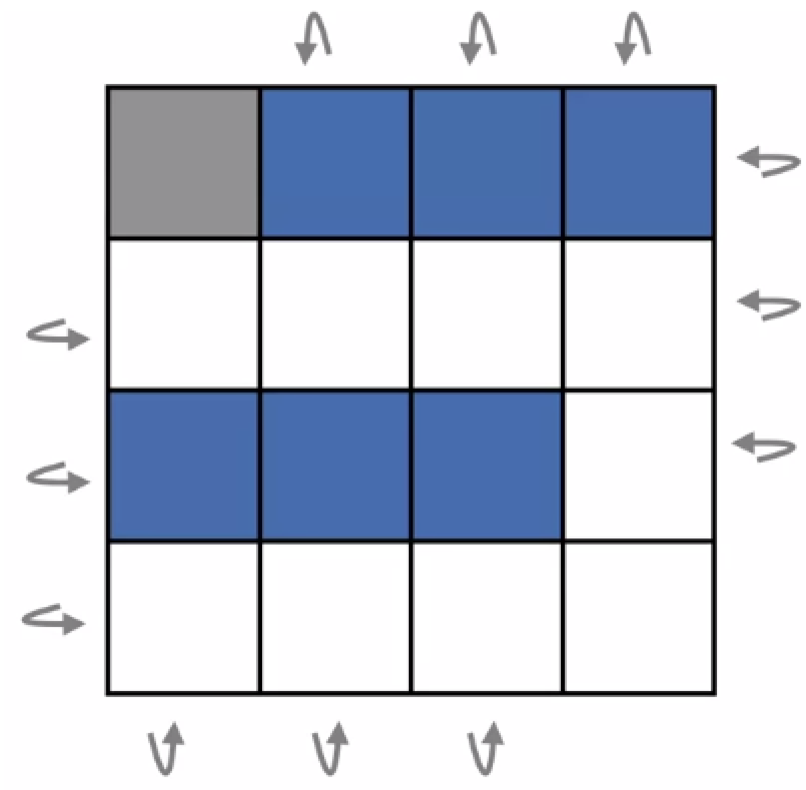
V_π, π'



Evaluation

Improvement

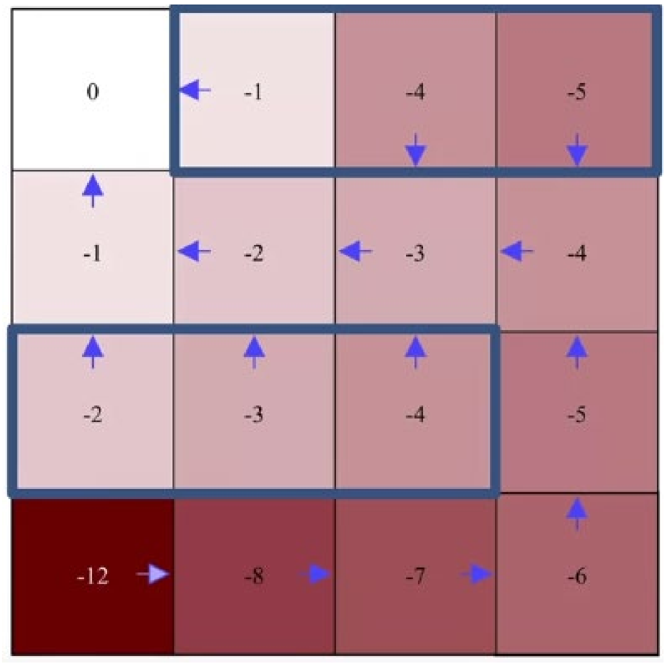
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

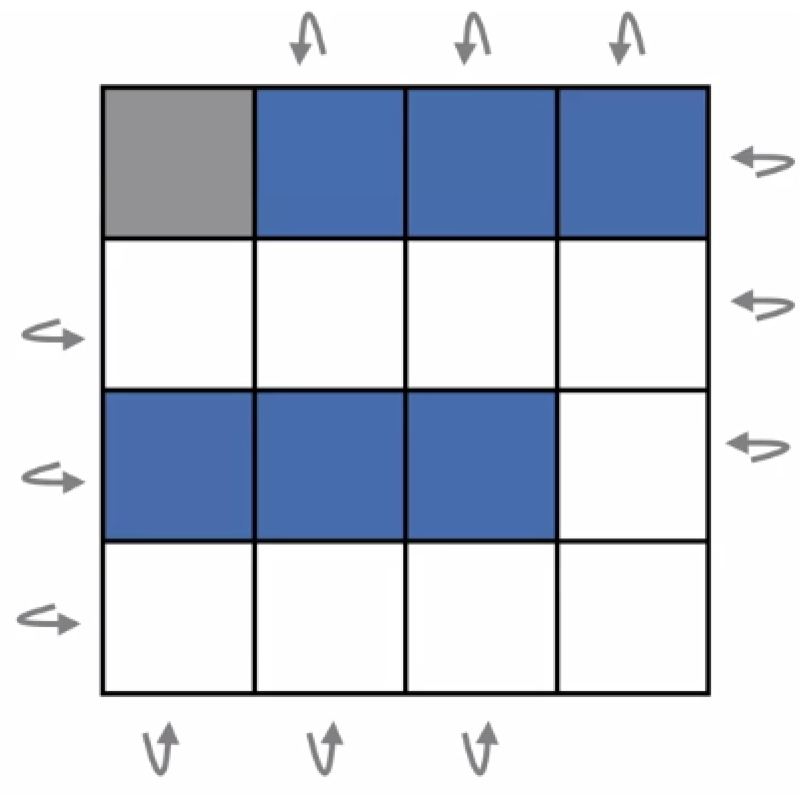
V_π, π'



Evaluation

Improvement

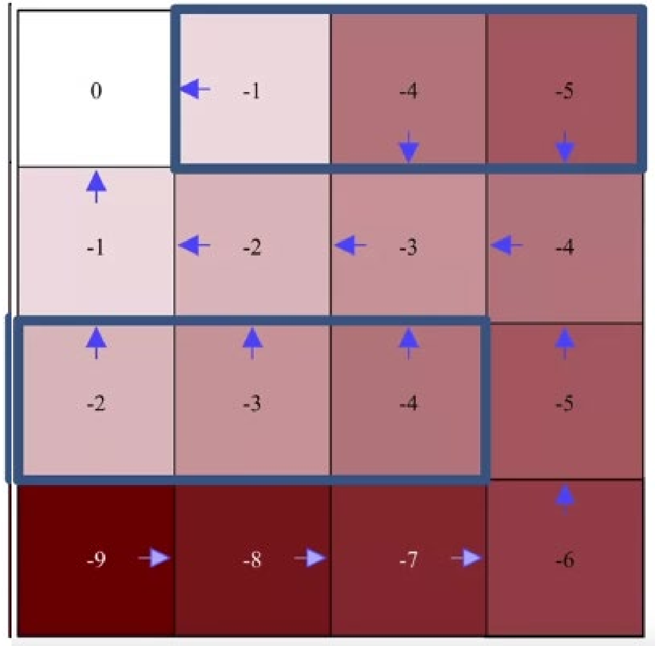
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

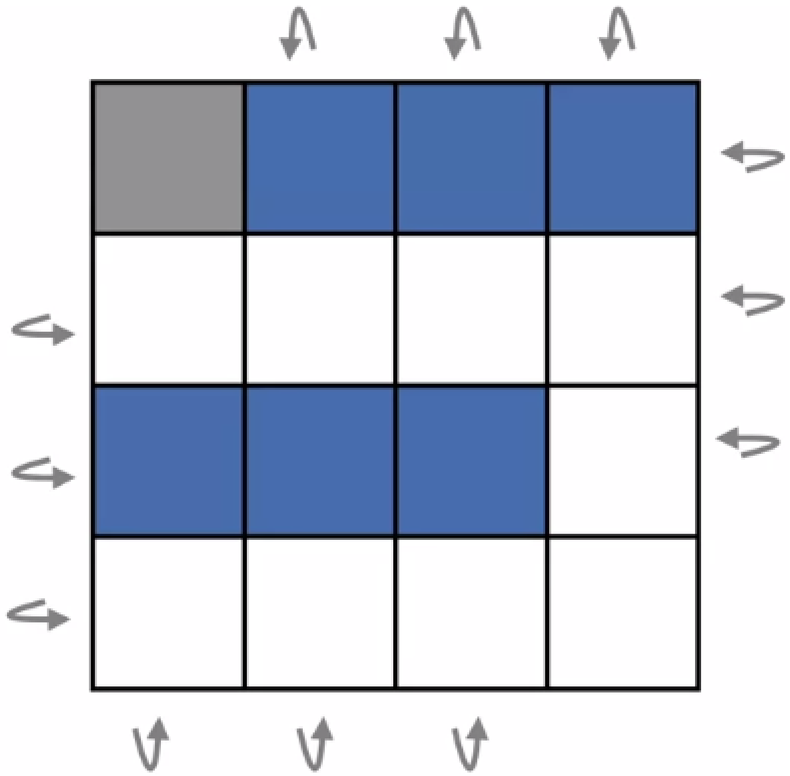
$$V_\pi, \pi'$$



Evaluation

Improvement

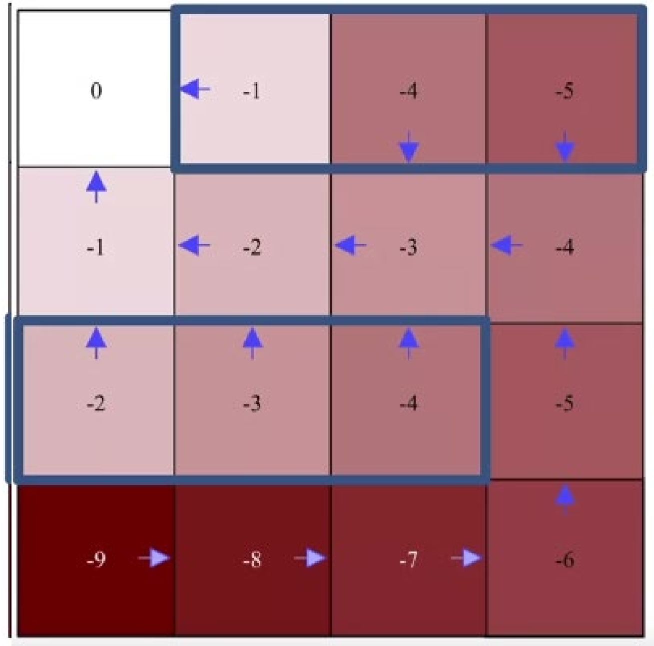
Policy Iteration: Example



$$R_t = \begin{cases} -10 & \text{in blue states} \\ -1 & \text{in other states} \end{cases}$$

$$\gamma = 1$$

V_π, π'

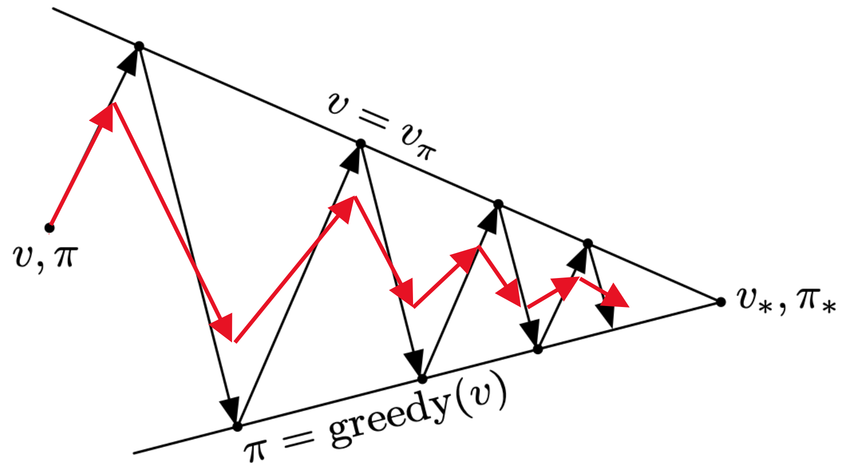


Optimal policy and value function

Generalized Policy Iteration

Generalized Policy Iteration

- ❑ Policy iteration alternates complete policy evaluation and improvement up to the convergence:



- ❑ Policy iteration framework allows also to find the optimal policy interleaving partial evaluation and improvement steps
- ❑ In particular, Value Iteration is one of the most popular GPI method

Value Iteration

- In the policy evaluation step, only a single sweep of updates is performed:

$$\pi'(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right\}, \forall s \in \mathcal{S}$$

$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi'(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_k(s') \right), \forall s \in \mathcal{S}$$

- Combining them, we simply need to iterate the update of the value function using the Bellman optimality equation:

$$V_{k+1}(s) \leftarrow \max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_k(s') \right], \forall s \in \mathcal{S}$$

- It can be proved that $\lim_{k \to \infty} V_k = V^*$

Value Iteration (2)

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 $V(terminal) = 0$

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

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

Efficiency of DP

Asynchronous Dynamic Programming

- All the DP methods described so far require exhaustive sweeps of the entire state set.
- Asynchronous DP does not use sweeps:
  - Pick a state at random
  - Apply the appropriate backup
  - Repeat until convergence criterion is met
- Can you select states to backup intelligently?
  - An agent's experience can act as a guide.

Efficiency of DP

- To find an optimal policy is **polynomial** in the number of states and actions
  - Value Iteration: $O(|\mathcal{S}|^2|\mathcal{A}|)$
  - Policy Iteration: iterative evaluation $O\left(\frac{|\mathcal{S}|^2 \log(1/\epsilon)}{\log(1/\gamma)}\right)$, improvement $O\left(\frac{|A|}{1-\gamma}\log\left(\frac{|S|}{1-\gamma}\right)\right)$
- Unfortunately, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (**curse of dimensionality**)
- In practice, classical DP can be applied to problems with a few millions of states
- Asynchronous DP can be applied to larger problems, and is appropriate for parallel computation
  - But it is **easy** to come up with MDPs for which DP methods are not practical
- **Linear programming** approaches can be also used instead of DP but they do not typically scale well on larger problems