

# 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 (Prediction)

---

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

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

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$

Recall: **Bellman equation for  $v_\pi$**

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

—a system of  $|S|$  simultaneous equations

# Iterative Policy Evaluation (Prediction)

---

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$$

a “sweep” 

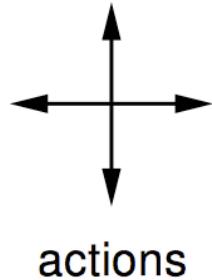
A sweep consists of applying a **backup operation** to each state.

A **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')] \quad \forall s \in \mathcal{S}$$

# A Small Gridworld Example

---



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

$R = -1$   
on all transitions

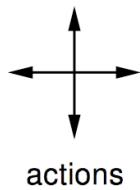
$\gamma = 1$

- 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

$V_k$  for the  
Random Policy

$\pi$  = equiprobable random action choices



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

$R = -1$   
on all transitions

$\gamma = 1$

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

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

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

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

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')] \quad \forall s \in \mathcal{S}$$

# Iterative Policy Evaluation – One array version

---

Input  $\pi$ , the policy to be evaluated

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

Repeat

$$\Delta \leftarrow 0$$

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

$$v \leftarrow V(s)$$

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

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

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

Output  $V \approx v_\pi$

Enough Prediction,  
let's start towards Control!

# Policy improvement theorem

- Given the value function for *any policy*  $\pi$ :

$$q_\pi(s, a) \quad \text{for all } s, a$$

- It can always be **greedified** to obtain a *better policy*:

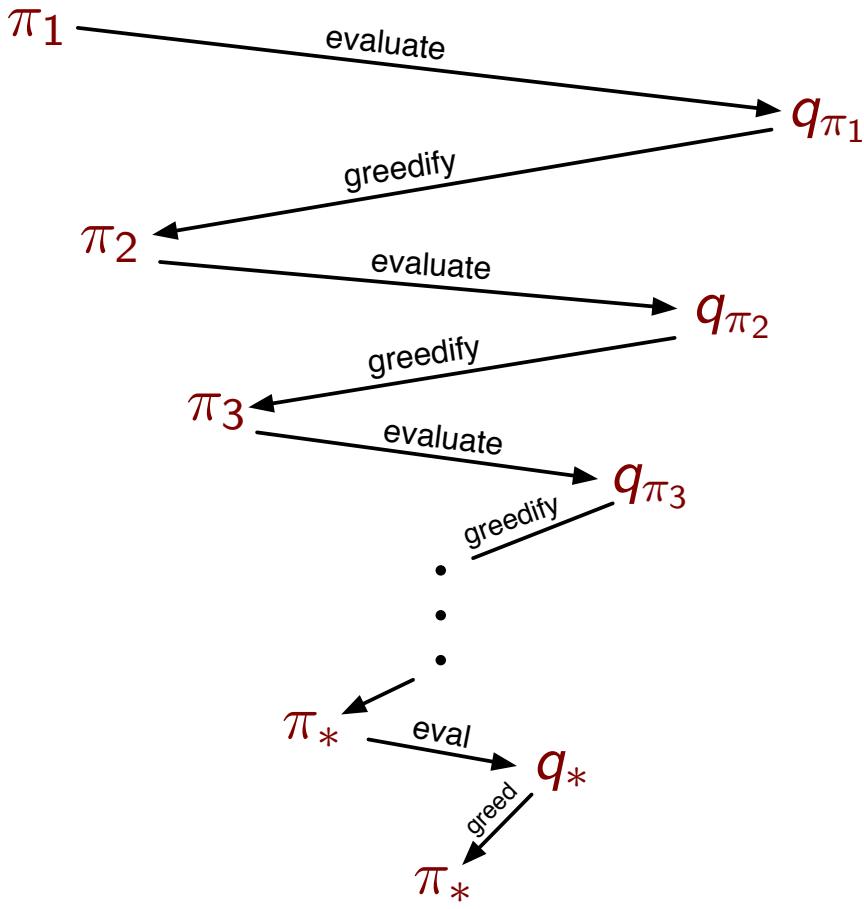
$$\pi'(s) = \arg \max_a q_\pi(s, a) \quad (\pi' \text{ is not unique})$$

- where better means:

$$q_{\pi'}(s, a) \geq q_\pi(s, a) \quad \text{for all } s, a$$

- with equality only if both policies are optimal

# The dance of policy and value (Policy Iteration)



Any policy evaluates to a unique value function (soon we will see how to learn it)

which can be greedified to produce a better policy

That in turn evaluates to a value function

which can in turn be greedified...

Each policy is *strictly better* than the previous, until *eventually both are optimal*

There are *no local optima*

The dance converges in a *finite number of steps*, usually very few

# 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  $a \neq \pi(s)$ ?

And, we can compute  $q_\pi(s, a)$  from  $v_\pi$  by:

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

# 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  $a \neq \pi(s)$ ?

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

$$q_\pi(s, a) > v_\pi(s)$$

And, we can compute  $q_\pi(s, a)$  from  $v_\pi$  by:

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

# Policy Improvement Cont.

---

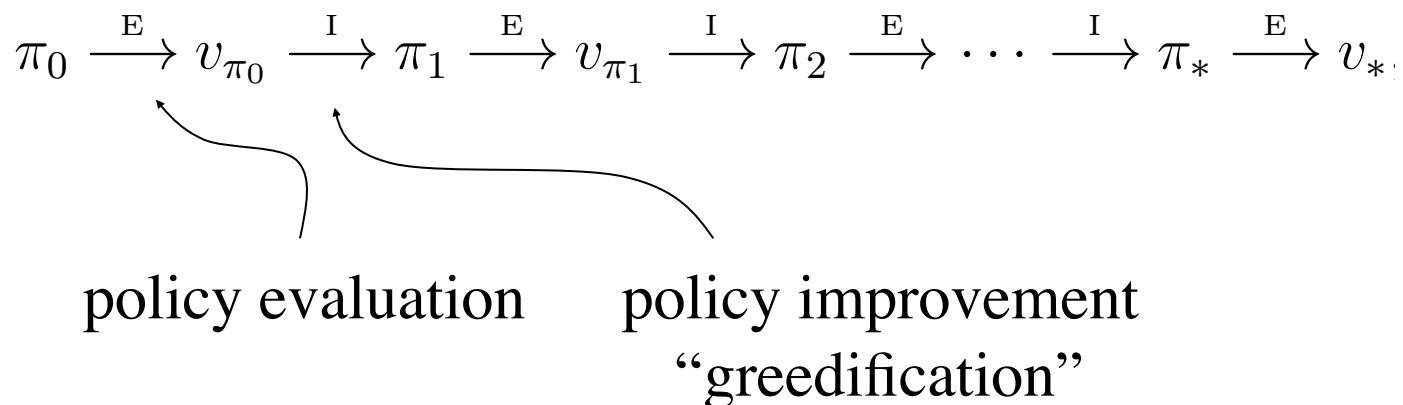
Do this for all states to get a new policy  $\pi' \geq \pi$  that is **greedy** with respect to  $v_\pi$ :

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

What if the policy is unchanged by this?  
Then the policy must be optimal!

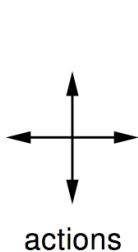
# Policy Iteration

---



# Iterative Policy Eval for the Small Gridworld

$\pi$  = equiprobable random action choices



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

$R = -1$   
on all transitions

$\gamma = 1$

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

for all  $s \in \mathcal{S}$

$V_k$ for the Random Policy	Greedy Policy w.r.t. $V_k$																
<table border="1"> <tbody> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> <tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr> </tbody> </table>	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	
0.0	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														

$k = 0$

$k = 1$	Greedy Policy w.r.t. $V_k$																
<table border="1"> <tbody> <tr><td>0.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr> <tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr> <tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr> <tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>0.0</td></tr> </tbody> </table>	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	
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														

$k = 2$

$k = 2$	Greedy Policy w.r.t. $V_k$																
<table border="1"> <tbody> <tr><td>0.0</td><td>-1.7</td><td>-2.0</td><td>-2.0</td></tr> <tr><td>-1.7</td><td>-2.0</td><td>-2.0</td><td>-2.0</td></tr> <tr><td>-2.0</td><td>-2.0</td><td>-2.0</td><td>-1.7</td></tr> <tr><td>-2.0</td><td>-2.0</td><td>-1.7</td><td>0.0</td></tr> </tbody> </table>	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	
0.0	-1.7	-2.0	-2.0														
-1.7	-2.0	-2.0	-2.0														
-2.0	-2.0	-2.0	-1.7														
-2.0	-2.0	-1.7	0.0														

$k = 3$

$k = 3$	Greedy Policy w.r.t. $V_k$																
<table border="1"> <tbody> <tr><td>0.0</td><td>-2.4</td><td>-2.9</td><td>-3.0</td></tr> <tr><td>-2.4</td><td>-2.9</td><td>-3.0</td><td>-2.9</td></tr> <tr><td>-2.9</td><td>-3.0</td><td>-2.9</td><td>-2.4</td></tr> <tr><td>-3.0</td><td>-2.9</td><td>-2.4</td><td>0.0</td></tr> </tbody> </table>	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	
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														

$k = 10$

$k = 10$	Greedy Policy w.r.t. $V_k$																
<table border="1"> <tbody> <tr><td>0.0</td><td>-6.1</td><td>-8.4</td><td>-9.0</td></tr> <tr><td>-6.1</td><td>-7.7</td><td>-8.4</td><td>-8.4</td></tr> <tr><td>-8.4</td><td>-8.4</td><td>-7.7</td><td>-6.1</td></tr> <tr><td>-9.0</td><td>-8.4</td><td>-6.1</td><td>0.0</td></tr> </tbody> </table>	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	
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														

$k = \infty$

$k = \infty$	Greedy Policy w.r.t. $V_k$																
<table border="1"> <tbody> <tr><td>0.0</td><td>-14.</td><td>-20.</td><td>-22.</td></tr> <tr><td>-14.</td><td>-18.</td><td>-20.</td><td>-20.</td></tr> <tr><td>-20.</td><td>-20.</td><td>-18.</td><td>-14.</td></tr> <tr><td>-22.</td><td>-20.</td><td>-14.</td><td>0.0</td></tr> </tbody> </table>	0.0	-14.	-20.	-22.	-14.	-18.	-20.	-20.	-20.	-20.	-18.	-14.	-22.	-20.	-14.	0.0	
0.0	-14.	-20.	-22.														
-14.	-18.	-20.	-20.														
-20.	-20.	-18.	-14.														
-22.	-20.	-14.	0.0														

random policy

optimal policy

# Jack's Car Rental

---

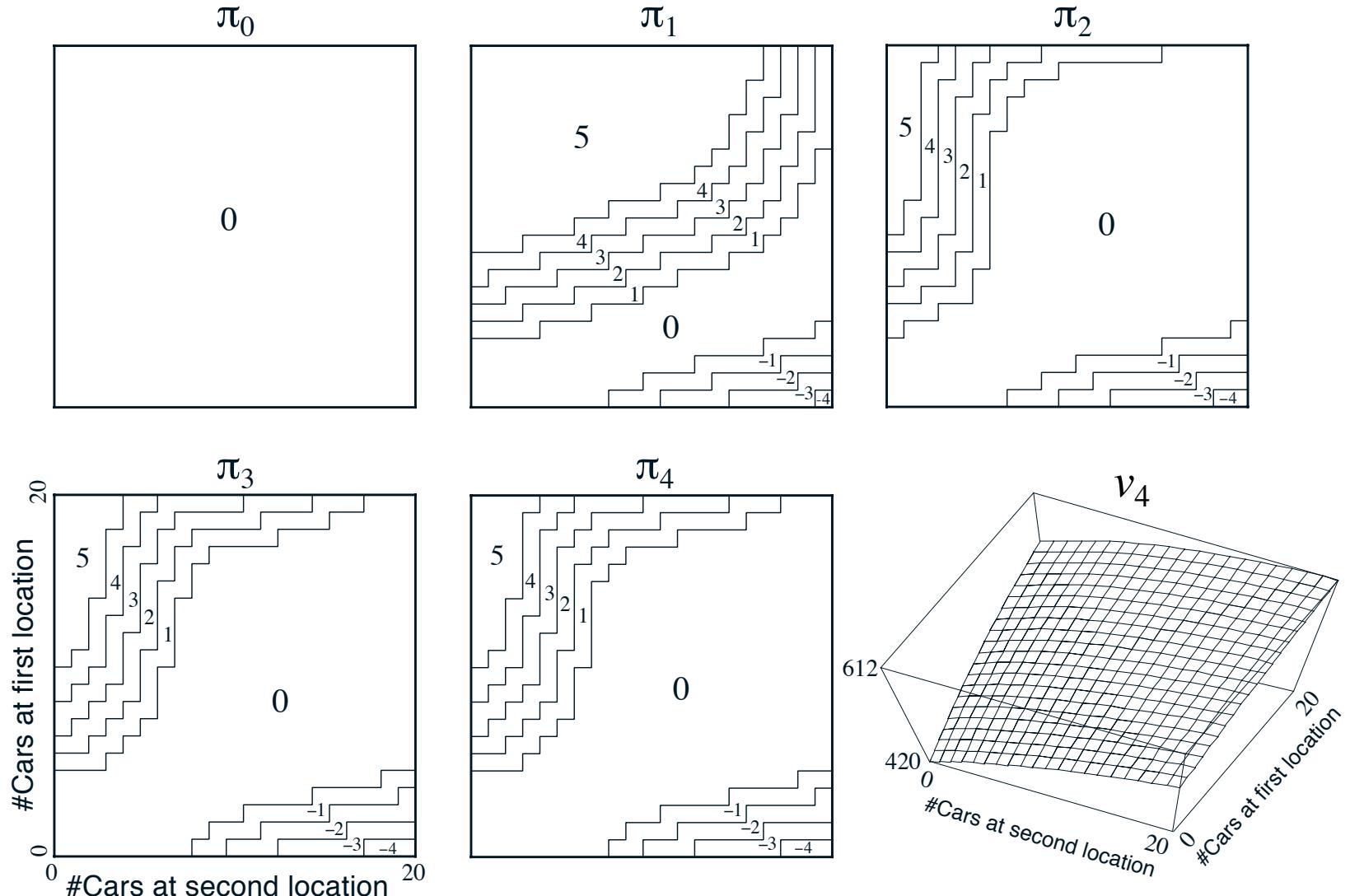
- \$10 for each car rented (must be available when request rec'd)
- Two locations, maximum of 20 cars at each
- Cars returned and requested randomly
  - $n$  returns/requests with prob  $\frac{\lambda^n}{n!}e^{-\lambda}$  (Poisson distribution)
  - 1st location: average requests = 3, average returns = 3
  - 2nd location: average requests = 4, average returns = 2
- Can move up to 5 cars between locations overnight
  - at a cost of \$2/car
- States, Actions, Rewards?
- Transition probabilities? Discounting?

# Jack's Car Rental

---

- \$10 for each car rented (must be available when request rec'd)
- Two locations, maximum of 20 cars at each
- Cars returned and requested randomly
  - $n$  returns/requests with prob  $\frac{\lambda^n}{n!}e^{-\lambda}$  (Poisson distribution)
  - 1st location: average requests = 3, average returns = 3
  - 2nd location: average requests = 4, average returns = 2
- Can move up to 5 cars between locations overnight
  - at a cost of \$2/car
- States, Actions, Rewards?
- Transition probabilities? Discounting?  $\gamma = 0.9$

# Jack's Car Rental



# Jack's CR Exercise

---

- Suppose the first car moved is free
  - From 1st to 2nd location
  - Because an employee travels that way anyway (by bus)
- Suppose only 10 cars can be parked for free at each location
  - More than 10 cost \$4 for using an extra parking lot
- Such arbitrary nonlinearities are common in real problems

# Policy Iteration – One array version (+ policy)

---

## 1. Initialization

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

## 2. Policy Evaluation

Repeat

$$\Delta \leftarrow 0$$

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

$$v \leftarrow V(s)$$

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

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

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

## 3. Policy Improvement

*policy-stable*  $\leftarrow$  true

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

$$a \leftarrow \pi(s)$$

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

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

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

# Value Iteration

---

Recall the **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')] \quad \forall s \in \mathcal{S}$$

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

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')] \quad \forall s \in \mathcal{S}$$

# Value Iteration – One array version

---

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

Repeat

$$\Delta \leftarrow 0$$

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

$$v \leftarrow V(s)$$

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

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

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

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

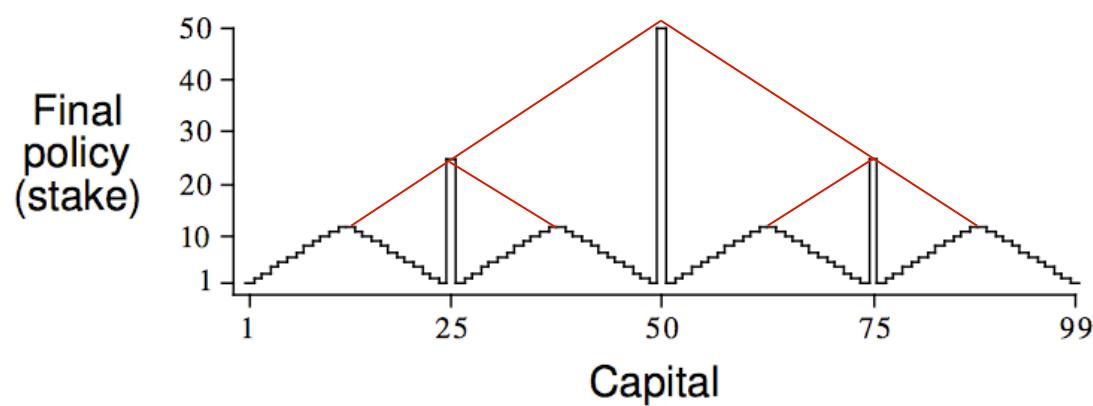
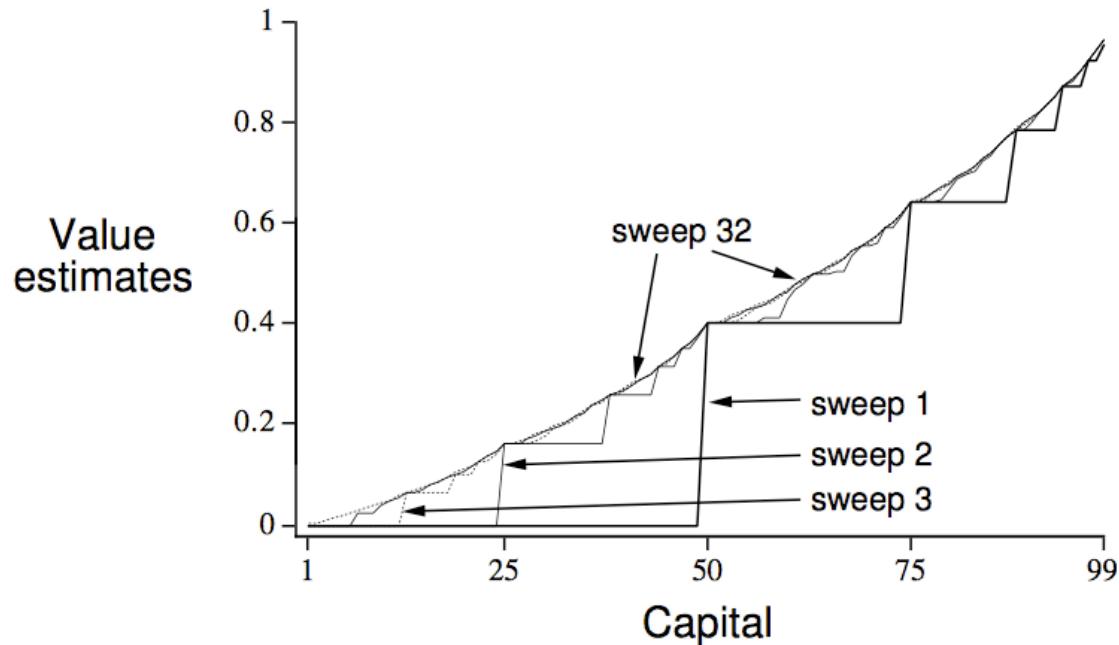
$$\pi(s) = \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

# Gambler's Problem

---

- Gambler can repeatedly bet \$ on a coin flip
- Heads he wins his stake, tails he loses it
- Initial capital  $\in \{\$1, \$2, \dots \$99\}$
- Gambler wins if his capital becomes \$100  
loses if it becomes \$0
- Coin is unfair
  - Heads (gambler wins) with probability  $p = .4$
- States, Actions, Rewards? Discounting?

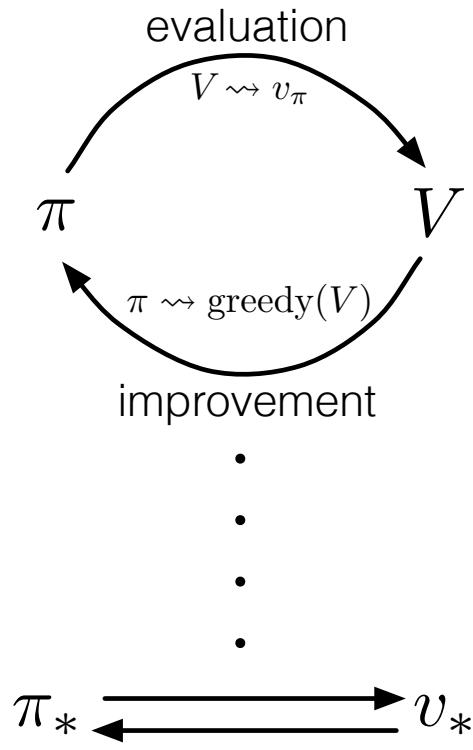
# Gambler's Problem Solution



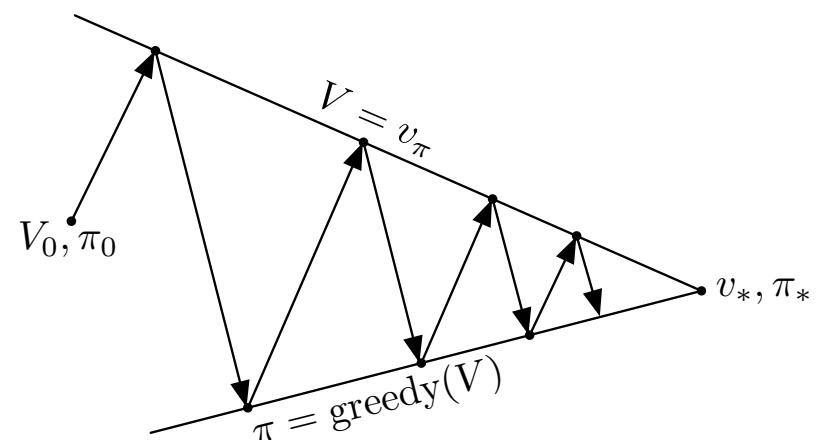
# Generalized Policy Iteration

**Generalized Policy Iteration (GPI):**

any interaction of policy evaluation and policy improvement,  
independent of their granularity.



A geometric metaphor for convergence of GPI:



# 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.

# 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 is 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 (prediction)
  - ❑ Policy improvement: form a greedy policy, if only locally
  - ❑ Policy iteration: alternate the above two processes (control)
  - ❑ Value iteration: backups with a max (control)
- 
- ❑ 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
  - ❑ Biggest limitation of DP is that it requires a *probability model* (as opposed to a generative or simulation model)