Reinforcement Learning

11-785, Spring 2019

Defining MDPs, Planning

QUESTIONS before we dive?

Planning with an MDP

- Problem:
  - **Given:** an MDP $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$
  - **Find:** Optimal policy π_*

- Can either
  - **Value-based Solution:** Find optimal value (or action value) function, and derive policy from it OR
  - **Policy-based Solution:** Find optimal policy directly

Value-based Planning

- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Value-based Planning

- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Preliminaries

- How do we represent the value function?
- Table:
  - Value function
    - $s \rightarrow v_\pi(s)$
    - For a process with N discrete states, must store/compute N unique values
  - Action value functions
    - $s, a \rightarrow q_\pi(s, a)$
    - For a process with N discrete states and M discrete actions, must store/compute NM unique values
- Later we will see how to represent these when the number of states/actions is too large or continuous

The Bellman Expectation Equation for the value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

- In vector form

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix} = \begin{bmatrix} R_{s_1} \\ R_{s_2} \\ \vdots \\ R_{s_N} \end{bmatrix} + \gamma \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ P_{s_1,s_N} & P_{s_2,s_N} & \cdots & P_{s_N,s_N} \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix}$$

- Where
  - $R_s = \sum_{a \in \mathcal{A}} \pi(a|s) R_s^a$
  - $P_{s',s} = \sum_{a \in \mathcal{A}} \pi(a|s) P_{s',s}^a$

The Bellman Expectation Equation for the value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

- In vector form

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix} = \begin{bmatrix} R_{s_1} \\ R_{s_2} \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ & & & P_{s_N,s_N} \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix}$$

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Where
  - $R_s = \sum_{a \in \mathcal{A}} \pi(a|s) R_s^a$
  - $P_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} P_{s,s'}^a$

Solving the MDP

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Given the expected rewards at every state, the transition probability matrix, the discount factor and the policy:

$$\mathcal{V}_\pi = (\mathrm{I} - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi$$

- Easy for processes with a small number of states
- Matrix inversion $O(N^3)$; intractable for large state spaces

What about the action value function?

- The Bellman expectation equation for action value function

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \sum_{a \in \mathcal{A}} \pi(a|s') q_\pi(s',a)$$

$$Q_\pi = \mathcal{R}_{\pi,Q} + \gamma \mathcal{P}_{\pi,Q} Q_\pi$$

$NM \times 1$ $NM \times 1$ $NM \times NM$ $NM \times 1$

Even worse!!

So how do we solve these

- The equations are too large, how do we solve them?

- First, a little lesson – from middle school…

What they never taught you in school

- Consider the following equation:

$$ax = b$$

- Where $0 < a < 2$
- Trivial solution: $x = a^{-1}b = \frac{b}{a}$

- But my CPU does not permit division..
  - How do I solve this?

What they never taught you in school

- Must solve the following without division

$$ax = b$$

  - where $0 < a < 2$
- Rewrite as follows

$$x = (1 - a)x + b$$

- The following iteration solves the problem:

$$x^{(k+1)} = (1 - a)x^{(k)} + b$$

- Can start with any $x^{(0)}$
- Proof??

What they never taught you in school

- Must solve the following without division

$$ax = b$$

  - where $0 < a < 2$
- Rewrite as follows

$$x = (1 - a)x + b$$

- The following iteration solves the problem:

$$x^{(k+1)} = (1 - a)x^{(k)} + b$$

- Can start with any $x^{(0)}$
- Proof?? Hint: $0 < a < 2 \;\Rightarrow\; |1 - a| < 1$

What they never taught you in school

- Consider any vector equation

$$\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{b}$$

  - Where all Eigen values $|\lambda(\mathbf{A})| \leq 1$
    - And some extra criteria…
      - The square submatrix of $(\mathbf{I} - \mathbf{A})$ corresponding to non-zero entries of $\mathbf{b}$ is full rank
      - The square submatrix of $(\mathbf{I} - \mathbf{A})$ corresponding to zero entries of $\mathbf{b}$ is an identity matrix

- The following iteration solves the problem:

$$\mathbf{x}^{(k+1)} = \mathbf{A}\mathbf{x}^{(k)} + \mathbf{b}$$

Eigen values of a probability matrix

- For any Markov transition probability matrix $\mathcal{P}$, all Eigenvalues have magnitude less than or equal to 1

$$|\lambda(\mathcal{P})| \leq 1$$

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

- But how did that help if we need infinite iterations to converge?

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

- But how did that help if we need infinite iterations to converge?
  - Solution: Stop when the changes becomes small

$$\left| \mathcal{V}_\pi^{(k+1)} - \mathcal{V}_\pi^{(k+1)} \right| < \varepsilon$$

Actual Implementation

- Initialize $v_\pi^{(0)}(s)$ for all states
- Update

$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi^{(k)}(s') \right)$$

- Update may be in *batch* mode
  - Keep sweep through all states to compute $v_\pi^{(k+1)}(s)$
  - Update $k = k + 1$
- Or incremental
  - Sweep through all the states performing

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

Actual Implementation

- Initialize $v_\pi^{(0)}(s)$ for all states
- Update

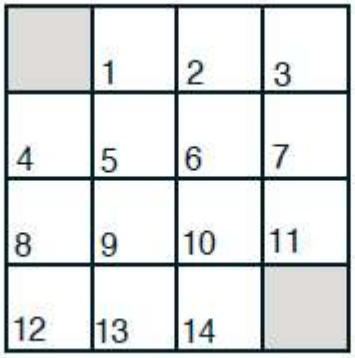
$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi^{(k)}(s') \right)$$

- This is an instance of *dynamic programming:*
- **dynamic programming** (also known as **dynamic optimization**) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions. The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space. (Each of the subproblem solutions is indexed in some way, typically based on the values of its input parameters, so as to facilitate its lookup.) (from wikipedia)

An Example

Example from Sutton

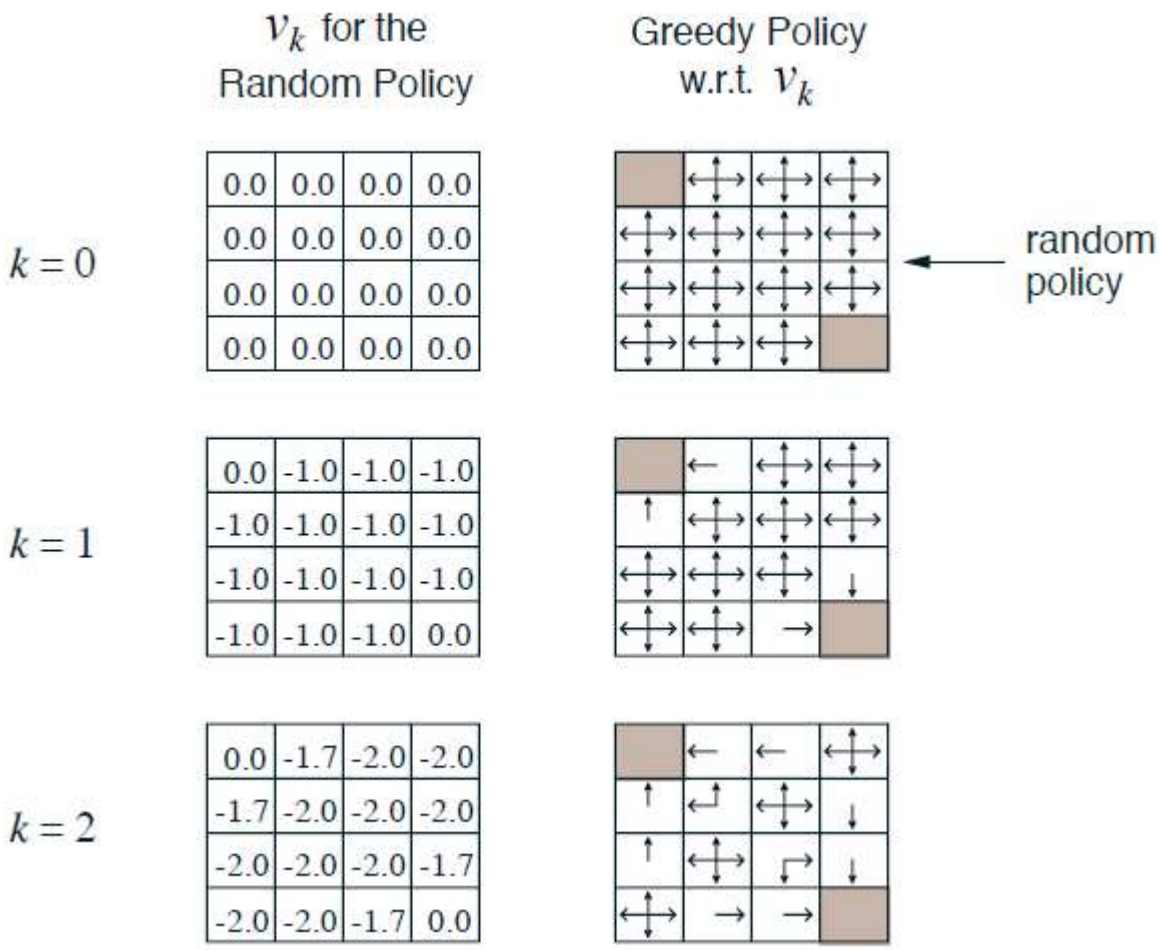
actions



$R = -1$
on all transitions

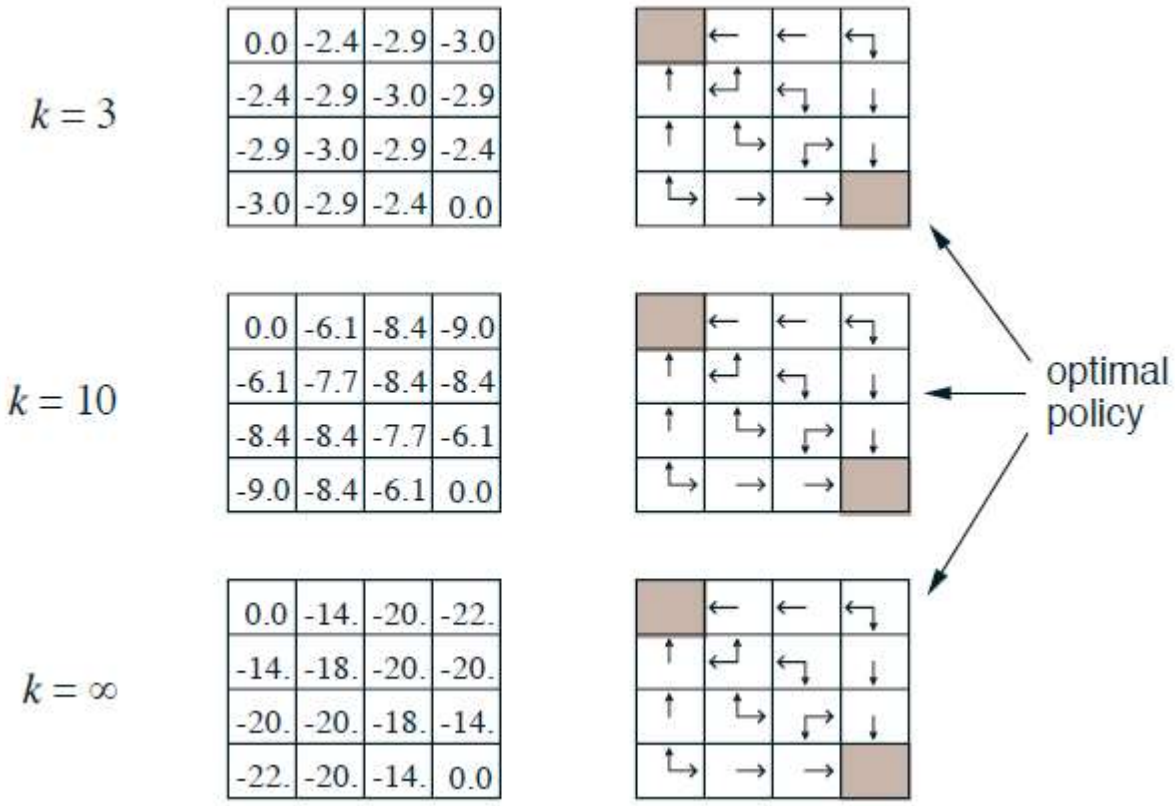
- All squares, except shaded square have reward -1, shaded square has reward 0
- **Policy:** Random – can step in any of the four directions with equal probability
  - If you run into a wall, you just return to the square
- Find the value of being in each square

The Gridworld Example



- Actual iterations use random policy
- Right column shows greedy policy according to current value function

The Gridworld Example



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Value-based Planning

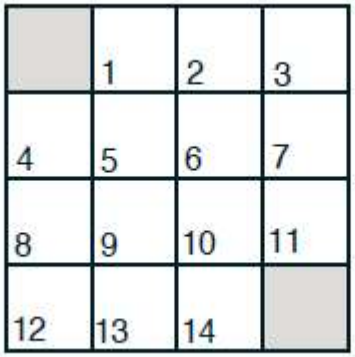
- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Revisit the gridworld

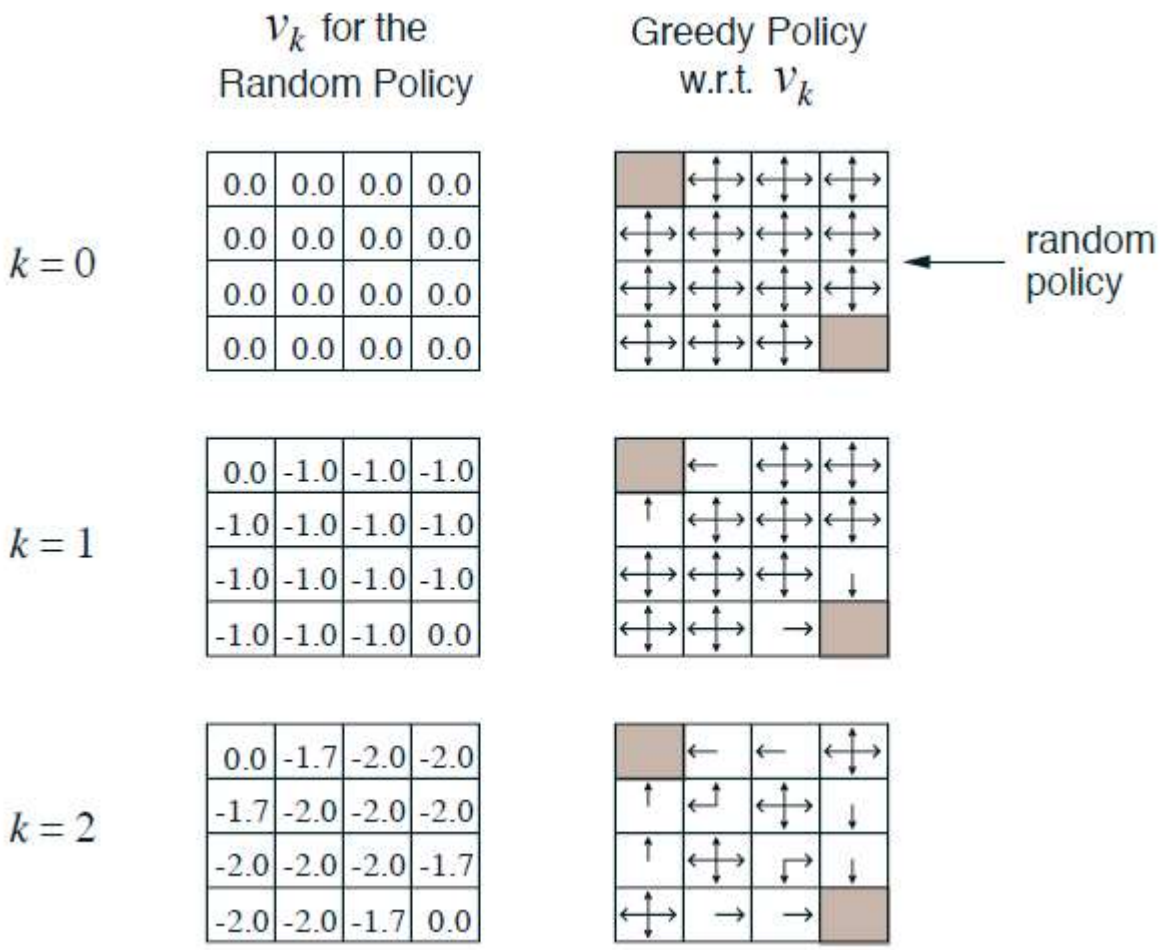
Example from Sutton

actions



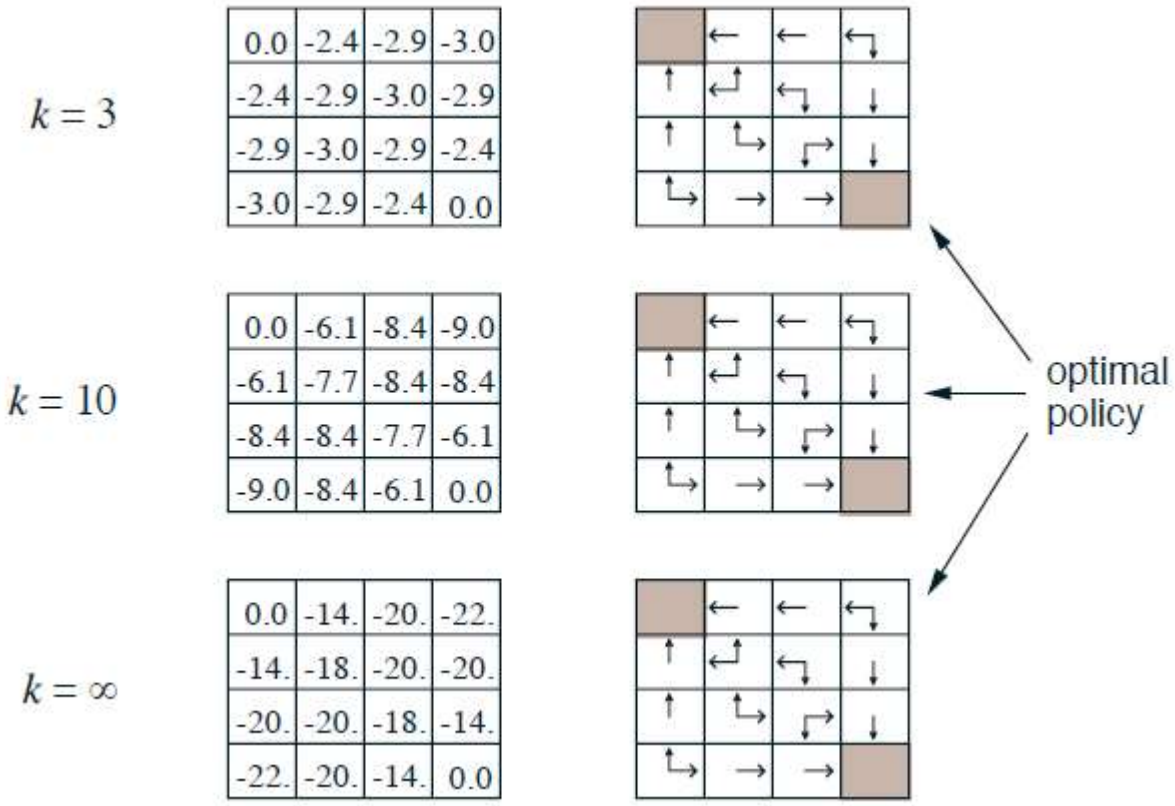
$R = -1$
on all transitions

Revisit the gridworld



- Actual iterations use random policy
- Right column shows greedy policy according to current value function

Revisit the gridworld



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Finding an optimal policy

- Start with any policy, e.g. random policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Compute action value function $\forall s, a$:

$$q_{\pi^{(k)}}(s,a) = R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_{\pi^{(k)}}(s')$$

  - Find the greedy policy

$$\pi^{(k+1)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a\prime}{\text{argmax}} \;\; q_{\pi^{(k)}}(s,a') \\ 0 \qquad\qquad otherwise \end{cases}$$

Finding an optimal policy: Compact

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a'}{\operatorname{argmax}} \;\; R_s^{a'} + \gamma \sum_{s'} P_{s,s'}^{a'} v_{\pi^{(k)}}(s') \\ 0 \qquad\qquad otherwise \end{cases}$$

Finding an optimal policy: Shorthand

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(s) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

THIS IS KNOWN AS **POLICY ITERATION**
In each iteration, we find a policy, and then find its value

Policy Iteration

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Generalized Policy Iteration

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use *any algorithm* to find the value function $v_{\pi^{(k)}}(s)$
  - Use *any algorithm* to find an update policy

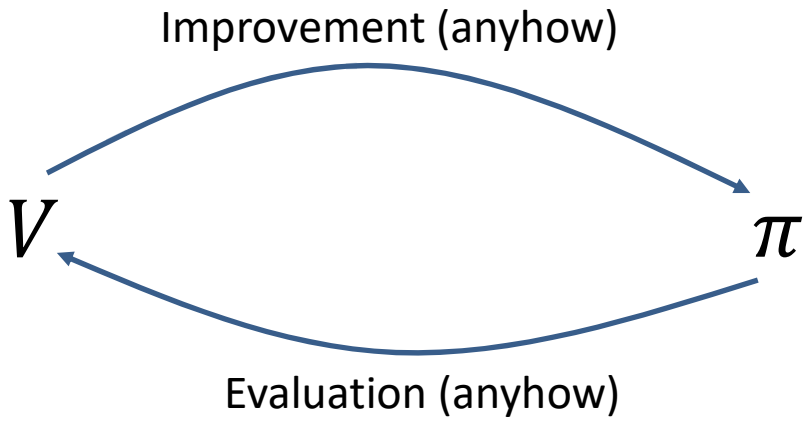
$$\pi^{(k+1)}(\mathrm{s}) = algorithm\left(v_{\pi^{(k)}}(s)\right)$$

Such that $\pi^{(k+1)}(\mathrm{s}) \geq \pi^{(k)}(\mathrm{s})$

- Guaranteed to converge to the optimal policy

Generalized Policy Iteration

- Start with any policy $\pi^{(0)}$



- Guaranteed to converge to the optimal policy

Optimality theorem

- *All* states will hit their optimal value together

- **Theorem:**

  A policy $\pi(a|s)$ has optimal value

  $$v_\pi(s) = v_*(s)$$

  in any state s if and only if for *every* state s' reachable from s,

  $$v_\pi(s') = v_*(s')$$

Policy Iteration

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Policy Iteration

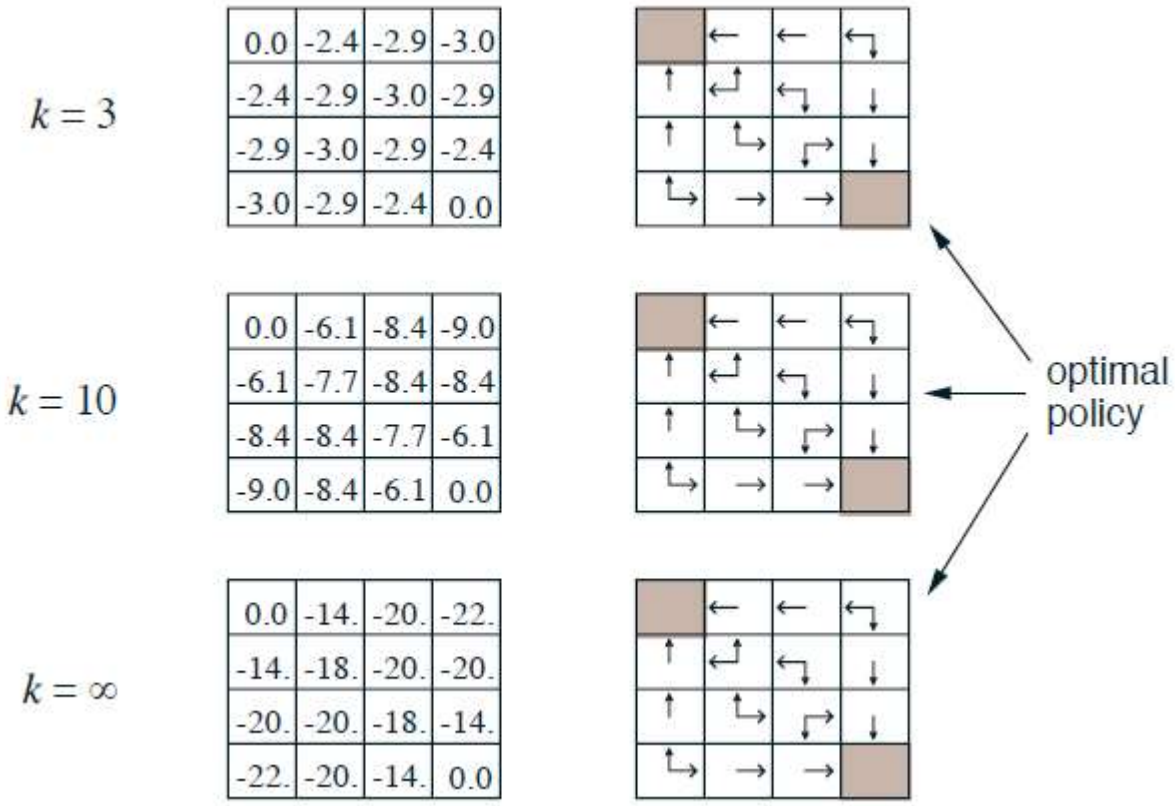
- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

In the gridworld example we didn't even need to run *this* to convergence

The optimal policy was found long before the actual value function converged even in the first upper iteration

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Revisit the gridworld



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Policy Iteration

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(s) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

In the gridworld example we didn't even need to run *this* to convergence

The optimal policy was found long before the actual value function converged even in the first upper iteration

Do we even need the prediction DP to converge?

- More generally, it will take several iterations
  - Convergence when policy no longer changes

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use L iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s') \right)$$

  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = \underset{a}{\operatorname{argmax}} \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s')$$

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

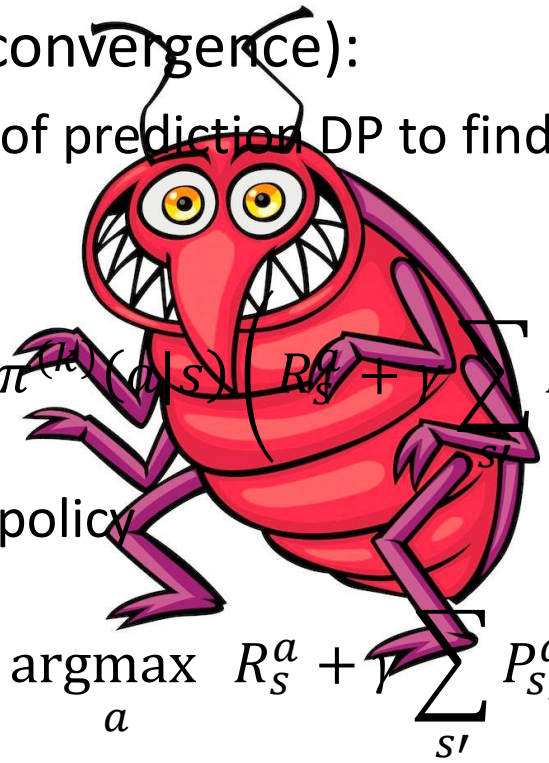
- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

BUG

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s') \right)$$

  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = \operatorname*{argmax}_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s')$$



Reordering and writing carefully

- Start with any initial value function $v_{\pi^{(0)}}(s)$
- Iterate ($k = 1$ … convergence):
  - Find the greedy policy

$$\pi^{(k)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a'}{\text{argmax}} \;\; R_s^{a'} + \gamma \sum_{s'} P_{s,s'}^{a'} v_{\pi^{(k-1)}}(s') \\ 0 \qquad otherwise \end{cases}$$

  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k-1)}}(s') \right)$$

Merging

- Start with any initial value function $v_{\pi(0)}(s)$
- Iterate ($k = 1$ … convergence):
  - Update the value function

$$v_{\pi(k)}(s) = \max_a R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_{\pi(k-1)}(s')$$

- Note: no explicit policy estimation
  - Directly learns value
  - The subscript π is a misnomer

Value Iteration

- Start with any initial value function $v_*^{(0)}(s)$

- Iterate ($k = 1$ … convergence):
  - Update the value function

$$v_*^{(k)}(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*^{(k-1)}(s')$$

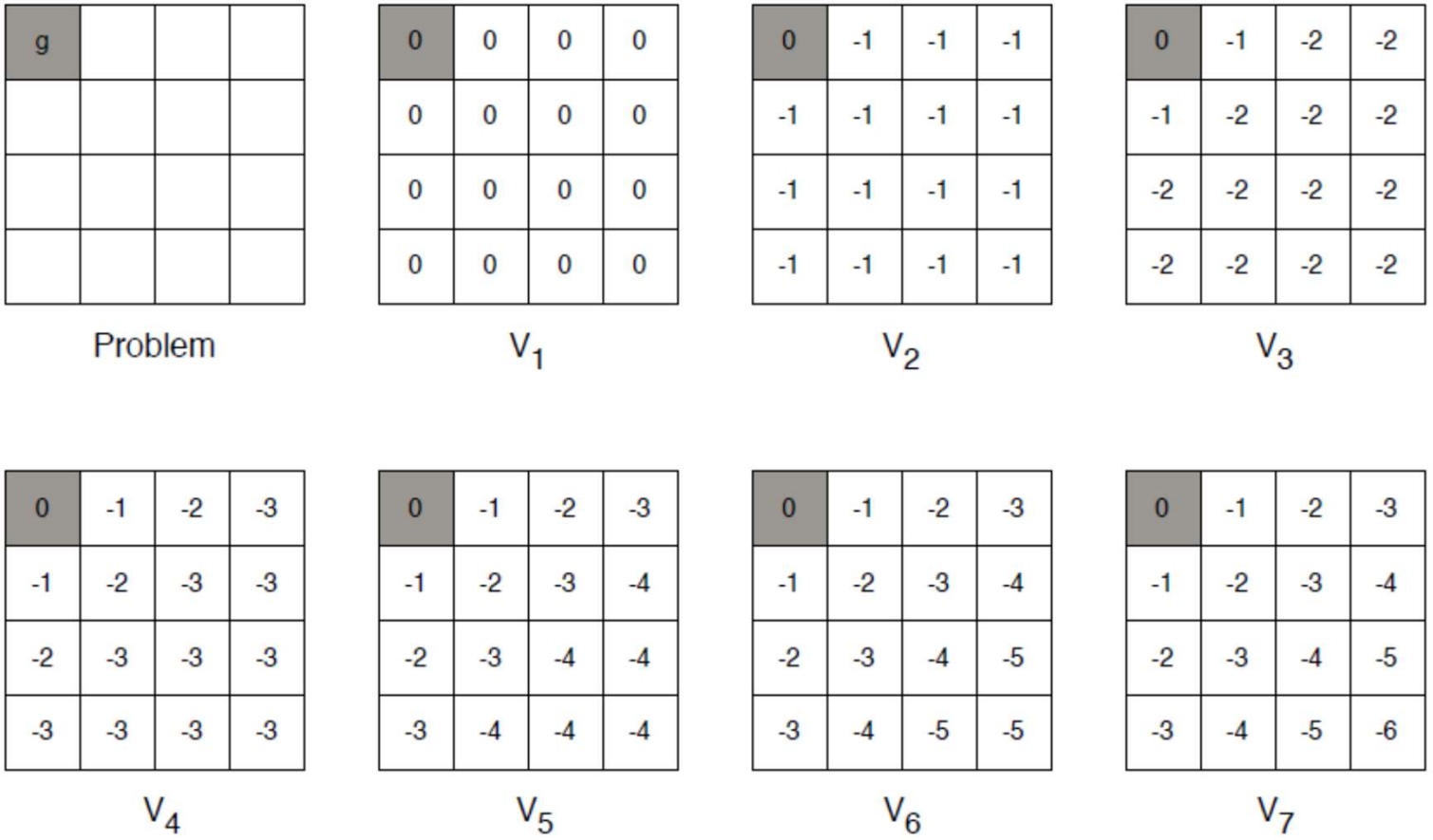
- Note: no explicit policy estimation
- Directly learning *optimal* value function
- Guaranteed to give you optimal value function at convergence
  - But intermediate value function estimates may not represent any policy

Value iteration

$$v_*^{(k)}(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*^{(k-1)}(s')$$

- Each state simply inherits the cost of its best neighbour state
  - Cost of neighbor is the value of the neighbour plus cost of getting there

Value Iteration Example



- Target: Find the shortest path
- Every step costs -1

Practical Issues

- Updates can be batch mode
  - Explicitly compute $v_*^{(k+1)}(s)$ from $v_*^{(k)}(s)$for all states
  - Set k = k+1

- Or asynchronous
  - Compute $v_*(s)$ in place while we sweep over states
  - $v_*(s) \leftarrow \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s)$

Recap

- Learned about *prediction*
  - Estimating value function given MDP and policy
- Learned *Policy* iteration
  - Iterate prediction and policy estimation
- Learned about *Value* iteration
  - Directly estimate optimal value function

Alternate strategy

- Worked with *Value function*
  - For N states, estimates N terms
- Could alternately work with *action-value function*
  - For M actions, must estimate MN terms
    - Much more expensive
    - But more useful in some scenarios

Next Up

- We've worked so far with planning
  - Someone gave us the MDP
- Next: Reinforcement Learning
  - MDP unknown..