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 π, compute the state-value function V^π

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

$$V^\pi(s) = E\left\{R^\pi(s)\right\} = E\left\{\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right\}$$

Bellman equation for V^π :

$$V^\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^\pi(s')\right]$$

— 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.

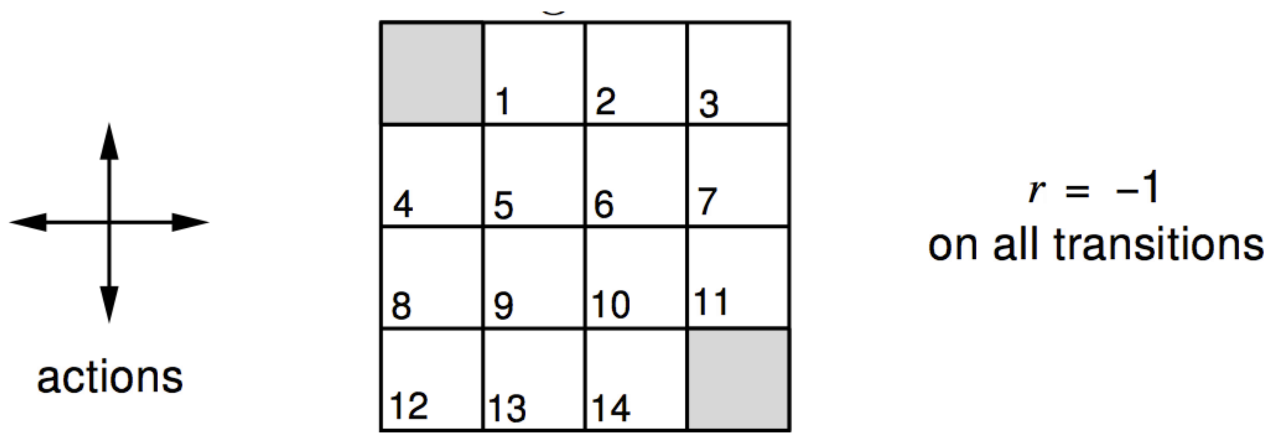
A **full policy-evaluation backup**:

$$V_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right]$$

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 $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
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

$\pi =$ equiprobable random action choices

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

$k = 3$

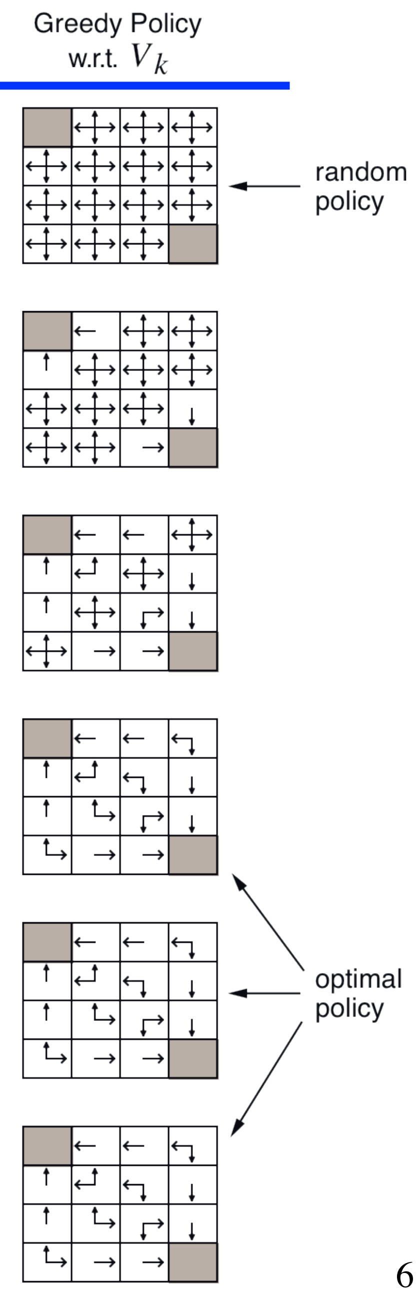
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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

$k =$ °

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



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 $a \neq \pi(s)$?

The value of doing a in state s is:

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

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

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

Policy Improvement Theorem

- Let π and π' be any pair of deterministic policies such that

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \quad , \quad \forall s \in \mathcal{S}$$

Then the policy π' must be as good as, or better than π

$$\boxed{V^{\pi'}(s) \geq V^{\pi}(s) \quad , \quad \forall s \in \mathcal{S}}$$

Policy Improvement Cont.

Do this for all states to get a new policy π' that is **greedy** with respect to V^{π} :

$$
\begin{aligned}
\pi'(s) &= \arg\max_a Q^{\pi}(s,a) \\
&= \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a}\left[\mathcal{R}_{ss'}^{a} + \gamma V^{\pi}(s')\right]
\end{aligned}
$$

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

Policy Improvement Cont.

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

i.e., for all $s \in S$, $V^{\pi'}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s') \right]$?

But this is the Bellman Optimality Equation.

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

Policy Iteration

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \cdots \pi^* \rightarrow V^* \rightarrow \pi^*$$

policy evaluation

policy improvement
"greedification"

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

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $b \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
   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**:

$$V_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right]$$

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

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right]$$

Value Iteration Cont.

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

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

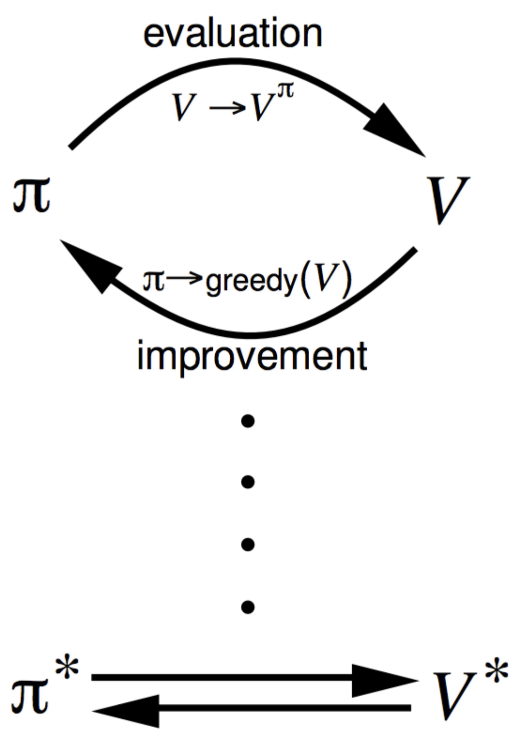
Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Asynchronous DP

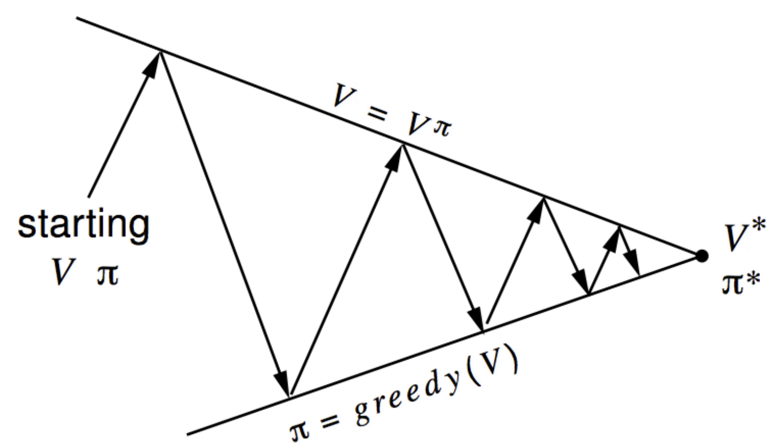
- All the DP methods described so far require exhaustive sweeps of the entire state set.
- Asynchronous DP does not use sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - Pick a state at random and apply the appropriate backup
- Still need lots of computation, but 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:



Linear Programming

- Since $\lim_{k\to\infty} T^k V = V^*$ for all V, we have

$$\mathcal{T}V \leq V \quad \longrightarrow \quad V^* = \mathcal{T}V^* \leq V$$

- Thus V^* is the smallest V that satisfies the constraint $\mathcal{T}V \leq V$

$$\min_V \sum_{s\in\mathcal{S}} V(s)$$

$$\text{subject to} \quad V(s) \quad \geq \quad r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V(s'), \qquad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

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.

Efficiency of DP and LP

- Total number of deterministic policies $|\mathcal{A}|^{|\mathcal{S}|}$

- **DP** methods are polynomial time algorithms
  - **VI (each iteration)** $O\left(|\mathcal{S}|^2|\mathcal{A}|\right)$
  - **PI (each iteration)** = *the cost of policy evaluation* + *the cost of policy improvement*

    Linear system of equations $O\left(|\mathcal{S}|^3\right)$ or $O\left(|\mathcal{S}|^{2.807}\right)$

    iterative $O\left(|\mathcal{S}|^2 \frac{\log(1/\theta)}{\log(1/\gamma)}\right)$

    $O\left(|\mathcal{S}|^2|\mathcal{A}|\right)$

  - Each iteration of **PI** is computationally more expensive than each iteration of **VI**
  - **PI** typically require fewer iterations to converge than **VI**
  - Exponentially faster than any direct search in policy space
  - Number of states often grows exponentially with the number of state variables

Efficiency of LP

- **LP** methods
  - Their worst-case convergence guarantees are better than those of **DP** methods
  - Become impractical at a much smaller number of states than do **DP** methods

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