

# Chapter 4: Dynamic Programming

---

Objectives of this chapter:

- ❑ Overview of a collection of classical solution methods for MDPs known as dynamic programming (DP)
- ❑ Show how DP can be used to compute value functions, and hence, optimal policies
- ❑ Discuss efficiency and utility of DP

# Policy Evaluation

---

**Policy Evaluation:** for a given policy  $\pi$ , compute the state-value function  $V^\pi$

Recall: **State - value function for policy  $\pi$  :**

$$V^\pi(s) = E_\pi \{R_t \mid s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}$$

**Bellman equation for  $V^\pi$  :**


$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$$

— a system of  $|S|$  simultaneous linear equations

# Iterative Methods

---

$$V_0 \rightarrow V_1 \rightarrow \cdots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \cdots \rightarrow V^\pi$$

a “sweep” 

A sweep consists of applying a **backup operation** to each state.

A **full policy evaluation backup**:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')]$$

# Iterative Policy Evaluation

---

Input  $\pi$ , the policy to be evaluated

Initialize  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

    For each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

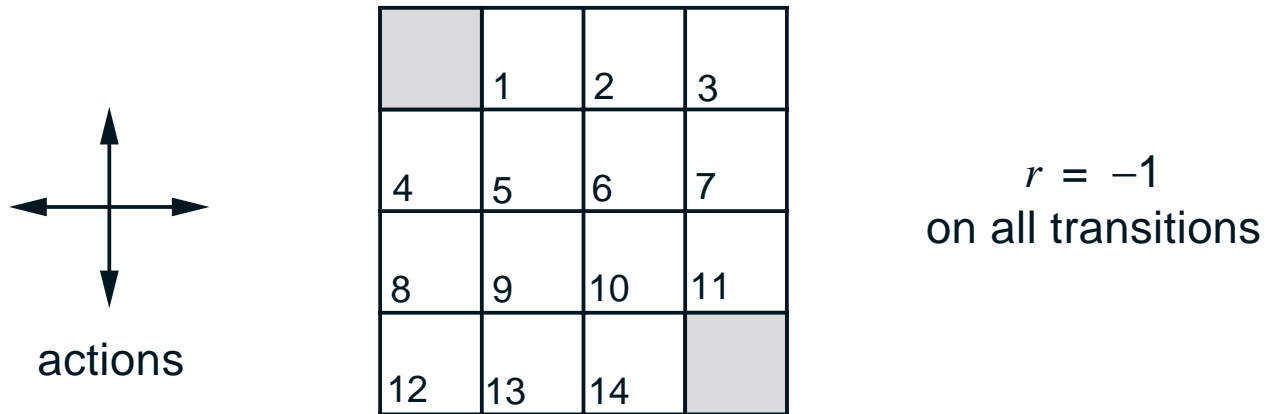
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until  $\Delta < \theta$  (a small positive number)

Output  $V \approx V^\pi$

# A Small Gridworld

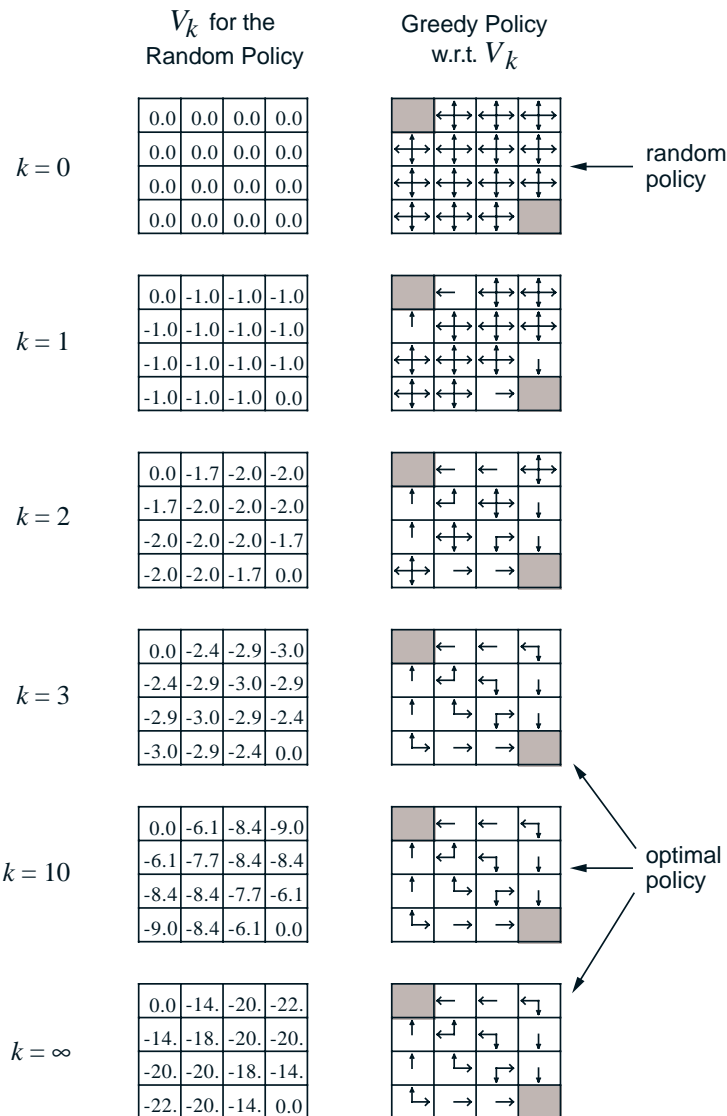
---



- ❑ An undiscounted episodic task
- ❑ Nonterminal states: 1, 2, . . . , 14;
- ❑ One terminal state (shown twice as shaded squares)
- ❑ Actions that would take agent off the grid leave state unchanged
- ❑ Reward is  $-1$  until the terminal state is reached

# Iterative Policy Eval for the Small Gridworld

$\pi =$  random (uniform) action choices



# Policy Improvement

---

Suppose we have computed  $V^\pi$  for a deterministic policy  $\pi$ .

For a given state  $s$ ,  
would it be better to do an action  $a \neq \pi(s)$ ?

The value of doing  $a$  in state  $s$  is :

$$\begin{aligned} Q^\pi(s, a) &= E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a \right\} \\ &= \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right] \end{aligned}$$

It is better to switch to action  $a$  for state  $s$  if and only if

$$Q^\pi(s, a) > V^\pi(s)$$

# Policy Improvement Cont.

---

Do this for all states to get a new policy  $\pi'$  that is **greedy** with respect to  $V^\pi$  :

$$\begin{aligned}\pi'(s) &= \operatorname{argmax}_a Q^\pi(s, a) \\ &= \operatorname{argmax}_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]\end{aligned}$$

Then  $V^{\pi'} \geq V^\pi$



# Policy Improvement Cont.

---

What if  $V^{\pi'} = V^{\pi}$  ?

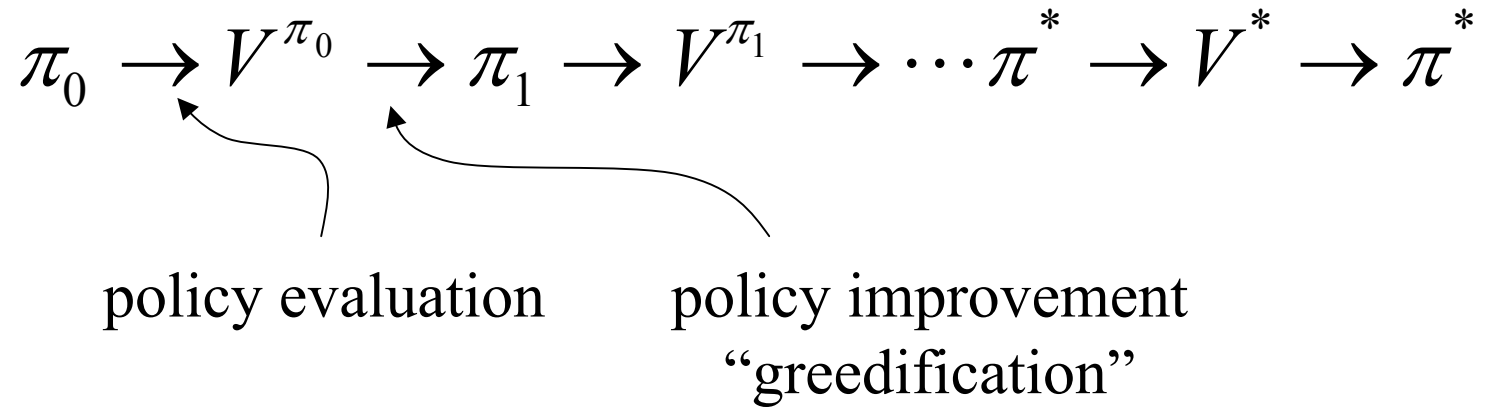
$$\text{i.e., for all } s \in S, \quad V^{\pi'}(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')] ?$$

But this is the Bellman Optimality Equation.

So  $V^{\pi'} = V^*$  and both  $\pi$  and  $\pi'$  are optimal policies.

# Policy Iteration

---



# Policy Iteration

---

## 1. Initialization

$V(s) \in \Re$  and  $\pi(s) \in \mathcal{A}(s)$  arbitrarily for all  $s \in \mathcal{S}$

## 2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

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

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} [\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$

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

until  $\Delta < \theta$  (a small positive number)

## 3. Policy Improvement

*policy-stable*  $\leftarrow$  true

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

$b \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

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

If *policy-stable*, then stop; else go to 2

# Value Iteration

---

Recall the **full policy evaluation backup**:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')]$$

Here is the **full value iteration backup**:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')]$$

# Value Iteration Cont.

---

Initialize  $V$  arbitrarily, e.g.,  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

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

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

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

until  $\Delta < \theta$  (a small positive number)

Output a deterministic policy,  $\pi$ , such that

$\pi(s) = \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

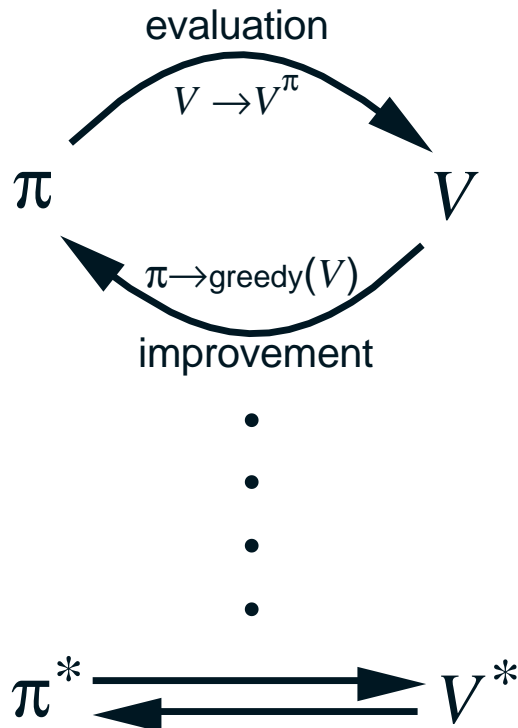
# Asynchronous DP

---

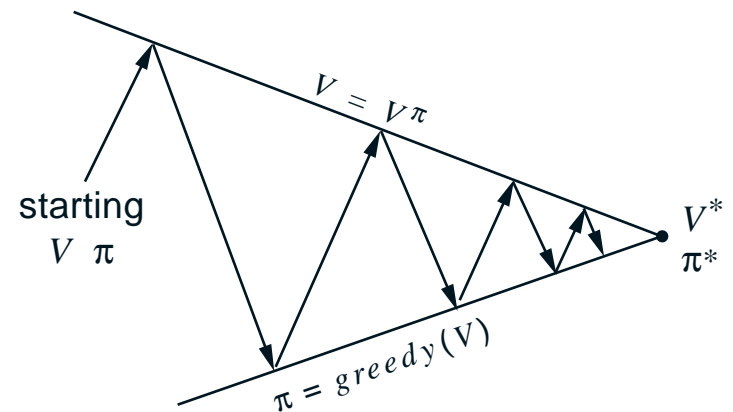
- ❑ All the DP methods described so far require exhaustive sweeps of the entire state set.
- ❑ Asynchronous DP does not use sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - Pick a state at random and apply the appropriate backup
- ❑ Still need lots of computation, but does not get locked into hopelessly long sweeps
- ❑ Can you select states to backup intelligently? YES: an agent's experience can act as a guide.

# Generalized Policy Iteration

**Generalized Policy Iteration (GPI):**  
any interaction of policy evaluation and policy improvement,  
independent of their granularity.



A geometric metaphor for convergence of GPI:



# Efficiency of DP

---

- ❑ To find an optimal policy is polynomial in the number of states...
- ❑ BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the 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 appropriate for parallel computation.
- ❑ It is surprisingly easy to come up with MDPs for which DP methods are not practical.



# Summary

---

- ❑ Policy evaluation: backups without a max
- ❑ Policy improvement: form a greedy policy, if only locally
- ❑ Policy iteration: alternate the above two processes
- ❑ Value iteration: backups with a max
- ❑ Full backups (to be contrasted later with sample backups)
- ❑ Generalized Policy Iteration (GPI)
- ❑ Asynchronous DP: a way to avoid exhaustive sweeps
- ❑ **Bootstrapping**: updating estimates based on other estimates