

Optimal Values and Policies

Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:
$$\pi \geq \pi' \quad \text{if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \mathcal{S}$$
- There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π_* .
- Optimal policies share the same **optimal state-value function**:

$$v_*(s) = \max_{\pi} v_{\pi}(s) \quad \text{for all } s \in \mathcal{S}$$

- Optimal policies also share the same **optimal action-value function**:

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a) \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}$$

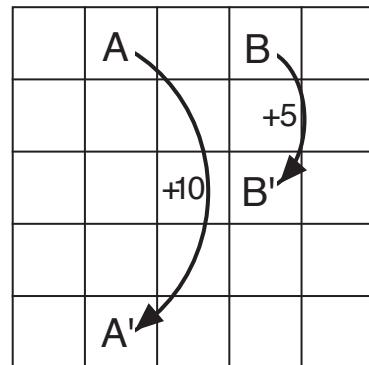
This is the expected return for taking action a in state s and thereafter following an optimal policy.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to v_* is an optimal policy.

Therefore, given v_* , one-step-ahead search produces the long-term optimal actions.

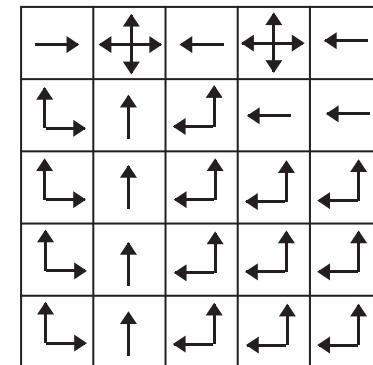
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

What About Optimal Action-Value Functions?

Given q_* , the agent does not even have to do a one-step-ahead search:

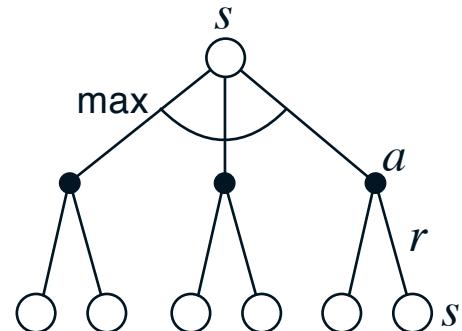
$$\pi_*(s) = \arg \max_a q_*(s, a)$$

Bellman Optimality Equation for v_*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

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

The relevant backup diagram:

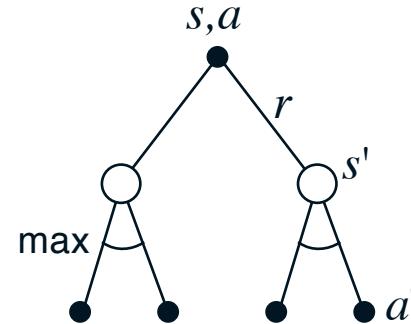


v_* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for q_*

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

The relevant backup diagram:



q_* is the unique solution of this system of nonlinear equations.

Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
 - accurate knowledge of environment dynamics;
 - we have enough space and time to do the computation;
 - the Markov Property.
- How much space and time do we need?
 - polynomial in number of states,
 - BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

Recall: Policy Evaluation

Policy Evaluation: for a given policy π , compute the state-value function v_π

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

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

Recall: **Bellman equation for v_π**

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

—a system of $|S|$ simultaneous equations

Iterative Policy Evaluation – 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 $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

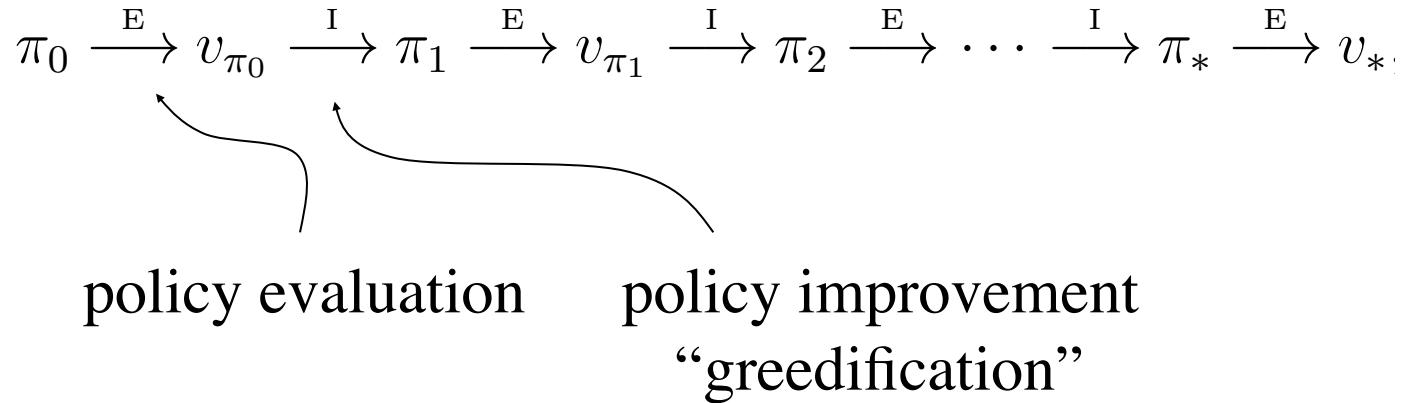
$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

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

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

Output $V \approx v_\pi$

Policy Iteration



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:

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

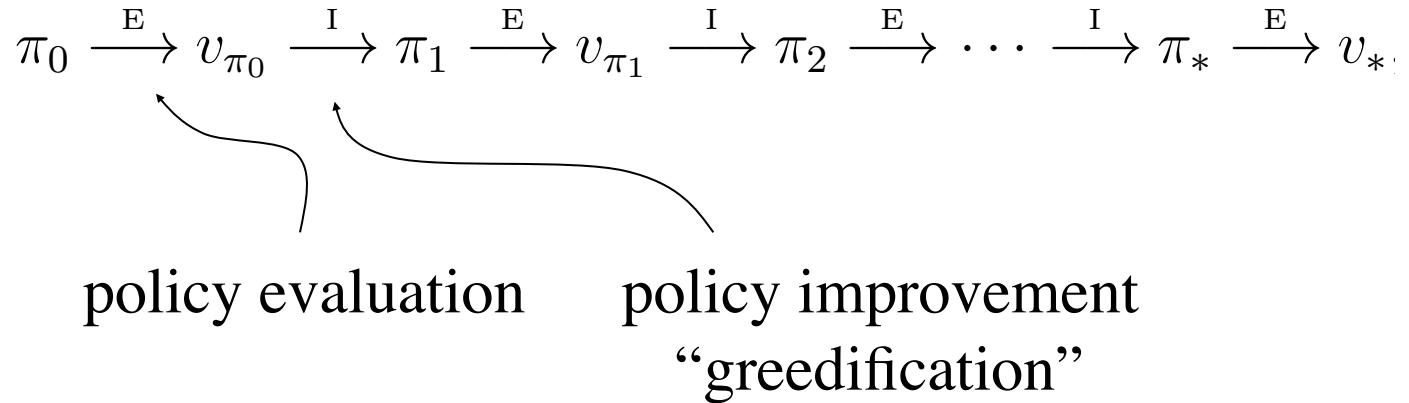
Policy Improvement Cont.

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

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

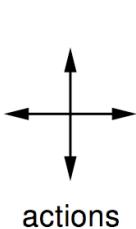
What if the policy is unchanged by this?
Then the policy must be optimal!

Policy Iteration



Iterative Policy Eval for the Small Gridworld

π = 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

V_k for the
Random Policy

Greedy Policy
w.r.t. V_k

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

$k = 10$

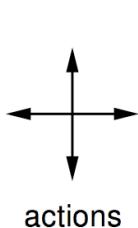
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

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

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



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

$R = -1$
on all transitions

$\gamma = 1$

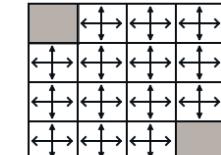
- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

$k = 0$

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

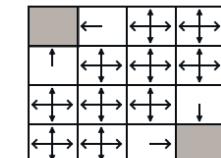
Greedy Policy
w.r.t. V_k



random
policy

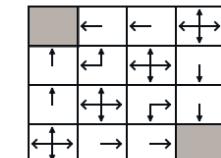
$k = 1$

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



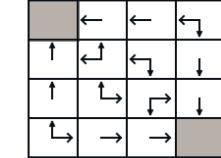
$k = 2$

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



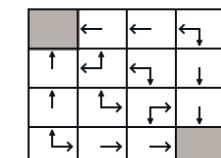
$k = 3$

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



$k = 10$

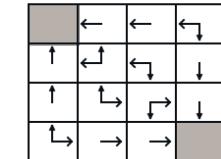
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



optimal
policy

$k = \infty$

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



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 $s \in \mathcal{S}$:

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

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

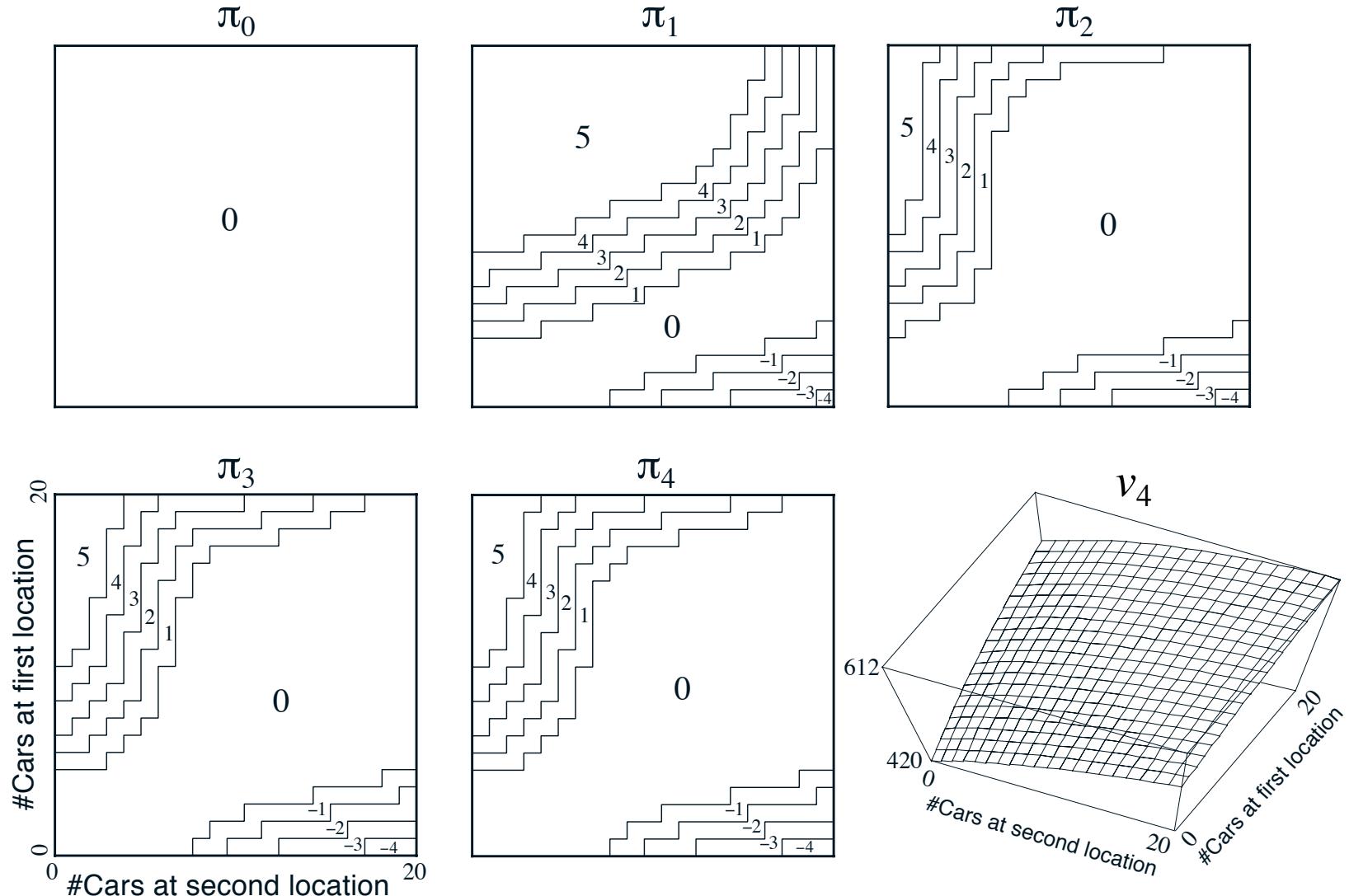
If $a \neq \pi(s)$, then *policy-stable* \leftarrow false

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

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
 - Poisson distribution, n returns/requests with prob $\frac{\lambda^n}{n!} e^{-\lambda}$
 - 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



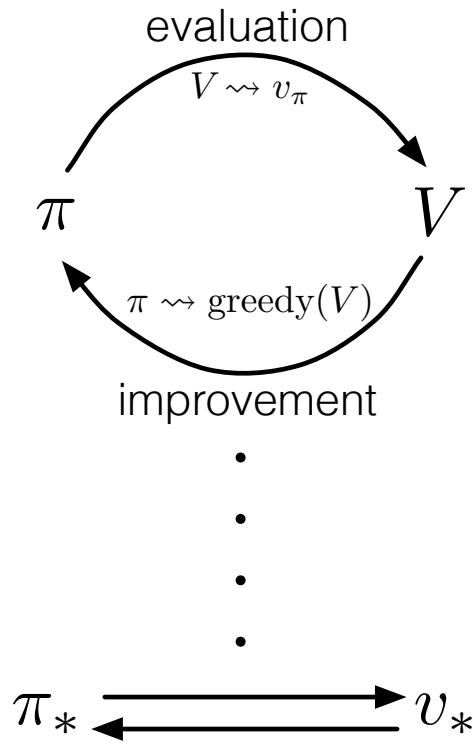
Jack's CR Exercise

- 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

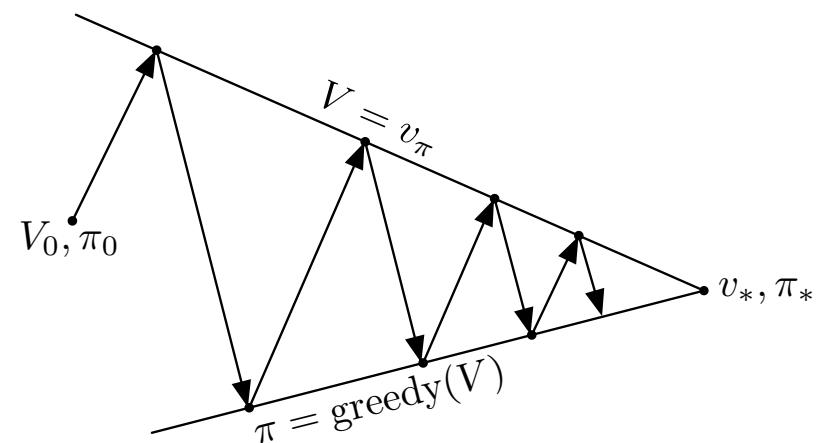
Generalized Policy Iteration

Generalized Policy Iteration (GPI):

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



A geometric metaphor for convergence of GPI:



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)

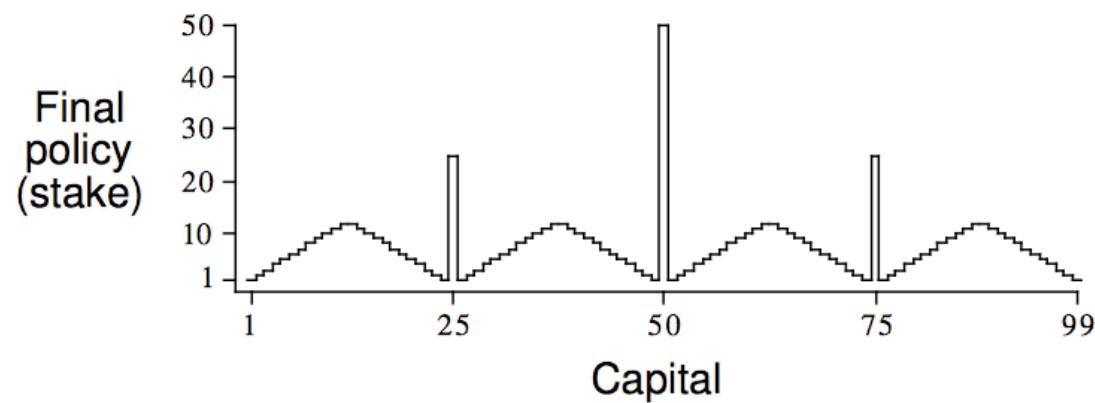
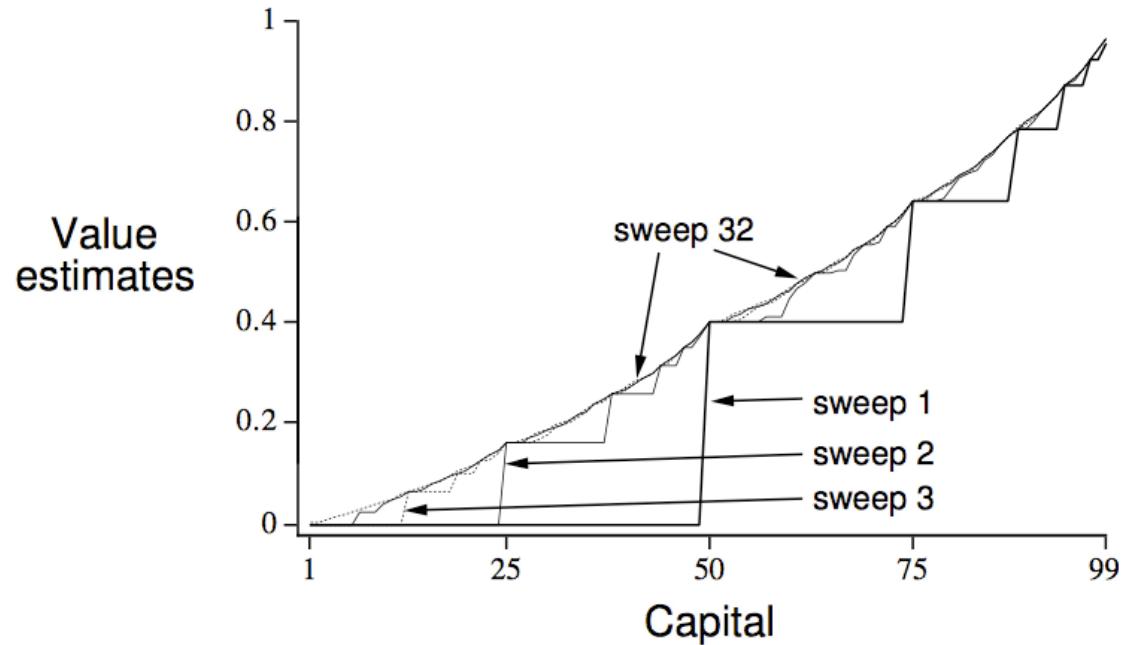
Output a deterministic policy, π , such that

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

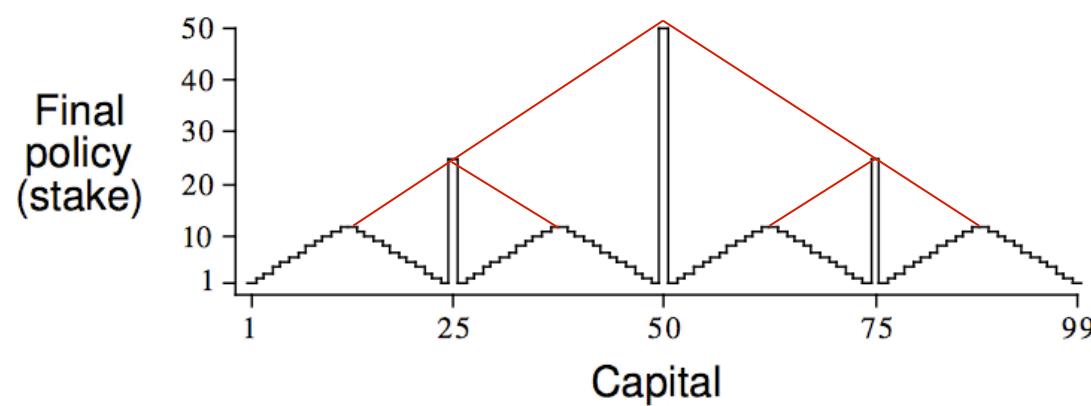
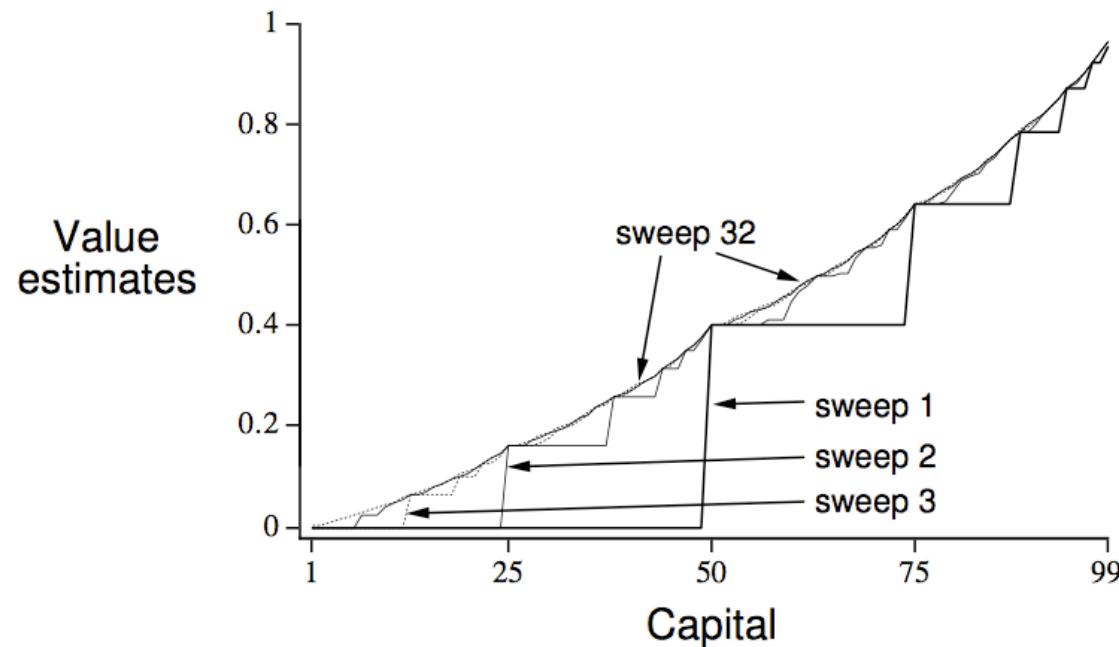
Gambler's Problem

- 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



Gambler's Problem Solution



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
- 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)