

Chapter 4: Dynamic Programming

Objectives of this lecture:

- ❑ 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
- ❑ Introduce idea of Generalized Policy Iteration (GPI)
- ❑ Discuss efficiency and utility of DP

Evaluation then Improvement

- First we cover *Policy Evaluation*
 - Given a policy π , what is π 's value function V^π ?
 - Also called the *prediction problem* since we're predicting the value of each state

- Then we cover *Policy Improvement*
 - Given V^π , can we get an improved policy π' ?
 - Also called the *control problem* since we learn a new policy to control an agent

Policy Evaluation

Policy Evaluation: for a given policy π compute the state-value function V^π

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

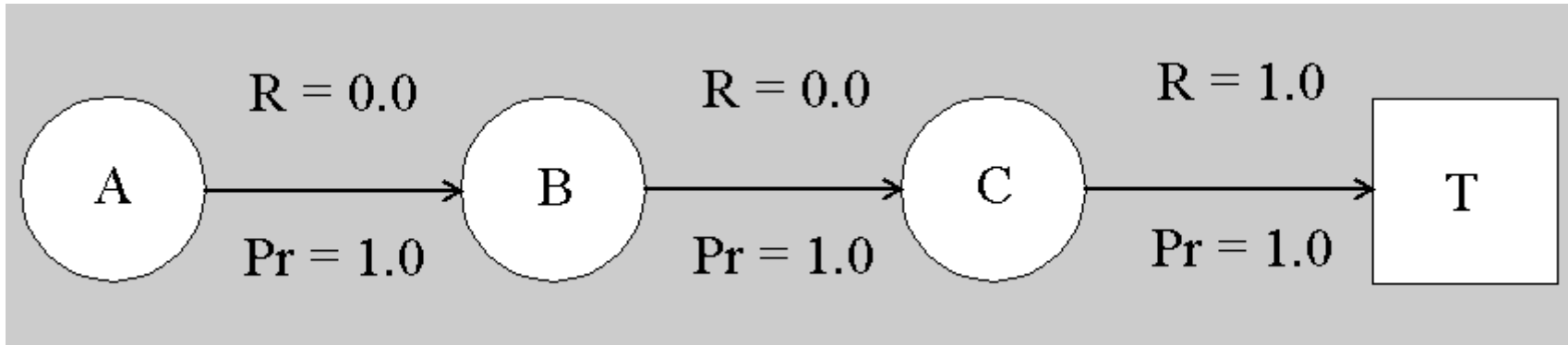
$$V^\pi(s) = E_\pi \{R_t \mid s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}$$

Bellman equation for V^π (deterministic policies):

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

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

Markov Process Example



Use the Bellman equation to calculate the value function for each state using $\gamma = 1.0$ and $\gamma = 0.9$. The terminal state has value 0

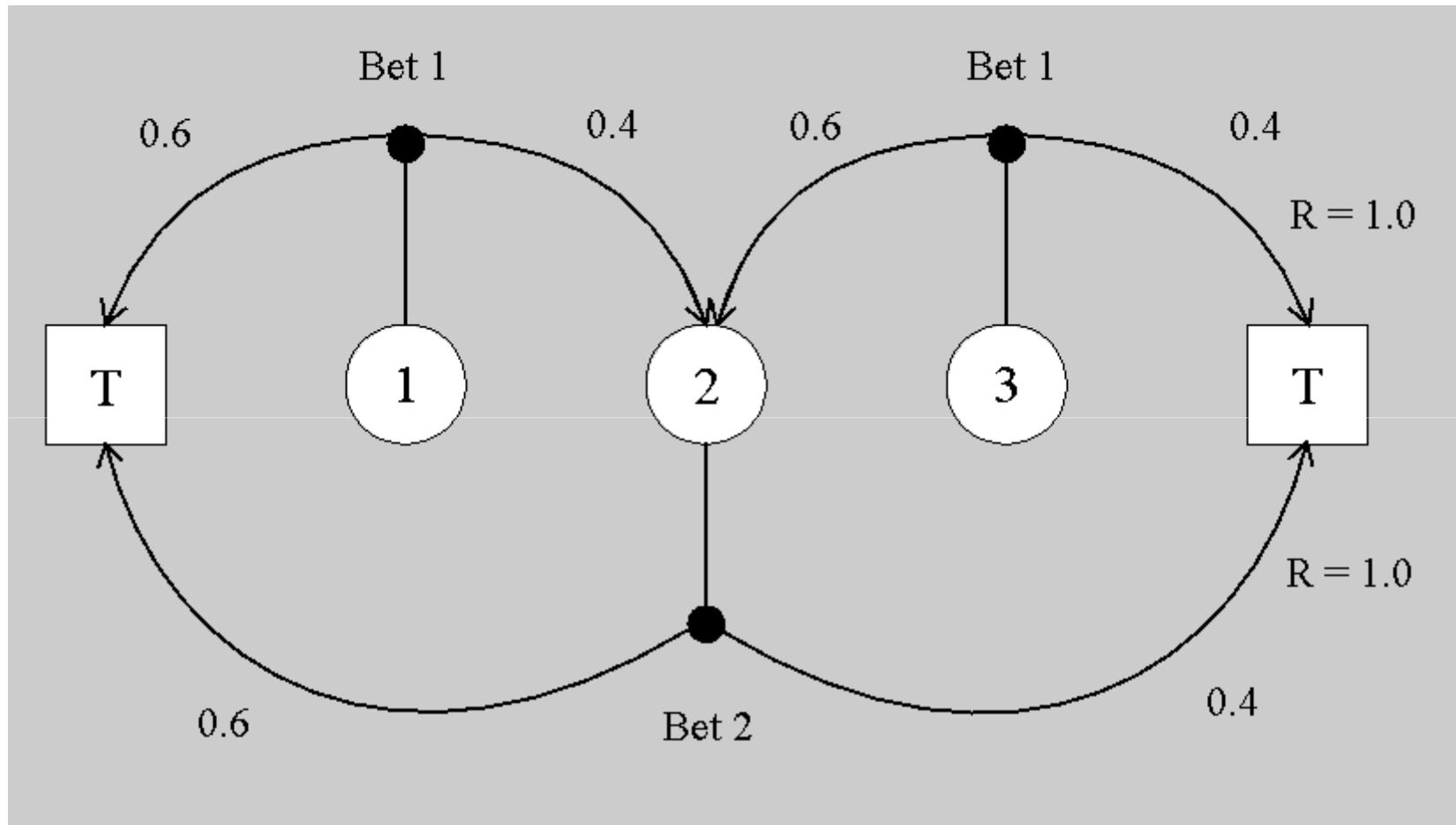
$$\begin{aligned} V(c) &= \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] \\ &= P_{CT} [R_{CT} + 1.0V(T)] \\ &= \end{aligned}$$

Compute the Value Function for the MP

$$\begin{aligned} V(c) &= \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] & V(a) &= \\ &= P_{CT} [R_{CT} + \gamma V(T)] \\ &= 1[1.0 + 1.0(0)] \\ &= 1 \end{aligned}$$

$$V(b) =$$

MDP Example: 3-State Gambler's Problem



- Find the value function for the policy which bets 1, 2 and 1 in states 1, 2 and 3 respectively using $\gamma = 1.0$

Compute the Value Function for the MDP

$$V^\pi(3) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')] \quad V^\pi(1) =$$
$$=$$

$$V^\pi(2) =$$

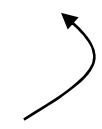
Iterative Computation

- Notice that the $V(1)$ and $V(3)$ depend on the value of $V(2)$
- In most MPs and MDPs there are many states whose values depend on each other. E.g. $V(X)$ depends on $V(Y)$ and vice versa.
 - I.e. We need to solve simultaneous equations
- One way to do this is to use an arbitrary initial value function and use the Bellman equation as a backup many times for each state
- Eventually V (estimated value function) will approximate V^π (true value function)

Synchronous Iterative Methods

A series of value functions:

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

a “sweep” 

A sweep consists of applying a **backup operation** to each state.
Note: no exploration issues – we do all states. Simple :)

A **full policy evaluation backup** (stochastic policies):

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

Iterative Policy Evaluation

Input π , the policy to be evaluated

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

Repeat

$\Delta \leftarrow 0$

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

$v \leftarrow V(s)$

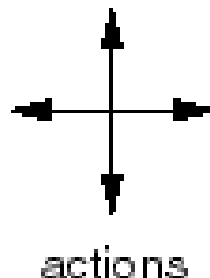
$V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

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

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

Output $V \approx V^\pi$

A Small Gridworld



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

$r = -1$
on all transitions

- 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

□ An example of iterative policy evaluation

□ $V_k =$ value function

after k sweeps for the random policy

□ A state's value converge to average number of steps when following π from state to goal (since $r = -1$ on each step)

$k = 0$

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

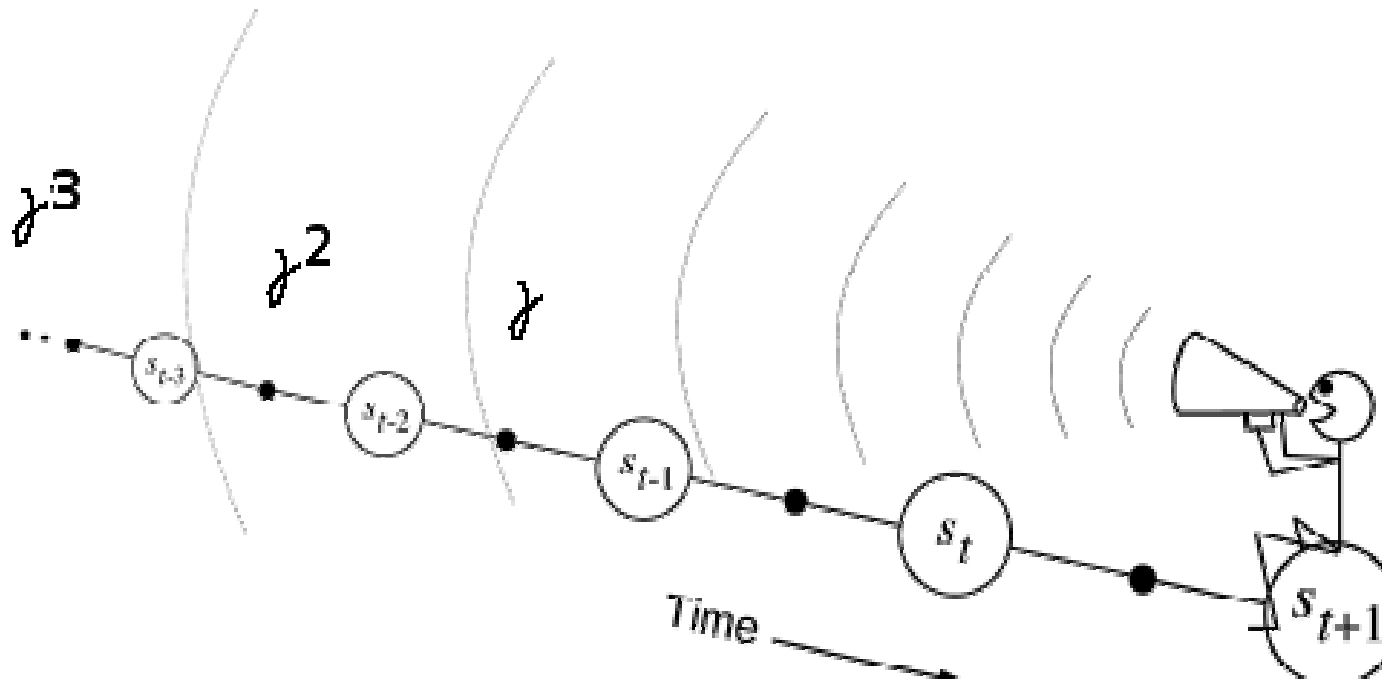
$k = 10$

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

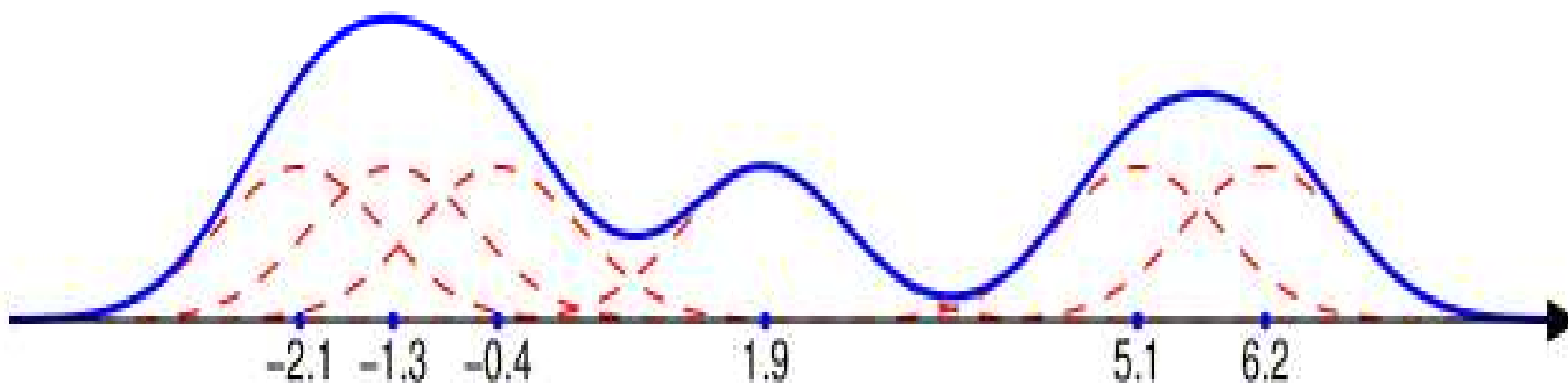
Effect of Backups



- ❑ Backups shift value backwards in time
- ❑ Magnitude decreases the farther it goes
 - γ controls rate of decrease
 - Sound is a good analogy

Value is a bit like Sound

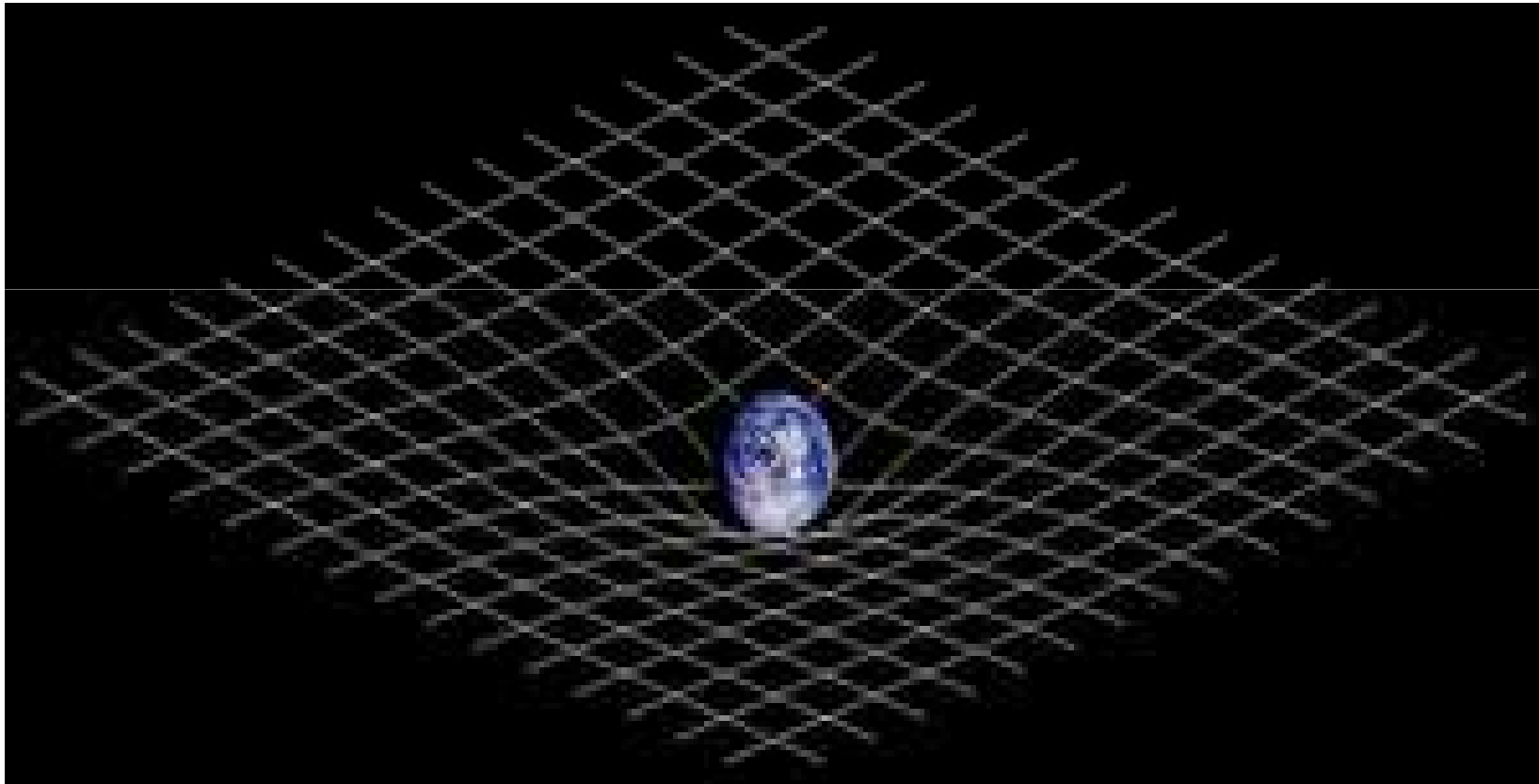
- 1-dimensional continuous space
 - Reward function has 6 non-zero rewards at -2.1, -1.3 ...
 - Dashed lines show contribution of each reward to value
 - Solid line shows value function



- Now imagine each reward is an alarm
 - Dashed lines show a contribution to total noise
 - Solid line shows total noise

Value is also like a surface

- Rewards are weights which cause the surface to stretch



Policy Improvement

Policy Improvement

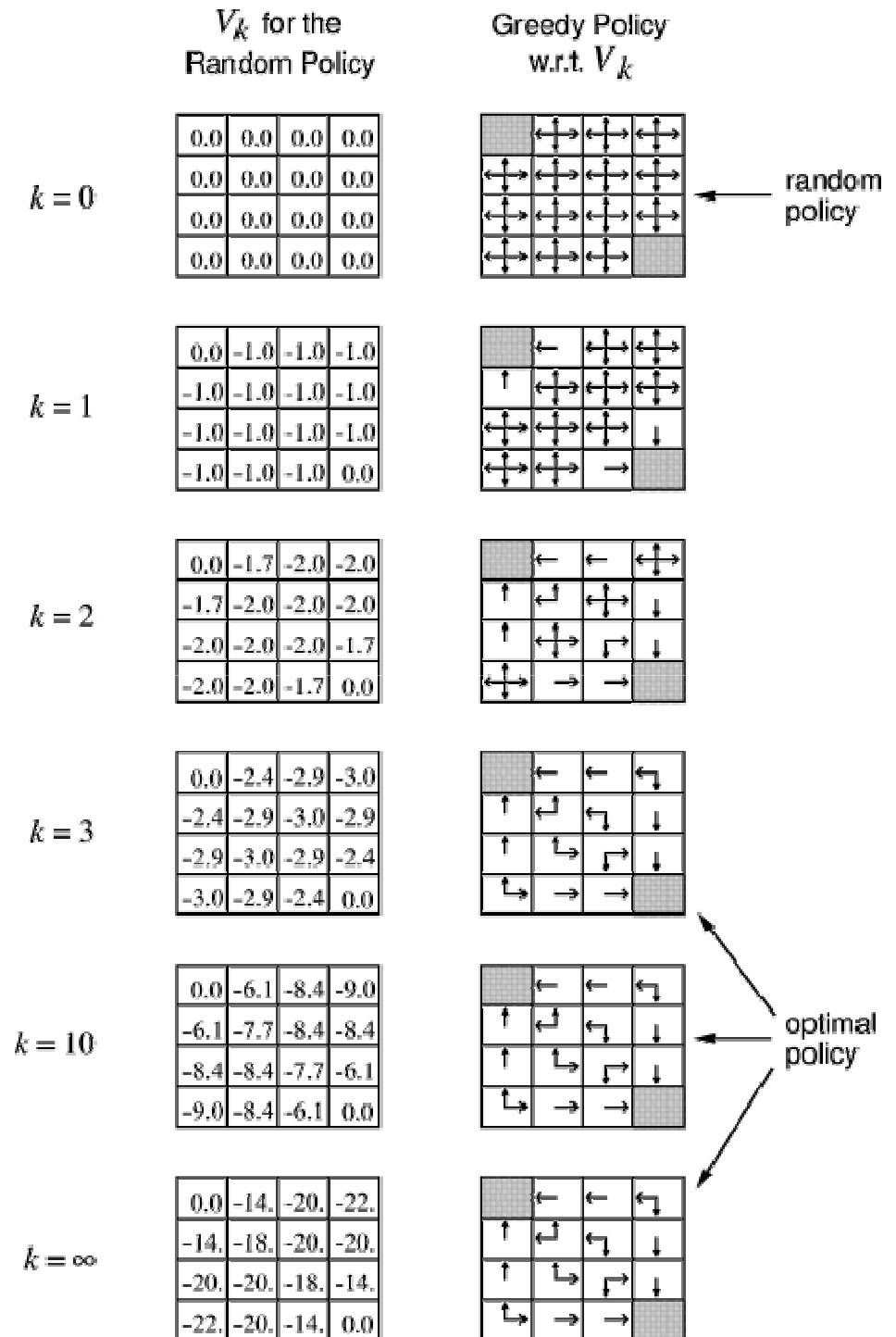
- Suppose we have computed V^π for a deterministic policy π
- For any state, is it better to take a different action to the one specified by π ?
- To maximise the expected return take the greedy action
 - In other words, compare the action-values for all actions in the state and use the greedy action for the new policy

$$\pi'(s) = \arg \max_a Q^\pi(s, a)$$

- Do this for all states to get the new policy π'
 - We say π' is greedy with respect to V^π
- Now the value of each state using π' is at least as much as using π

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

- Here we see the greedy policy for each value function
- Note the greedy policy is the optimal policy long before the value estimates are accurate
 - This is typical



Policy Improvement: Where will it end?

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

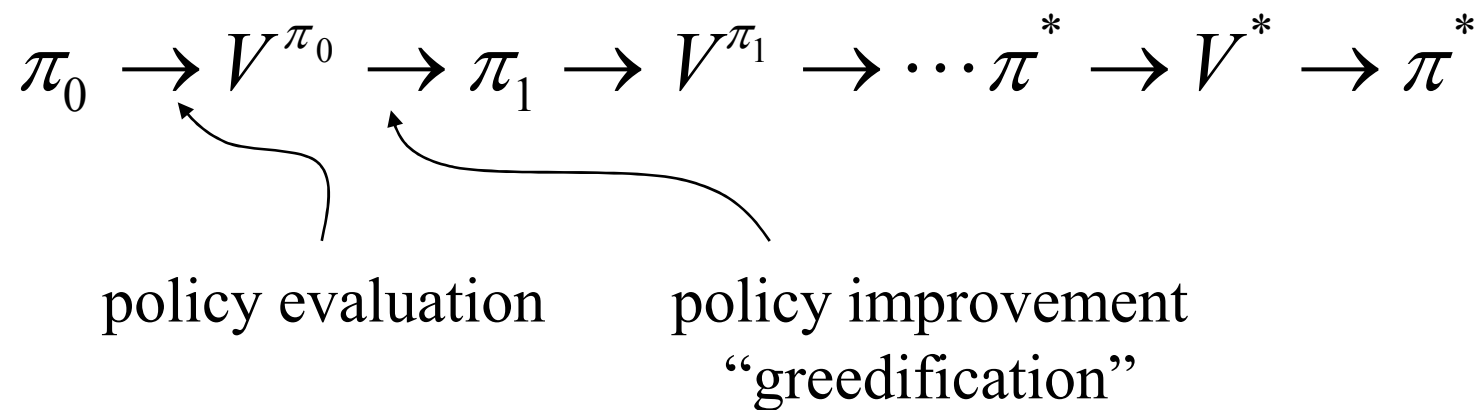
i.e., for all $s \in S$,
$$V^{\pi'}(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')] ?$$

But this is the Bellman Optimality Equation.

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

Policy Iteration

A series of policies and their value functions:



Policy Iteration

Algorithm for deterministic policies only

1. Initialization

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

2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

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

$v \leftarrow V(s)$

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

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

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 **policy evaluation backup** (for deterministic policies):

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

And greedy **policy improvement**:

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

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

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

It combines evaluation and improvement in 1 update.

It's policy evaluation with a max added for improvement

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, π , such that

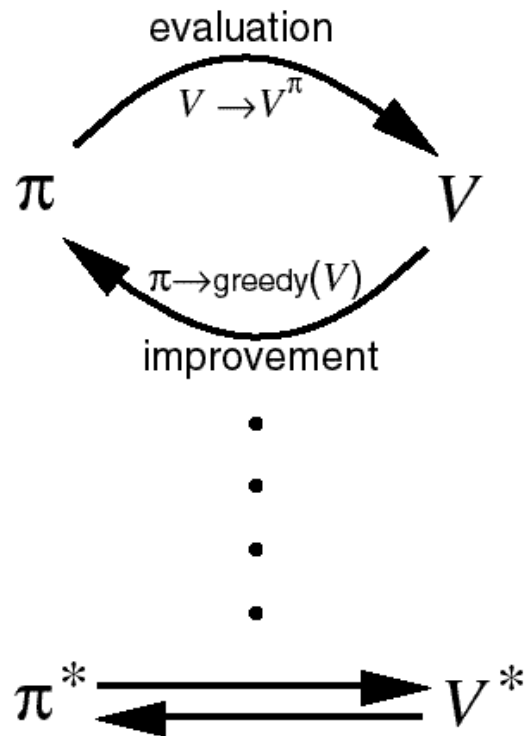
$$\pi(s) = \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$$

Asynchronous DP

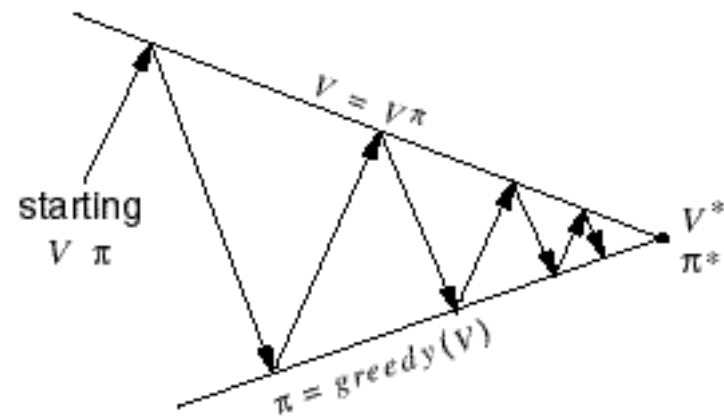
- ❑ All the DP methods described so far require exhaustive sweeps of the entire state set.
- ❑ Asynchronous DP does not use sweeps. Instead:
 - Repeat until convergence criterion is met:
 - Pick a state at random and apply the appropriate backup
- ❑ Still needs lots of computation, but does not get locked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

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 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
- ❑ 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