Dynamic Programming

Chapter 4, Sutton & Barto

DP concepts

- What is a backup?
  - how many states would be involved?
  - what do we use to determine the states involved in a backup?
- What is a sweep?
  - what determines the states involved in a sweep?

Relating DP concepts to incremental learning

- How do we handle exploration in dynamic programming?
  - what is exploration?
  - we don’t interact with the environment!
- Can dynamic programming work in non-deterministic domains? How do we modify the algorithms to account for stochastic outcomes? Where did we assume determinism?
- Computing state value in a sweep != visiting a state!

Policy Evaluation

- Given a policy π, compute the state-value function v_π

- Recall the **State-value function for policy** π

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}$$

- and the Bellman equation for v_π

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

Iterative methods

- DP methods are called *iterative* because they produce a sequence of value function updates that eventually converge to v_π:

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

- Policy-evaluation methods perform what is called a **full backup** because they update the value of each state, based on **all possible** next states

- **Every iteration** backs up the value of **every single state**: a sweep consists of applying a **backup operation** to each state.

Basic idea of full policy-evaluation backup

- Turn the Bellman equation into an incremental update formula:

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

- How?

Expected update

Basic idea of full policy-evaluation backup

- Turn the Bellman equation into an incremental update formula:

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

- How?

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

Expected update

Iterative Policy Evaluation – One array version

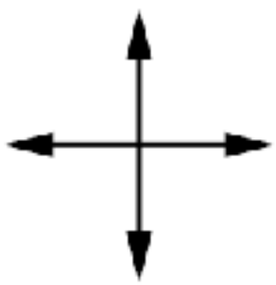
Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Iterative Policy Evaluation – One array version

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

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \big[r + \gamma V(s') \big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

A Small Gridworld



actions

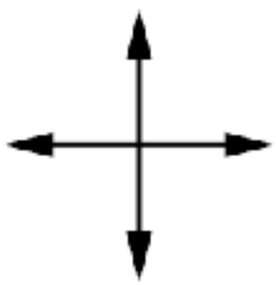
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
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

A Small Gridworld



actions

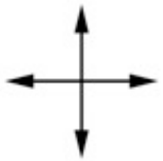
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

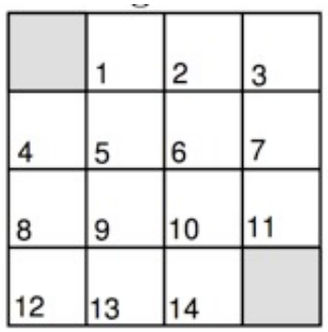
$\gamma = 1$

- What is p(6, -1 | 5, right) = ?
- p(10, r | 5, right) = ?
- p(13, -1 | 13, down) = ?

Iterative Policy Eval for the Small Gridworld



actions



$R = -1$
on all transitions

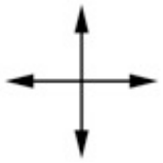
$\gamma = 1$

Single array version

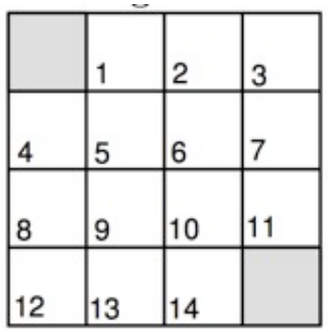
Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

$\pi =$ equiprobable random action choices

Iterative Policy Eval for the Small Gridworld



actions



$R = -1$
on all transitions

$\gamma = 1$

$k = 0$

v_k for the
Random Policy

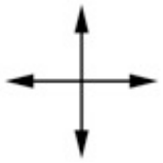
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Single array version

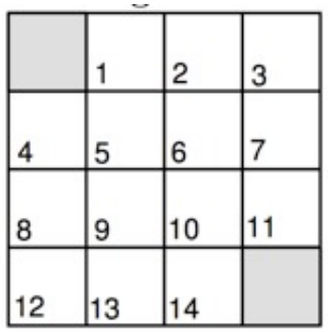
Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

$\pi =$ equiprobable random action choices

Iterative Policy Eval for the Small Gridworld



actions



$R = -1$
on all transitions

$\gamma = 1$

Single array version

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

$k = 0$

v_k for the Random Policy

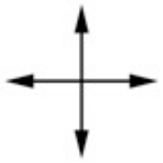
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

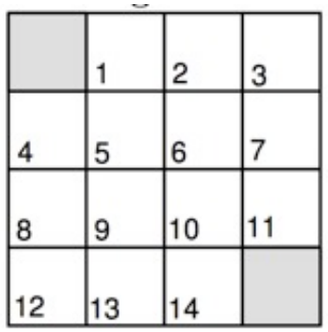
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$\pi =$ equiprobable random action choices

Iterative Policy Eval for the Small Gridworld



actions



$R = -1$
on all transitions

$\gamma = 1$

Single array version

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

v_k for the Random Policy

$k = 0$

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

$k = 1$

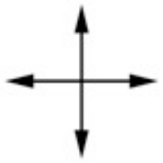
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

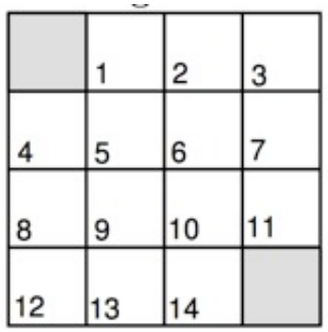
$k = 2$

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

$\pi =$ equiprobable random action choices

Iterative Policy Eval for the Small Gridworld





$R = -1$
on all transitions

$\gamma = 1$

Single array version

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

v_k for the Random Policy

$k = 0$

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

$k = 10$

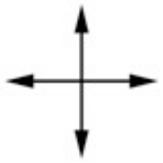
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

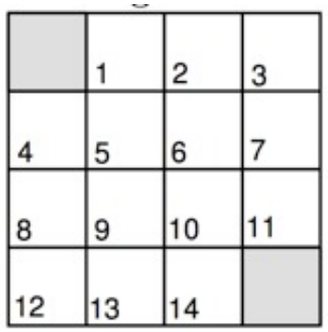
$k = \infty$

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

$\pi =$ equiprobable random action choices

Iterative Policy Eval for the Small Gridworld





$R = -1$
on all transitions

$\gamma = 1$

Single array version

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

results using **2-array version**

v_k for the Random Policy

$k = 0$

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

$k = 10$

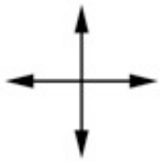
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

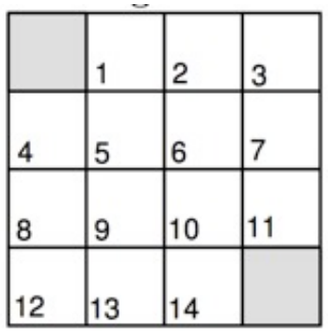
$k = \infty$

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

$\pi =$ equiprobable random action choices

Iterative Policy Eval for the Small Gridworld





$R = -1$
on all transitions

$\gamma = 1$

Single array version

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

results using **2-array version**

v_k for the Random Policy

$k = 0$

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

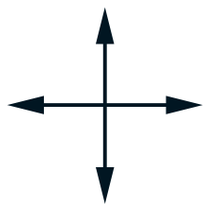
$\pi =$ equiprobable random action choices

- $v_{k=\infty}$ is v_π.
- What does v_π tell us?

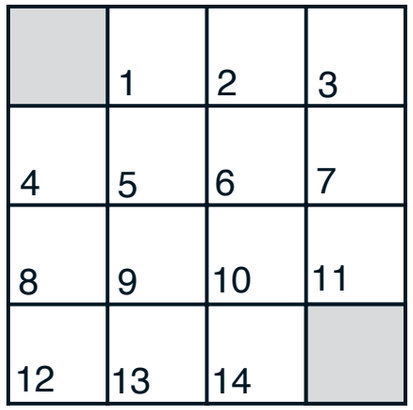
Implementing Iterative Policy Eval

- Two array version:
  - use one array for the old values, and another for the new values
- One array version:
  - Update in place
  - Each new value immediately overwrites the old value
  - Sometimes new values from the current **sweep** will be used instead of old values
- Both versions converge, and the one array version is often faster

Relating q and v



actions

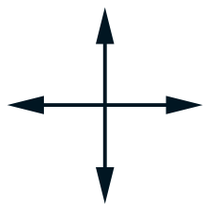


$R = -1$
on all transitions

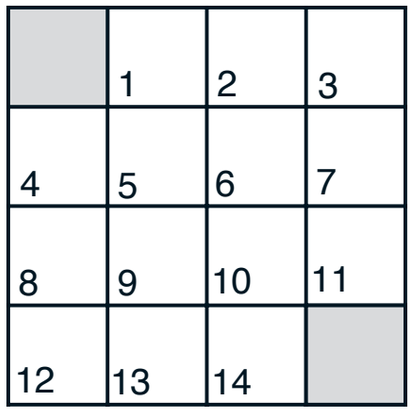
v_k for the
Random Policy

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

Relating q and v



actions



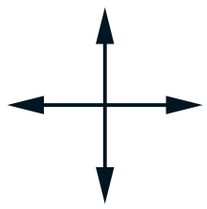
$R = -1$
on all transitions

v_k for the
Random Policy

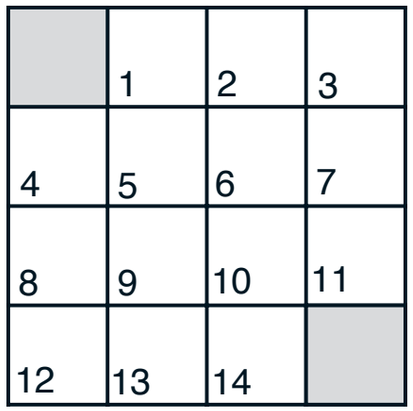
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

- what is $q_\pi(11, \textit{down})$?

Relating q and v



actions



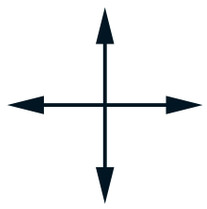
$R = -1$
on all transitions

v_k for the
Random Policy

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

- what is $q_\pi(11, \mathit{down})$?
- what is $q_\pi(7, \mathit{down})$?

Relating q and v



actions

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

$R = -1$
on all transitions

v_k for the
Random Policy

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

- what is $q_\pi(11, \textit{down})$?
- what is $q_\pi(7, \textit{down})$?

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

Policy evaluation

- Given a policy, we can use iterative policy **evaluation** to tell us how good the policy is, in terms of expected discounted total future reward
  - v_π allows us to *predict*, in each state, the expected return

- RL is largely about finding near-optimal policies, maybe we can use the Bellman equation again to make a DP algorithm to find better policies…

Enough Prediction,
let's start towards Control!

Policy improvement

- We want to know how to change the policy to make it better
- Could ask: what if we did a **different** action in state s, and then followed π after that
  - Is this better or worse than π
  - Bettor or worse than $v\pi(s)$

Policy improvement theorem

- Given the value function for *any policy* π

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

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

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

- where better means:

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

- with equality only if both policies are optimal

Policy Improvement

Suppose we have computed v_π for a deterministic policy π.

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_π by:

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

Policy Improvement

Suppose we have computed v_π for a deterministic policy π.

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_π by:

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

Policy Improvement Cont.

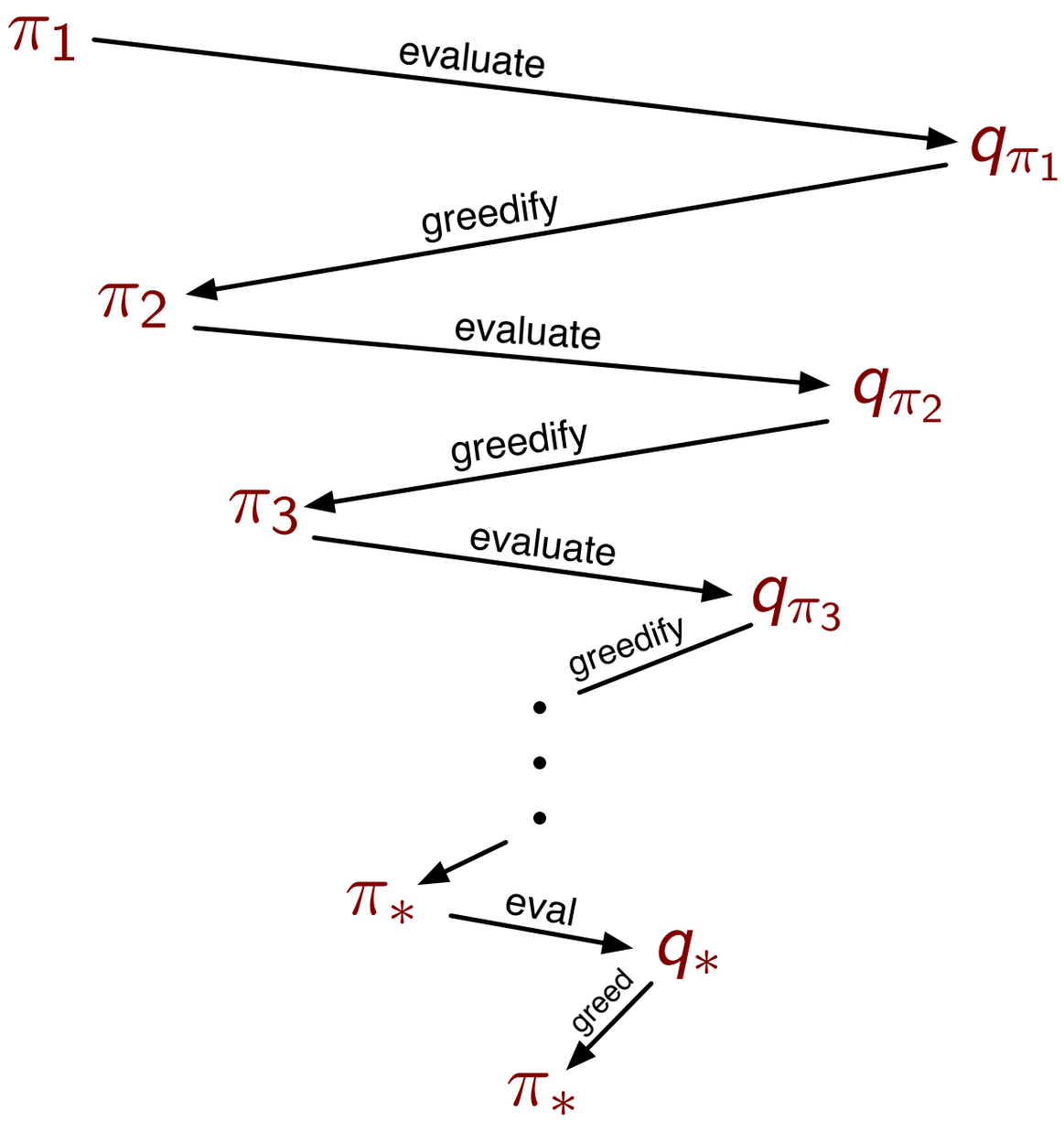
Do this for all states to get a new policy $\pi' \geq \pi$ that is **greedy** with respect to v_π :

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

What if the policy is unchanged by this?

Then the policy must be 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…

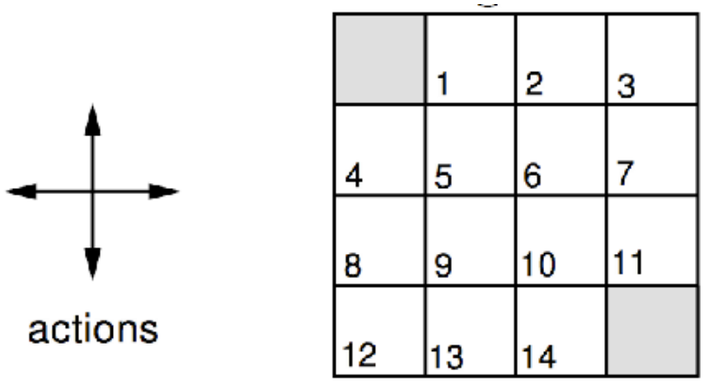
Each policy is *strictly better* than the previous, until *eventually both are optimal*

There are *no local optima*

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

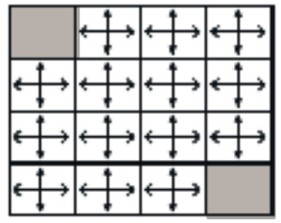
Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



$R = -1$
on all transitions

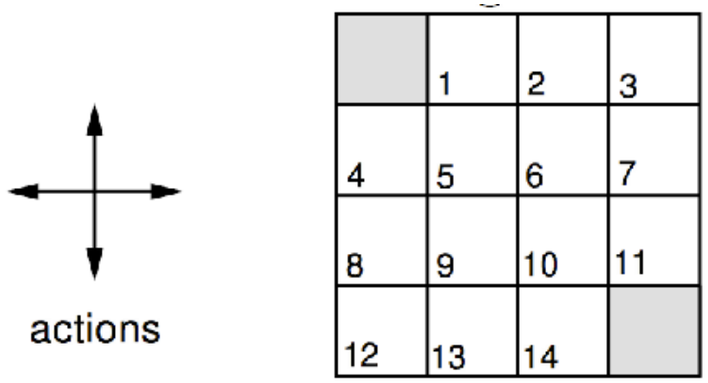
$\gamma = 1$

	V_k for the Random Policy	Greedy Policy w.r.t. V_k
$k = 0$	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	 ← random policy
$k = 1$	0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 0.0	
$k = 2$	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0	
$k = 3$	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	

- 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

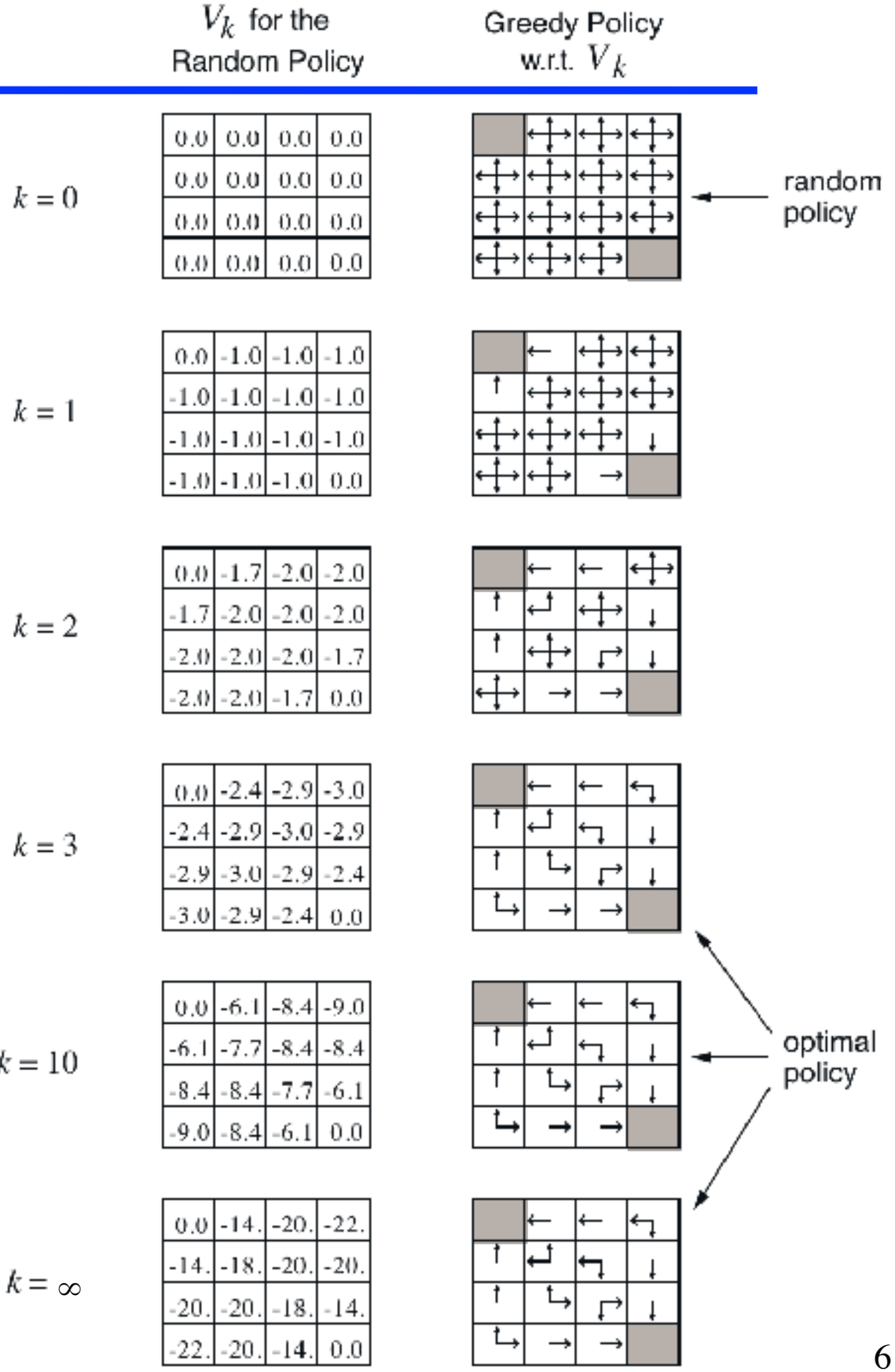
π = equiprobable random action choices



$R = -1$
on all transitions

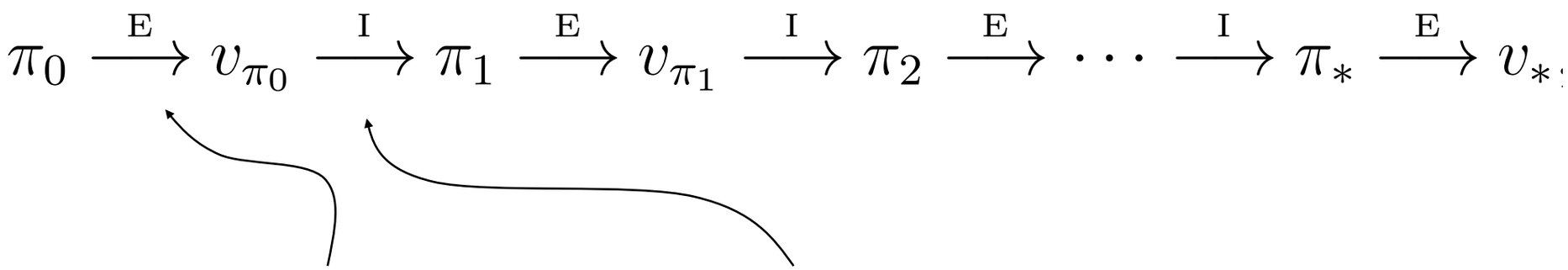
$\gamma = 1$

- 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



Policy Iteration

Once we improve π using v_π, we get a new policy π', then we can compute $v_{\pi'}$, then we can use $v_{\pi'}$ to get π'', and so on



Policy iteration: basic algorithmic structure

1. initialize V and π

2. run **policy evaluation** till near convergence (compute V_π)

3. greedify: produce π' from π, making π' equal to the action the maximizes V_π in **each state**

4. set $\pi = \pi$'

5. Goto 2

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

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

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

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

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

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

think of this part like argmax Q(s,a)

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

Step 2 involves repeated sweeps over the states

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

think of this part like argmax Q(s,a)

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

each time we call policy evaluation (step 2) we start with the value function of previous policy

>> speeds up convergence

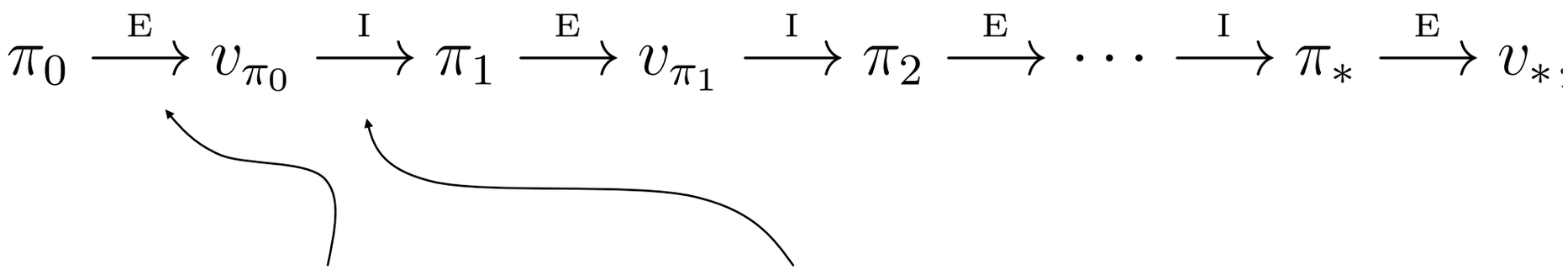
Step 2 involves repeated sweeps over the states

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

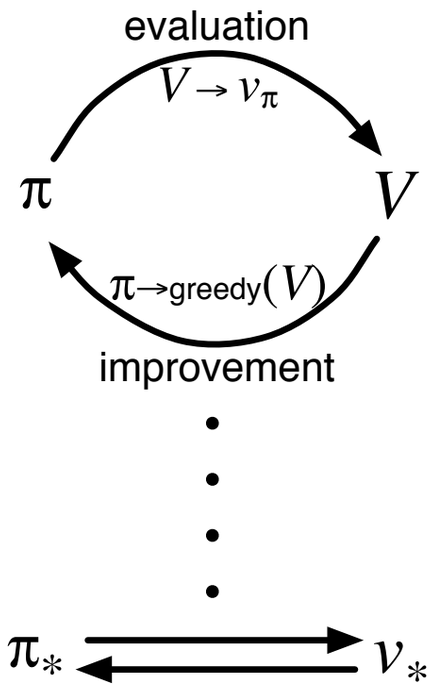
think of this part like argmax Q(s,a)

Policy Iteration

Once we improve π using v_π, we get a new policy π', then we can compute $v_{\pi'}$, then we can use $v_{\pi'}$ to get π'', and so on



Policy Iteration



- In policy iteration:
  1. *policy evaluation* makes the value function consistent with the current policy
  2. *policy improvement* makes the policy greedy wrt to the current value function

- These two processes alternate, one then the other, each waiting for the other to complete

Modifying Policy iteration

- What if we only did a few iterations of **policy evaluation**—updating v ?
    - or just one?

- Recall the impact of greedifying the value function for the random policy in the gridworld

Value iteration

- Recall the Bellman optimality equation, or the Bellman equation for π*:

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

- and the full value-iteration backup:

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

Value iteration algorithm

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

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

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

Value iteration example: Gambler's problem

- Gambler can repeatedly bet money on a coin flip
- Gambler has capital, total money he has to bet
- He bet $0, $1, unto his total capital call it stake
- If coin lands on **heads** he gets his stake
- If coin lands tails he losses his stake (subtracted from his capital)
- Coin in unfair
  - Heads (gambler win) with probability p =.4
- Game ends when he wins, gets $100, or losses gets $0

Gambler’s problem

Gambler's problem

- What are the states?

Gambler's problem

- What are the states?
  - gambler's capital, s ϵ {1,2, …, 99}

Gambler's problem

- What are the states?
  - gambler's capital, s ϵ {1,2, …, 99}
  - terminal states 0 and 100; Let $V_0 = 0$

Gambler's problem

- What are the states?
  - gambler's capital, s ϵ {1,2, …, 99}
  - terminal states 0 and 100; Let $V_0 = 0$
- What are the rewards?

Gambler's problem

- What are the states?
  - gambler's capital, s ϵ {1,2, …, 99}
  - terminal states 0 and 100; Let $V_0 = 0$
- What are the rewards?
  - +1 win, zero otherwise

Gambler's problem

- What are the states?
  - gambler's capital, s ϵ {1,2, …, 99}
  - terminal states 0 and 100; Let $V_0 = 0$
- What are the rewards?
  - +1 win, zero otherwise
- What are the actions:

Gambler's problem

- What are the states?
  - gambler's capital, s ϵ {1,2, …, 99}
  - terminal states 0 and 100; Let $V_0 = 0$
- What are the rewards?
  - +1 win, zero otherwise
- What are the actions:
  - a = [0:min(s,100-s)]

Gambler’s problem

Gambler’s problem

- What is γ ? Episodic? Continuing?

Gambler’s problem

- What is γ ? Episodic? Continuing?
  - γ = 1.0

Gambler's problem

- What is γ ? Episodic? Continuing?
  - γ = 1.0
- What is the goal?

Gambler's problem

- What is γ ? Episodic? Continuing?
  - γ = 1.0
- What is the goal?
  - learn $v_*(s)$, which we could then use to compute π_*

Gambler's problem

- What is γ ? Episodic? Continuing?
  - γ = 1.0
- What is the goal?
  - learn $v_*(s)$, which we could then use to compute π_*
- Policy maps states to actions, capital —> stakes

Gambler's problem

- What is γ ? Episodic? Continuing?
  - γ = 1.0
- What is the goal?
  - learn $v_*(s)$, which we could then use to compute π_*
  - Policy maps states to actions, capital —> stakes
- What does $v_*(s)$ encode?

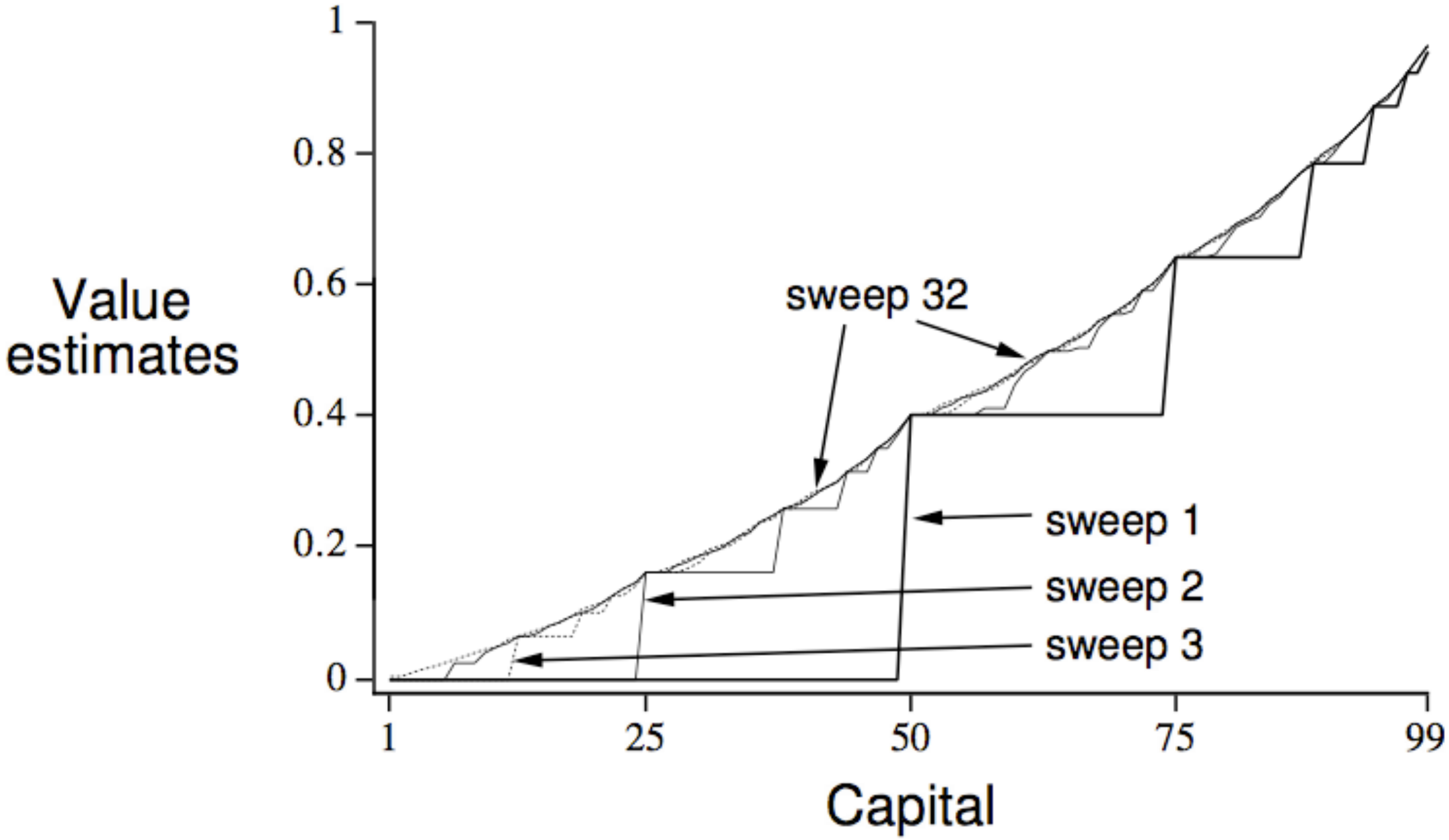
Gambler's problem

- What is γ ? Episodic? Continuing?
  - γ = 1.0
- What is the goal?
  - learn $v_*(s)$, which we could then use to compute π_*
  - Policy maps states to actions, capital —> stakes
- What does $v_*(s)$ encode?
  - the probability of winning from each state (amount of $$) if we made bets according to the optimal policy

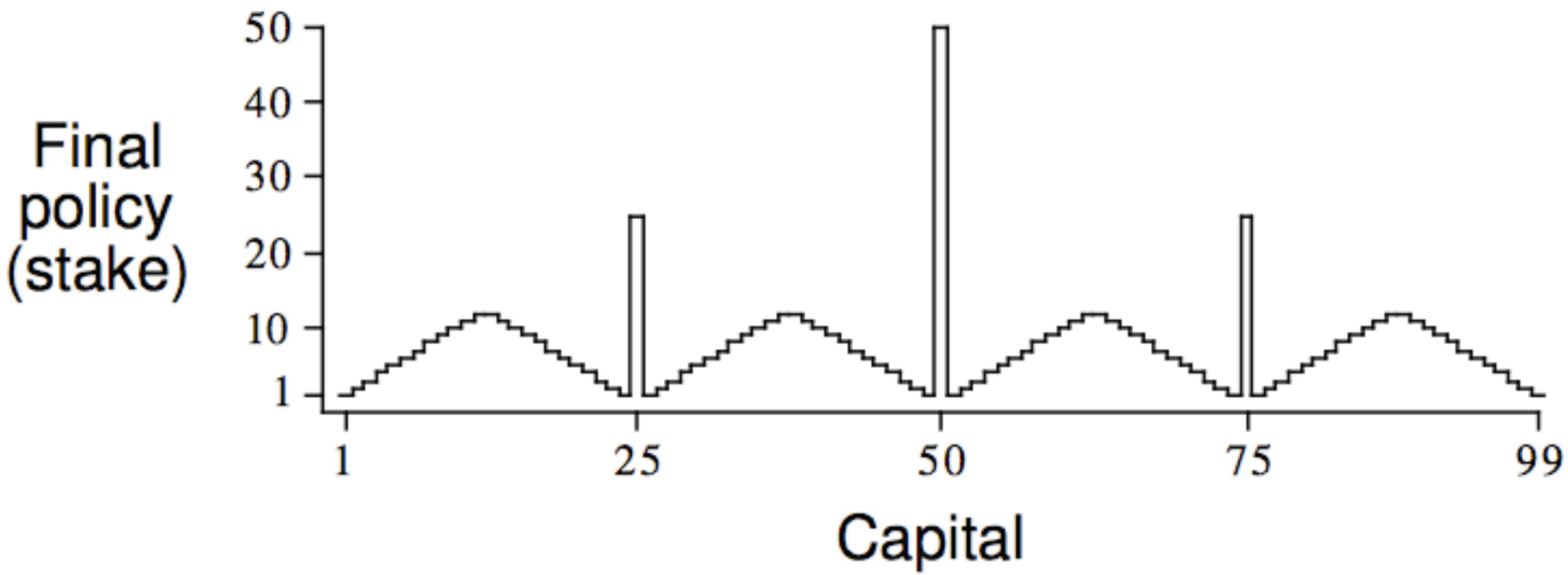
Gambler's problem

- What is γ ? Episodic? Continuing?
  - γ = 1.0
- What is the goal?
  - learn $v_*(s)$, which we could then use to compute π_*
  - Policy maps states to actions, capital —> stakes
- What does $v_*(s)$ encode?
  - the probability of winning from each state (amount of $$) if we made bets according to the optimal policy
- Why is it important we know pr(heads) = 0.4?

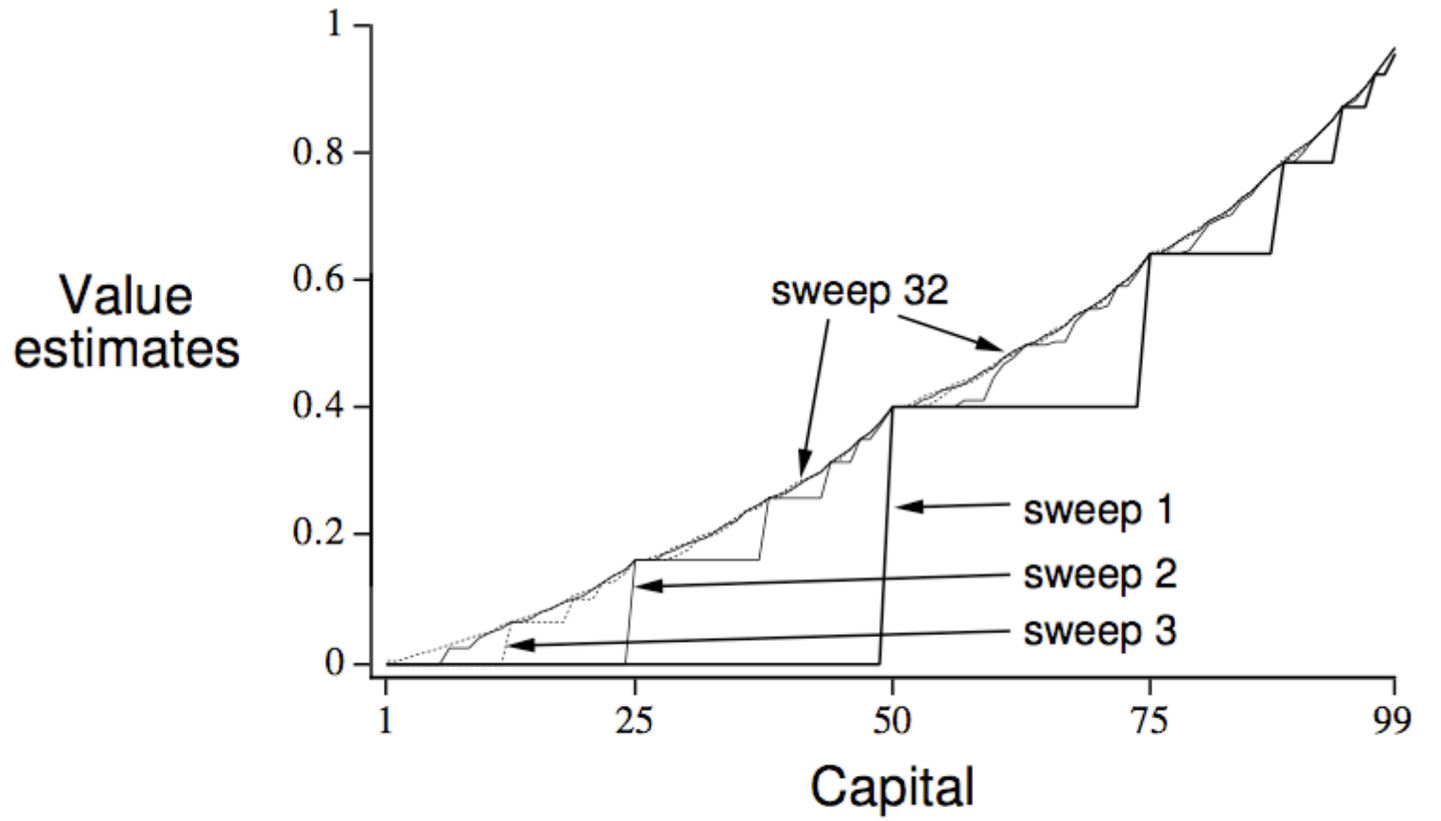
Gambler's Problem Solution

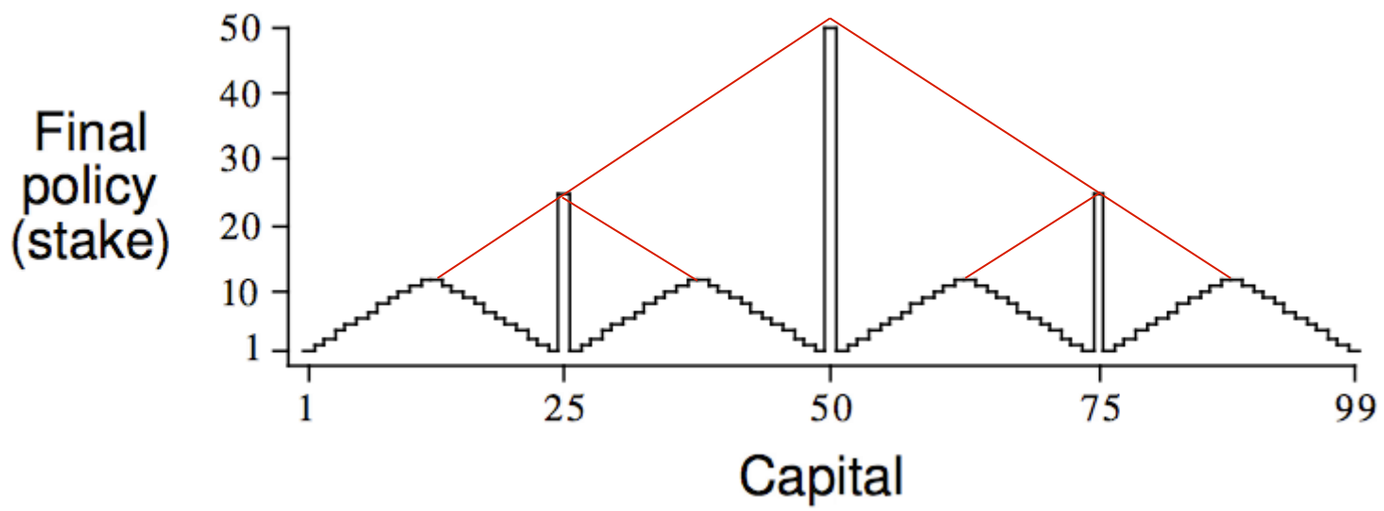


Gambler's Problem Solution



Gambler's Problem Solution





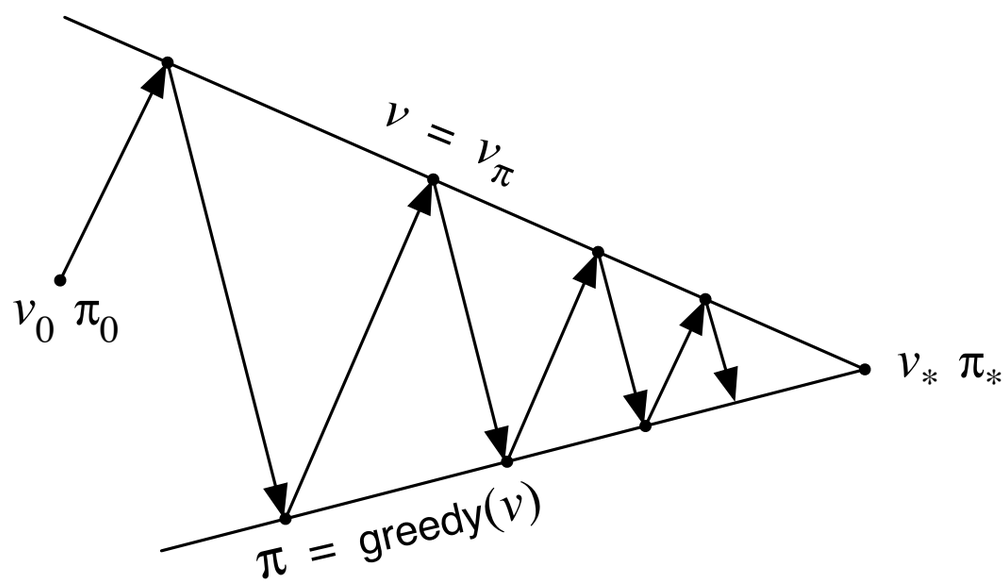
Generalized Policy Iteration

- In policy iteration consists of two simultaneous interacting processes: policy evaluation and policy improvement
  - Alternating, each completing before the other begins
- What other ways could this work?
  - a single iteration of policy iteration as in value iteration
  - only update the value or policy in some states
  - ??
- GPI: any interaction of policy evaluation and policy improvement independent of their granularity
  - As long long as both processes (PI and PE) continue to update all states, then this typically converges to optimal value func & policy
- Most RL methods can be described as GPI

GPI

- We have two interacting goals V —> v_π, and π=greedy(V)
- Progress towards one goal takes us away from the other, but the two processes together bring us closer to the goal of optimality
  - PI typically makes the value function incorrect wrt π'
  - PE (making V consistent with the policy), typically makes π no longer greedy wrt V

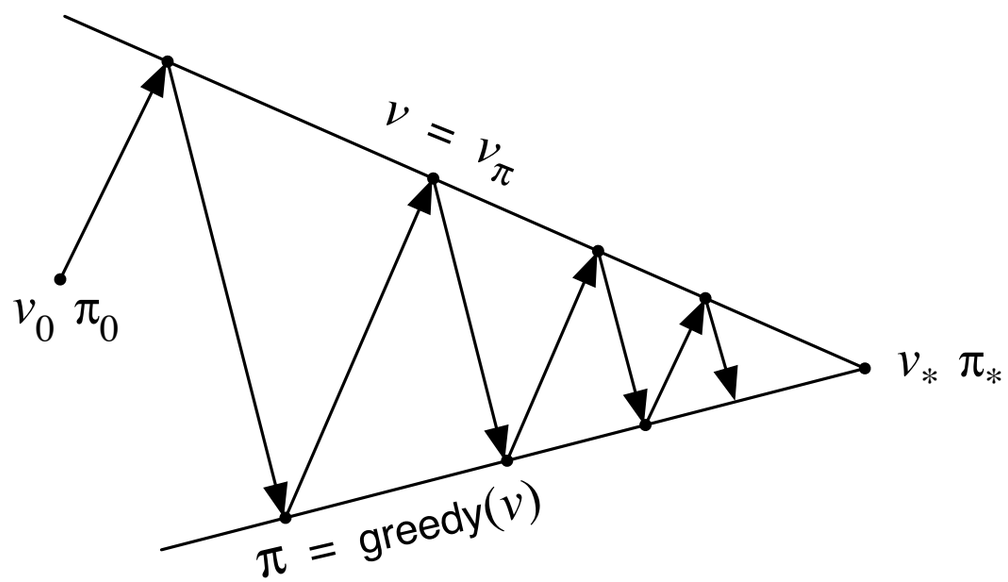
A geometric metaphor for convergence of GPI:



GPI

- We have two interacting goals V —> v_π, and π=greedy(V)
- Progress towards one goal takes us away from the other, but the two processes together bring us closer to the goal of optimality
  - PI typically makes the value function incorrect wrt π'
  - PE (making V consistent with the policy), typically makes π no longer greedy wrt V

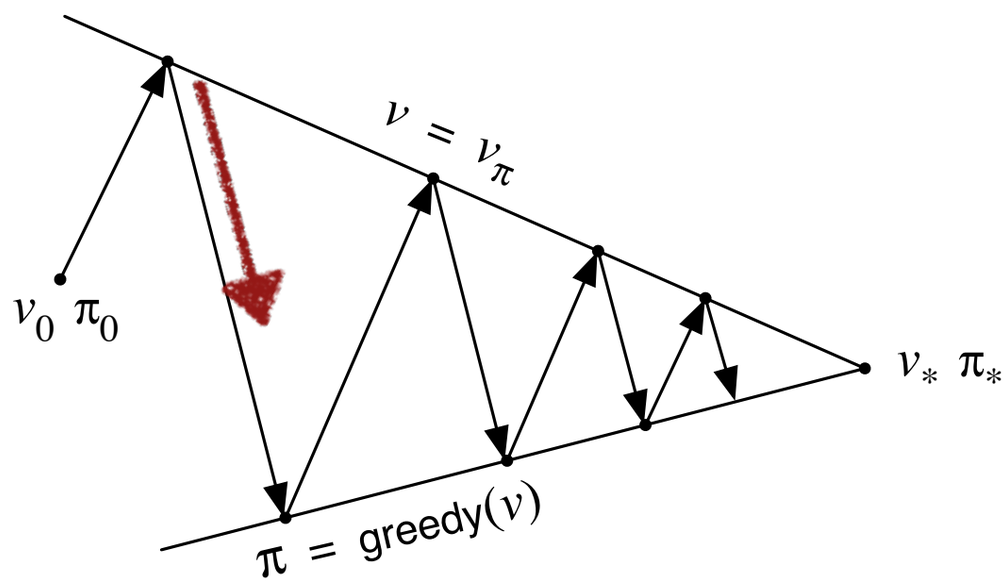
A geometric metaphor for convergence of GPI:



GPI

- We have two interacting goals V —> v_π, and π=greedy(V)
- Progress towards one goal takes us away from the other, but the two processes together bring us closer to the goal of optimality
  - PI typically makes the value function incorrect wrt π'
  - PE (making V consistent with the policy), typically makes π no longer greedy wrt V

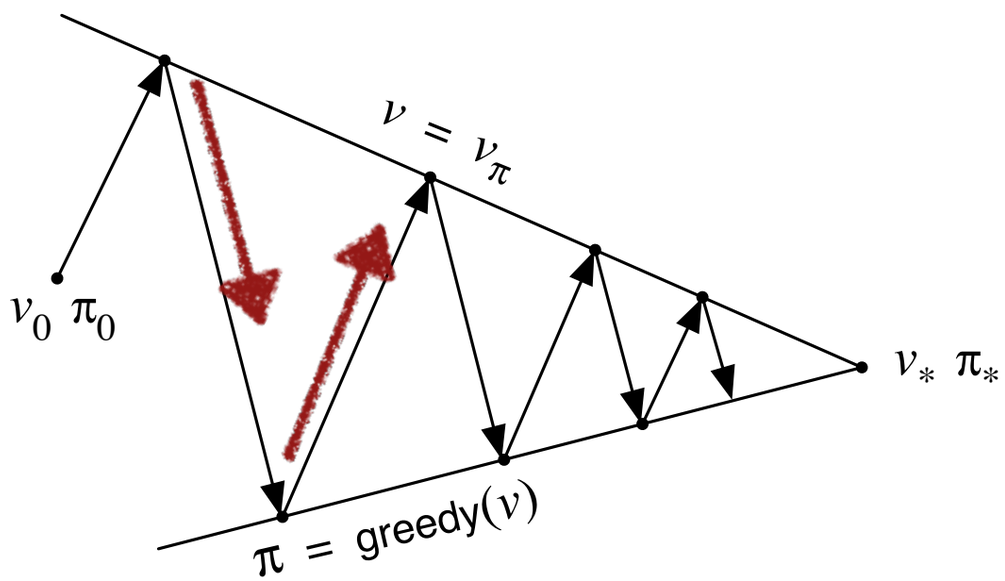
A geometric metaphor for convergence of GPI:



GPI

- We have two interacting goals V —> v_π, and π=greedy(V)
- Progress towards one goal takes us away from the other, but the two processes together bring us closer to the goal of optimality
  - PI typically makes the value function incorrect wrt π'
  - PE (making V consistent with the policy), typically makes π no longer greedy wrt V

A geometric metaphor for convergence of GPI:



Asynchronous Dynamic Programming

- All the DP methods we talked about so far require systematic sweeps of the entire state set: **loop over s**
- Asynchronous DP are organized in terms of systematic sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - ➡ Pick **a state** 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
- Distinct idea from GPI

Efficiency & Computation

- DP methods can find the optimal policy with time that is polynomial in the # states and actions, **worst case**
    - even though the # of policies is exponential in |*S*| and |*A*|
        - a policy search would require exponential, exhaustive search to get guarantees that we have for DP methods
- 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”).

DP in Practice

- DP methods typically converge must faster than the worst case
    - good initializations help!
- 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.
    - however, relatively few states occur along the optimal solution path
- Not clear in practice if policy iteration or value iteration is to be preferred

Jack's Car Rental

- We have **two rental locations**
- **Everyday** we have stochastic car returns to each location
- Everyday we have stochastic rental requests at each location
- Cars are available for rent the **day after** they are returned

Jack's Car Rental

- Jack can move cars between the two locations overnight
  - max cars you can move is 5, at a cost of $2 per car
- **Jack wants to rent cars, but he also only wants to move cars only when required**
  - every car he rents he gets $10
- Let's say each rental location holds 20 cars
  - if more than 20 are returned they go to national rental center

Jack’s car rental s,a,r

Jack’s car rental s,a,r

- What is t, the timestep

Jack’s car rental s,a,r

- What is t, the timestep
  - days

Jack’s car rental s,a,r

- What is t, the timestep
  - days
- What are the states?

Jack’s car rental s,a,r

- What is t, the timestep
  - days
- What are the states?
  - $[s_1,s_2]$, s1 ϵ [0,20] and s2 ϵ [0,20] // at the end of the day

Jack's car rental s,a,r

- What is t, the timestep
  - days
- What are the states?
  - $[s_1,s_2]$, s1 ϵ [0,20] and s2 ϵ [0,20] // at the end of the day
- What are the actions?

Jack's car rental s,a,r

- What is t, the timestep
  - days
- What are the states?
  - $[s_1,s_2]$, s1 ϵ [0,20] and s2 ϵ [0,20] // at the end of the day
- What are the actions?
  - a ϵ {-5,-4,…,0,1,…,5} // car movement overnight

Jack's car rental s,a,r

- What is t, the timestep
  - days
- What are the states?
  - $[s_1,s_2]$, s1 ϵ [0,20] and s2 ϵ [0,20] // at the end of the day
- What are the actions?
  - a ϵ {-5,-4,…,0,1,…,5} // car movement overnight
- What are the rewards?

Jack's car rental s,a,r

- What is t, the timestep
  - days
- What are the states?
  - $[s_1,s_2]$, s1 ϵ [0,20] and s2 ϵ [0,20] // at the end of the day
- What are the actions?
  - a ϵ {-5,-4,…,0,1,…,5} // car movement overnight
- What are the rewards?
  - R_t =10* satisfied_requests -2*moved

Jack's car rental s,a,r

- What is t, the timestep
  - days
- What are the states?
  - $[s_1,s_2]$, s1 ϵ [0,20] and s2 ϵ [0,20] // at the end of the day
- What are the actions?
  - a ϵ {-5,-4,…,0,1,…,5} // car movement overnight
- What are the rewards?
  - R_t =10* satisfied_requests -2*moved
- Is this episodic or continuing?

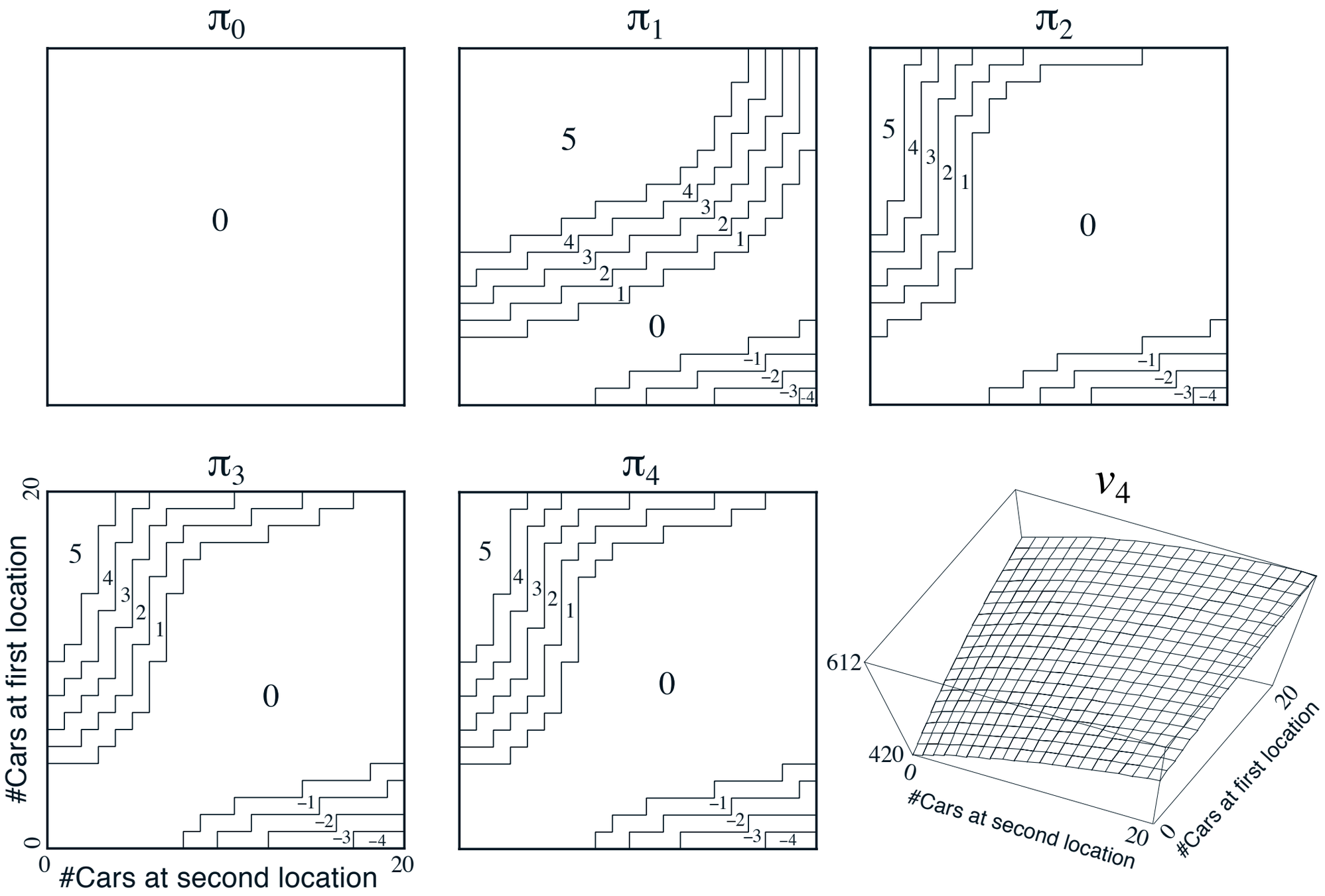
Jack's car rental s,a,r

- What is t, the timestep
  - days
- What are the states?
  - $[s_1,s_2]$, s1 ϵ [0,20] and s2 ϵ [0,20] // at the end of the day
- What are the actions?
  - a ϵ {-5,-4,…,0,1,…,5} // car movement overnight
- What are the rewards?
  - R_t =10* satisfied_requests -2*moved
- Is this episodic or continuing?
  - gamma = 0.9

Jack's car rental …

- Cars returned and requested randomly (these are not actions)
  - Poisson distribution governs the process
- What is p(s',r | s,a)?
  - location 1: #out ~ Poisson(λ=3), #in ~ Poisson(λ=3)
  - location 2: #out ~ Poisson(λ=4), #in ~ Poisson(λ=2)

Jack's Car Rental: Policy iteration result



Jack's Car Rental is different

- The state is 2D
- The transition dynamics are stochastic and based on more complex distributions
- Nontrivial implementation, compared to Gamble's domain

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
- Biggest limitation of DP is that it requires a *probability model* (as opposed to a generative or simulation model)