Dynamic Programming

Reading

- Sutton, Richard S., and Barto, Andrew G. Reinforcement learning: An introduction. MIT press, 2018.
  - http://www.incompleteideas.net/book/the-book-2nd.html
  - Chapter 4
- Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
  - Chapter 4
- David Silver lecture on Dynamic Programming
  - https://www.youtube.com/watch?v=Nd1-UUMVfz4

Overview of Dynamical Programming

- A classical algorithm for computing solutions to problems that can be separated into subproblems with known solutions
  - E.g., all shortest paths
  - Essentially, store all subproblem solutions in a table and reuse them when necessary
- If we have a finite (state and action) MDP, we can find the optimal policy by incremental search
  - Find the optimal policy for 1 step, then 2 steps, etc.
  - Actually done backwards in time
- Polynomial complexity in the number of states
  - Number of states can be large

Policy Evaluation

- In order to find the best policy, we first need a way to evaluate policies
  - i.e., compute the state-value function $v_\pi(s)$ for each state s
  - Also compute the action-value function $q_\pi(s, a)$
- Every time we change the policy, we need to evaluate it (in order to check if we improved it)
- So far, we've seen one way to compute state values
  - How?
  - For a given policy π, compute the matrix form of the value function vector

Policy Evaluation: the infinite-horizon case

- In the infinite-horizon case, we can use the Bellman equation:

$$v_\pi(s) = R_\pi(s) + \gamma \sum_{a,s'} P(s,a,s')\pi(a|s)v_\pi(s')$$

  – which can be rewritten in matrix form:

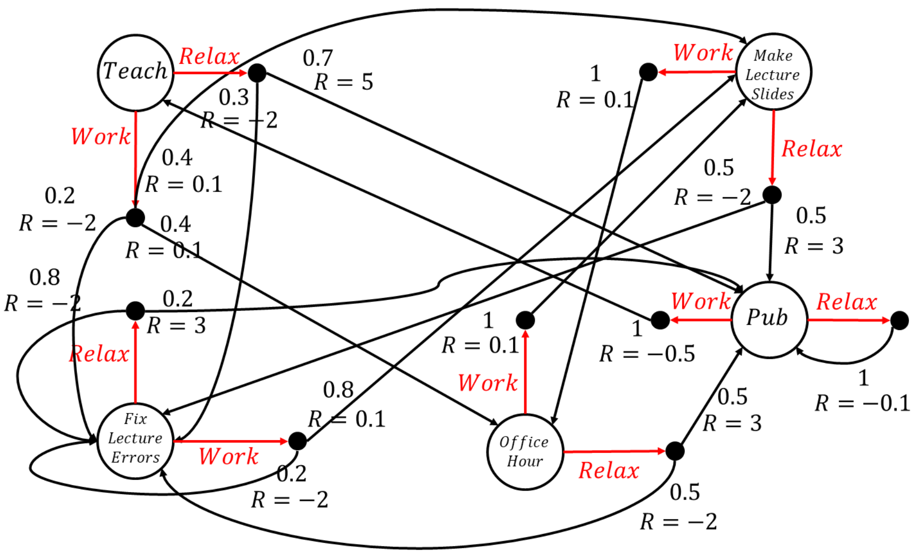
$$v_\pi(\boldsymbol{s}) = R_\pi(\boldsymbol{s}) + \gamma \boldsymbol{P}_\pi v_\pi(\boldsymbol{s})$$

- Thus, the state value vector is:

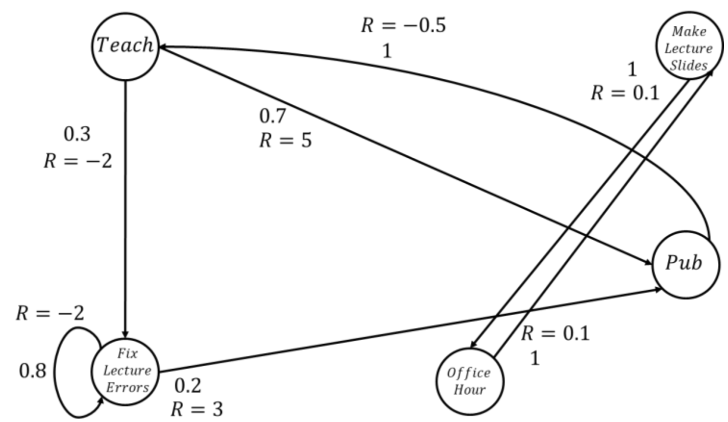
$$v_\pi(\boldsymbol{s}) = (\boldsymbol{I} - \gamma \boldsymbol{P}_\pi)^{-1} R_\pi(\boldsymbol{s})$$

Workday example, MDP -> MRP

MDP



MRP

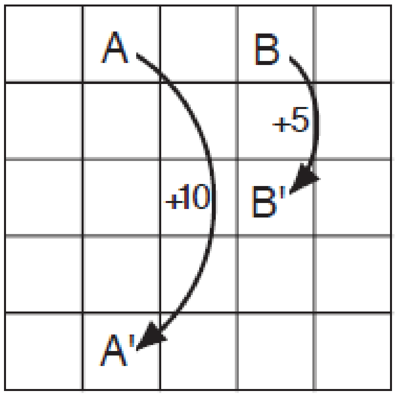


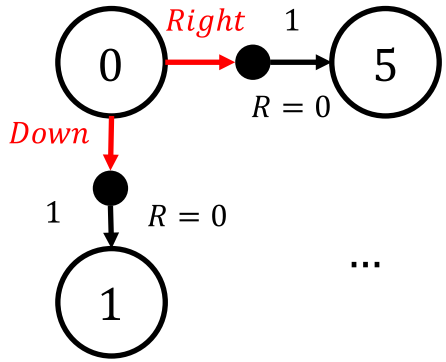
Policy

- Let's define π as follows:
  - $\pi(Teach) = Relax$
  - $\pi(OH) = Work$
  - $\pi(MLS) = Work$
  - $\pi(FLE) = Relax$
  - $\pi(Pub) = Work$

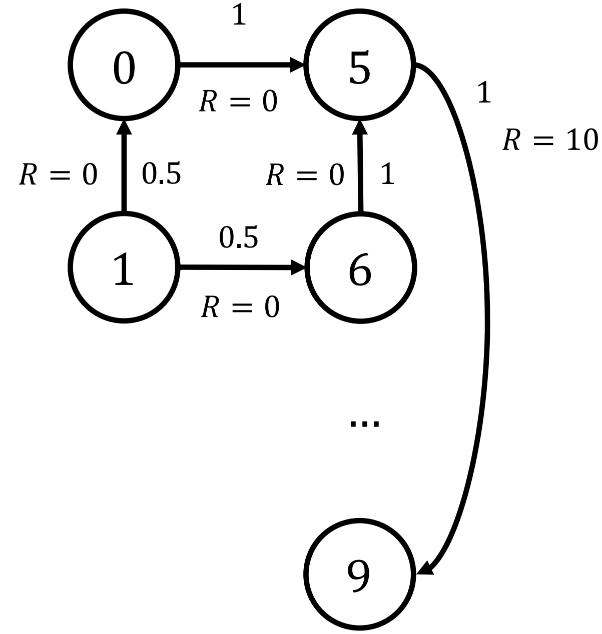
Gridworld example, MDP -> MRP

MDP

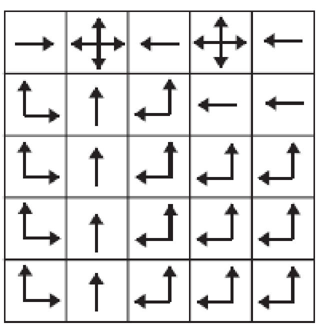




MRP



Policy



Iterative Policy Evaluation

- Inverting $\boldsymbol{I} - \gamma \boldsymbol{P}$ may be expensive if number of states is large
- Another approach is to use linear systems theory!
- Start from a random initialization for $v(\boldsymbol{s}) = v_0(\boldsymbol{s})$
- Look at linear system

$$v_k(\boldsymbol{s}) = R(\boldsymbol{s}) + \gamma \boldsymbol{P} v_{k-1}(\boldsymbol{s})$$

- System is stable. Why?
  - All entries (and eigenvalues) of $\gamma \boldsymbol{P}$ are < 1 (for $\gamma < 1$)
- System converges to unique solution $(\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} R(\boldsymbol{s})$
  - Why?
  - If $v_{k+1}(\boldsymbol{s}) = v_k(\boldsymbol{s}) = \boldsymbol{v}$, then $\boldsymbol{v} = R(\boldsymbol{s}) + \gamma \boldsymbol{P} \boldsymbol{v}$, i.e.,

$$\boldsymbol{v} = (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} R(\boldsymbol{s})$$

    - Keep in mind $\boldsymbol{I} - \gamma \boldsymbol{P}$ is not invertible when $\gamma = 1$

Workday example, Iterative Value evolution

- Recall state values are (for $\gamma = 0.9$)

$$(\boldsymbol{I} - \gamma\boldsymbol{P})^{-1}R(\boldsymbol{s}) = [5.54 \quad 1 \quad 1 \quad -0.69 \quad 4.49]^T$$

- Using iterative evaluation
  - Starting with $\boldsymbol{v}_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0]^T$

$$\boldsymbol{v}_{10} = [4.59 \quad 0.65 \quad 0.65 \quad -1.58 \quad 3.64]^T$$
$$\boldsymbol{v}_{30} = [5.43 \quad 0.96 \quad 0.96 \quad 0 - 0.80 \quad 4.38]^T$$
$$\boldsymbol{v}_{50} = [5.53 \quad 0.99 \quad 0.99 \quad 0 - 0.70 \quad 4.47]^T$$

- After 50 iterations, converged within 0.01 if the true values

Gridworld Example, iterative value evolution

Rensselaer

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

- Recall state values are
- Using iterative evaluation
  - Starting from $\boldsymbol{v}_0 = \mathbf{0} \in \mathbb{R}^{25}$
  - v_{10} is

14.31	15.90	14.31	10.90	9.81
12.88	14.31	12.88	11.59	10.44
11.59	12.88	11.59	10.44	5.90
10.44	11.59	10.44	5.90	5.31
5.90	10.44	5.90	5.31	4.78

  - v_{50} is

21.86	24.29	21.86	19.29	17.36
19.68	21.86	19.68	17.71	15.94
17.71	19.68	17.71	15.94	14.29
15.94	17.71	15.94	14.29	12.86
14.29	15.94	14.29	12.86	11.58

- After 50 iterations, converge within 0.15 of true values
- In general, can stop iterating when $||v_{k+1} - v_k|| \leq \epsilon$
  - Where ϵ is a hyperparameter

What about the finite-horizon case?

- Evaluate a policy recursively, starting from the last step T
  - Use the Bellman equation

$$v_\pi^t(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi^{t+1}(S_{t+1}) \middle| S_t = s\right]$$

- Remember, the policy and the value function may be time-dependent in the finite-horizon case!

Bellman Optimality Equation

- A policy π is better than another policy π' if
$$v_{\pi}(s) \geq v_{\pi'}(s), \forall s \in S$$
- A policy π^* is optimal if there exists no better policy than π^*
- Turns out the optimal policy also has a nice recursive property
$$\pi_*(s) = arg\max_a q_{\pi_*}(s,a)$$
  - Another version of the Bellman equation
  - Pick the action with the highest value
  - Obvious in a sense
    - But any policy that satisfies the Bellman optimality equation is optimal

Bellman Optimality Equation, proof

- Bellman optimality equation:

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

- Proof by induction backward in time (for finite horizon T):
  - Base case ($t = T - 1$):
    - Only one step to make
    - For any state s, the optimal action is

$$arg\max_a \mathbb{E}_*[R_T|S_{T-1} = s, A_{T-1} = a] =$$
$$= arg\max_a q_{\pi_*}(s,a)$$

    - So π_* is optimal by construction

Bellman Optimality Equation, proof

- Proof by induction backward in time (for finite horizon T):
  - Inductive case:
  - Assume π_* is optimal at time $t+1$
    - i.e., $v_{\pi_*}^{t+1}(s) \geq v_{\pi'}^{t+1}(s), \forall s, \pi'$
  - What do we need to show?

$$v_{\pi_*}^{t}(s) \geq v_{\pi'}^{t}(s), \forall s, \pi'$$

  - Using the Bellman equation for π_*:

$$\begin{aligned} v_{\pi_*}^{t}(s) &= q_{\pi_*}^{t}\big(s, \pi_*(s)\big) \\ &= \max_a \big[q_{\pi_*}^{t}(s,a)\big] \\ &= \max_a \Big[\mathbb{E}_{\pi_*}\big[R_{t+1} + \gamma v_{\pi_*}^{t+1}(S_{t+1}) \big| S_t = s, A_t = a\big]\Big] \end{aligned}$$

Bellman Optimality Equation, proof

- Proof by induction backward in time (for finite horizon T):
  - Assume π_* is optimal at time $t+1$
    - i.e., $v_{\pi_*}^{t+1}(s) \geq v_{\pi'}^{t+1}(s), \forall s, \pi'$
  - Consider any other policy π'

$$v_{\pi_*}^{t}(s) = \max_a \left[\mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_{\pi_*}^{t+1}(S_{t+1}) \middle| S_t = s, A_t = a \right] \right]$$

$$= \max_a \left[R_e(s,a) + \sum_{s'} \gamma v_{\pi_*}^{t+1}(s') P(s,a,s') \right]$$

$$\geq \max_a \left[R_e(s,a) + \sum_{s'} \gamma v_{\pi'}^{t+1}(s') P(s,a,s') \right]$$

  - Inequality true for any a, so true for max also

Bellman Optimality Equations, proof

- Proof by induction backward in time(for finite horizon T):

$$
\begin{aligned}
v_{\pi_*}^t(s) &= \max_a \left[\mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_{\pi_*}^{t+1}(S_{t+1}) \middle| S_t = s, A_t = a\right]\right] \\
&= \max_a \left[R(s,a) + \sum_{s'} \gamma v_{\pi_*}^{t+1}(s') P(s,a,s')\right] \\
&\geq \max_a \left[R(s,a) + \sum_{s'} \gamma v_{\pi'}^{t+1}(s') P(s,a,s')\right] \\
&\geq R\big(s, \pi'(s)\big) + \sum_{s'} \gamma v_{\pi'}^{t+1}(s') P(s, \pi'(s), s') \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_{\pi'}^{t+1}(S_{t+1}) \middle| S_t = s, A_t = \pi'(s)\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_{\pi'}^{t+1}(S_{t+1}) \middle| S_t = s\right] \\
&= v_{\pi'}^t(s)
\end{aligned}
$$

Deterministic
policy version

Bellman Optimality Equations, proof

- Proof by induction backward in time(for finite horizon T):

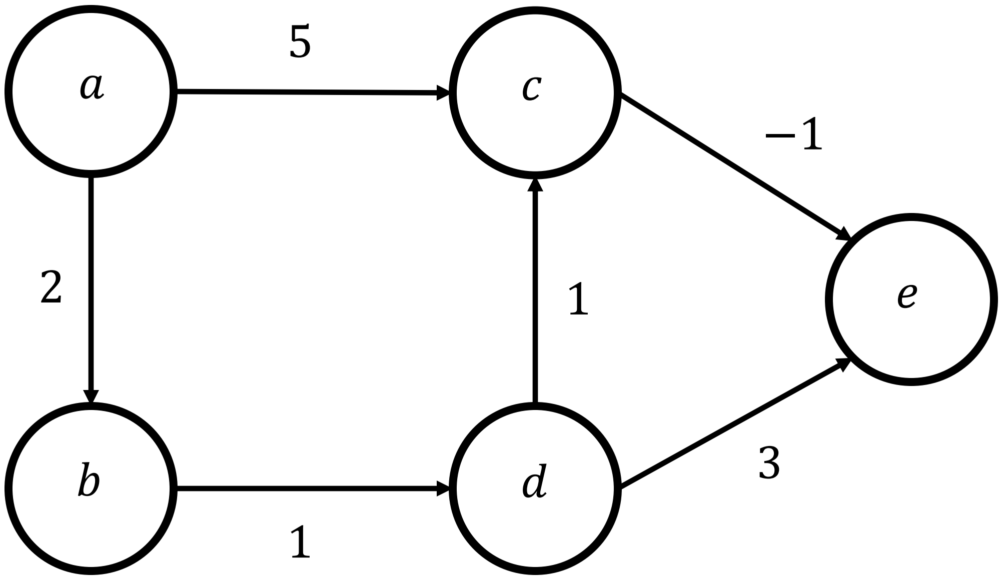
$$
\begin{aligned}
v_{\pi_*}^{t}(s) &= \max_a \left[\mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_{\pi_*}^{t+1}(S_{t+1}) \middle| S_t = s, A_t = a\right]\right] \\
&= \max_a \left[R(s,a) + \sum_{s'} \gamma v_{\pi_*}^{t+1}(s')P(s,a,s')\right] \\
&\geq \max_a \left[R(s,a) + \sum_{s'} \gamma v_{\pi'}^{t+1}(s')P(s,a,s')\right] \\
&\geq \sum_{a'} \pi'(a'|s) \left[R(s,a') + \sum_{s'} \gamma v_{\pi'}^{t+1}(s')P(s,a',s')\right] \\
&= v_{\pi'}^{t}(s)
\end{aligned}
$$

Stochastic policy version

Marginalize over all actions

Dynamic Programming: All Pairs Shortest Paths

- A famous application of dynamic programming
  - Compute shortest paths from a node to all other nodes in a graph
  - E.g., all shortest paths from a to other nodes

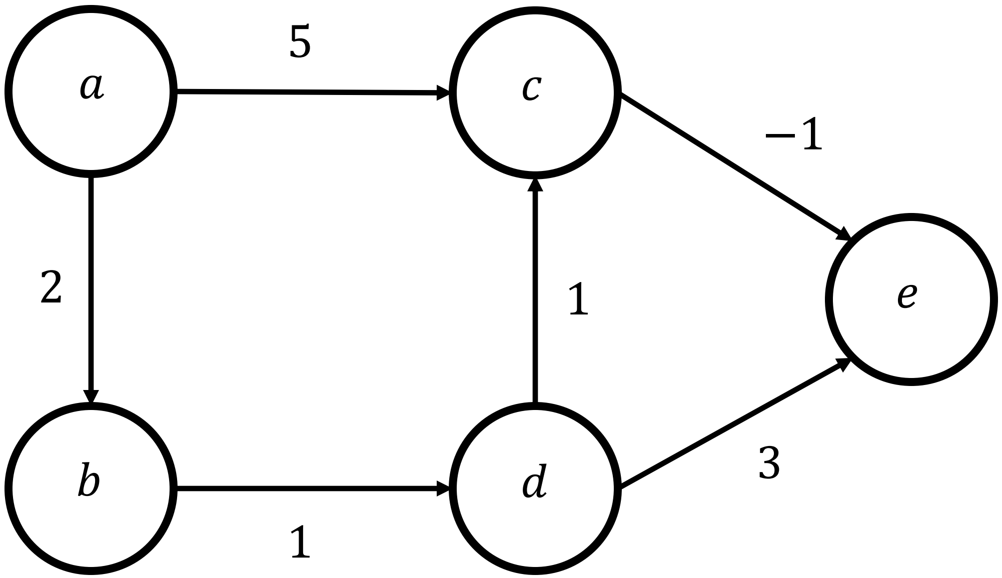


1 step

a	0		
b	2		
c	5		
d	∞		
e	∞		

Dynamic Programming: All Pairs Shortest Paths

- A famous application of dynamic programming
  - Compute shortest paths from a node to all other nodes in a graph
  - E.g., all shortest paths from a to other nodes

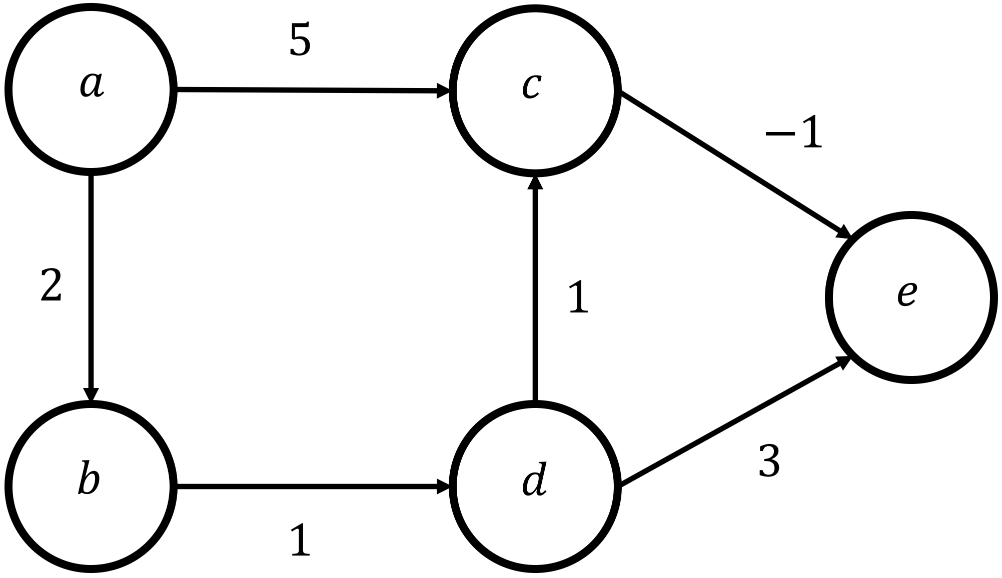


	1 step	2 steps	
a	0	0	
b	2	2	
c	5	5	
d	∞	3	
e	∞	4	

Loop through all 1-step nodes and see if you can reach other nodes in 1 step at lower cost

Dynamic Programming: All Pairs Shortest Paths

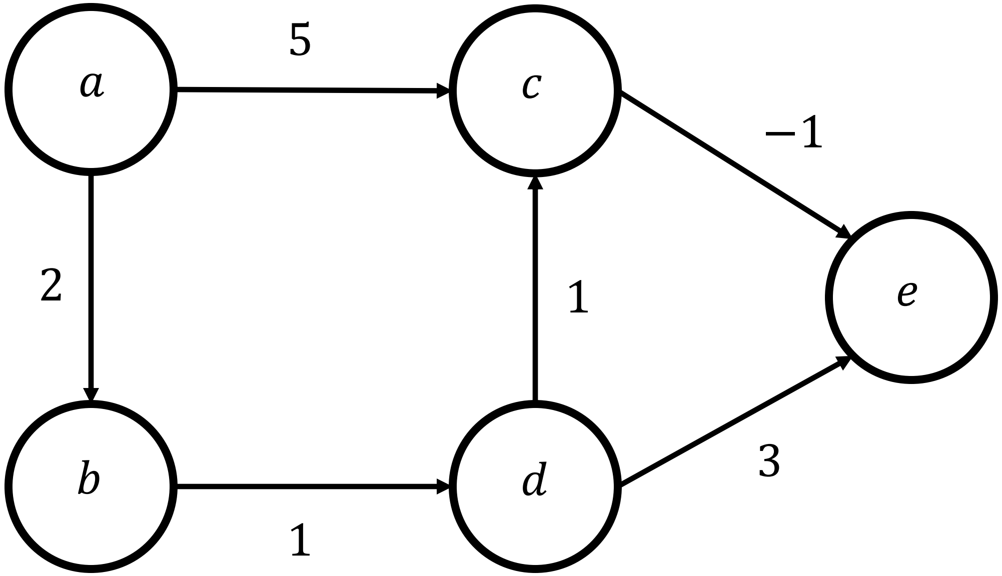
- A famous application of dynamic programming
  - Compute shortest paths from a node to all other nodes in a graph
  - E.g., all shortest paths from a to other nodes



	1 step	2 steps	3 steps
a	0	0	0
b	2	2	2
c	5	5	4
d	∞	3	3
e	∞	4	4

Dynamic Programming: All Pairs Shortest Paths

- A famous application of dynamic programming
  - Compute shortest paths from a node to all other nodes in a graph
  - E.g., all shortest paths from a to other nodes



	1 step	2 steps	3 steps	4 steps
a	0	0	0	0
b	2	2	2	2
c	5	5	4	4
d	∞	3	3	3
e	∞	4	4	3

Cost to e through c is updated to 3

Dynamic Programming: All Pairs Shortest Paths

- Repeat until no more improvements can be made
  - For each node n, go through all edges (n, n')
    - If $cost(a, n') > cost(a, n) + cost(n, n')$
    - Set $cost(a, n') = cost(a, n) + cost(n, n')$
- What is the worst-case complexity of the algorithm?
  - Complexity is $O(n^3)$, where n is the number of nodes
    - Each loop requires $\mathrm{O}(n^2)$ operations
      - For each node, go through all other nodes and see if a shorter path exists
    - A total of n loops
      - Longest path to any node is n steps
- Turns out the same algorithm can be applied to MDPs
  - Find the optimal policy from any node

Workday Example

- Suppose $T = 2$
- What is $v_*^{T-1}(Pub)$?

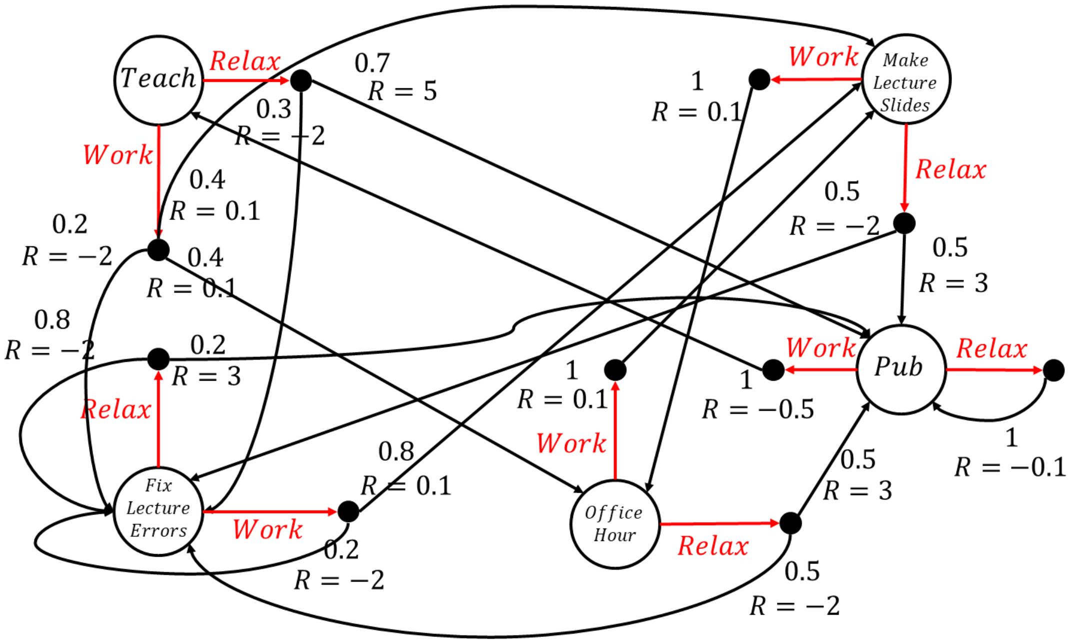
  $q_{\pi_*}^{T-1}(Pub, Work) = -0.5$

  $q_{\pi_*}^{T-1}(Pub, Relax) = -0.1$

  - $Relax$ is better
- What is $v_*^{T-1}(OH)$?

  $q_{\pi_*}^{T-1}(OH, Relax) = 0.5$
- What is $v_*^{T-1}(FLE)$?

  $q_{\pi_*}^{T-1}(FLE, Work) = -0.32$
- Similarly, $q_{\pi_*}^{T-1}(MLS, Relax) = 0.5$
- Similarly, $q_{\pi_*}^{T-1}(Teach, Relax) = 2.9$



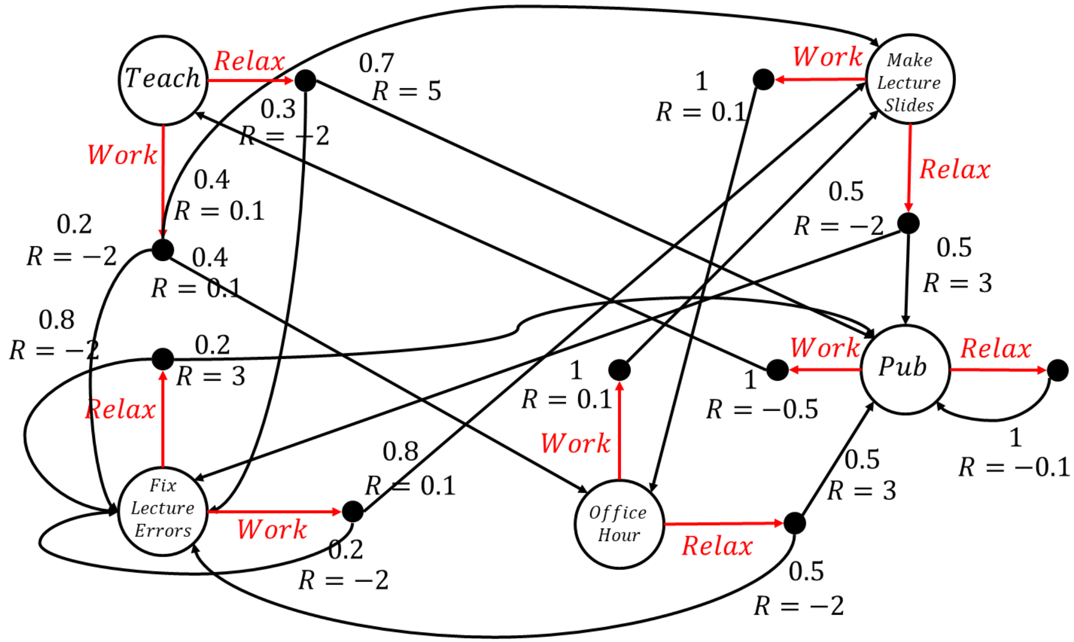
Workday Example

Rensselaer

- Suppose $\gamma = 0.9$
- What is $v_*^0(Pub)$?

  $q_{\pi_*}^0(Pub, Work) =$
  $-0.5 + 0.9 * v_{\pi_*}^1(Teach)$
  $= 2.11$

  $q_{\pi_*}^0(Pub, Relax) =$
  $-0.1 + 0.9 * v_{\pi_*}^1(Pub)$
  $= -0.19$



- What is $v_*^0(FLE)$?

$q_{\pi_*}^0(FLE, Work) = (0.2 * -2 + 0.8 * 0.1) +$
$0.2 * 0.9 * v_{\pi_*}^1(FLE) + 0.8 * 0.9 * v_{\pi_*}^1(MLS) =$
$= -0.0176$

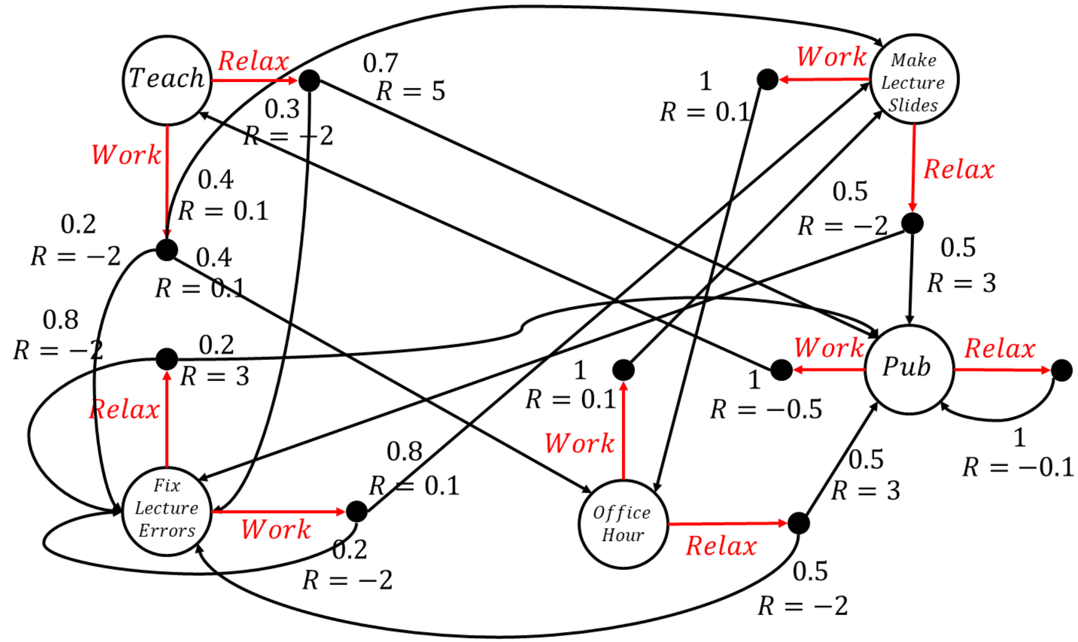
$q_{\pi_*}^0(FLE, Relax) = -1 - 0.9 * 0.276 = -1.25$

	$t = 0$	$t = 1$
$Teach$		2.9
OH		0.5
MLS		0.5
FLE		-0.32
Pub		-0.1

Value table

Workday Example

- Suppose $\gamma = 0.9$
- The other action values computed similarly



	$t = 0$	$t = 1$
$Teach$		2.9
OH		0.5
MLS		0.5
FLE	-0.0176	-0.32
Pub	2.11	-0.1

Value table

Dynamic Programming Summary

Rensselaer

- Iterate backwards, starting from last decision step $T-1$
- First, compute $q_{\pi_*}^{T-1}(s,a)$ for each state/action pair
  - i.e., compute the one-step expected reward
  - Then set $v_{\pi_*}^{T-1}(s) = \max_a q_{\pi_*}^{T-1}(s,a)$
- For $t < T-1$, use Bellman equation to compute $q_{\pi_*}^{t}(s,a)$

$$q_{\pi_*}^{t}(s,a) = \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_{\pi_*}^{t+1}(S_{t+1}) \middle| S_t = s, A_t = a\right]$$

  - Then set $v_{\pi_*}^{t}(s) = \max_a q_{\pi_*}^{t}(s,a)$
- What is the complexity of dynamic programming?

$$O\left(TS^2A\right)$$

  - where S is the number of states, A is the number of actions
  - for each state, loop through all actions and all other states

Dynamic Programming Assumptions

- So far dynamic programming only works for the case of
  - Finite horizon
  - Finite-state space
  - Finite-action space
- Finite-state and –action spaces hard to relax (for now)
- But we can modify algorithm for infinite horizon
- Policy iteration!

Policy Improvement Theorem

- Suppose we have a current policy π, potentially not optimal
  - Suppose we know $v_\pi(s), q_\pi(s,a), \forall s, a$
  - How can we improve the policy for a given s?
  - Pick an action that has a higher q value
- We know $v_\pi(s) = q_\pi(s, \pi(s))$
  - What if there existed an action a' s.t.
    $$q_\pi(s, a') \geq q_\pi(s, \pi(s))$$
  - Turns out the policy that selects a' is better
- **Policy Improvement Theorem**:
  - A policy π' is as good as, or better than, another policy π if for all $s \in S$
    $$q_\pi\big(s, \pi'(s)\big) \geq v_\pi(s)$$

Policy Improvement Theorem Proof

- First recall that for a specific action a, the q value is:

$$q_\pi(s,a) = R_e(s,a) + \gamma \sum_{s'} P(s,a,s')v(s')$$
$$= R_e(s,a) + \gamma \boldsymbol{p}(s,a)^T v(\boldsymbol{s})$$

  – where $\boldsymbol{p}(s,a)^T = [P(s,a,s_1), \dots, P(s,a,s_N)]$

- Wlog, suppose π' is different from π only at s_1, i.e.,

$$q_\pi\big(s_1, \pi'(s_1)\big) \geq v_\pi(s_1)$$

- Using the Bellman equation:

$$q_\pi\big(s_1, \pi'(s_1)\big) = R_e\big(s_1, \pi'(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi'(s_1)\big)^T v_\pi(\boldsymbol{s})$$
$$v_\pi(s_1) = q_\pi\big(s_1, \pi(s_1)\big) = R_e\big(s_1, \pi(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi(s_1)\big)^T v_\pi(\boldsymbol{s})$$

- Then

$$R_e\big(s_1, \pi'(s_1)\big) + \gamma \boldsymbol{p}\big(s_1\ , \pi'(s_1)\big)^T v_\pi(\boldsymbol{s}) \geq R_e\big(s_1, \pi(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi(s_1)\big)^T v_\pi(\boldsymbol{s})$$

Policy Improvement Theorem Proof, cont'd

- We now construct matrix-form Bellman equations for π and π'
- For π

$$v_\pi(\boldsymbol{s}) = \begin{bmatrix} q_\pi\big(s_1, \pi(s_1)\big) \\ \dots \\ q_\pi\big(s_N, \pi(s_N)\big) \end{bmatrix}$$

$$= \begin{bmatrix} R_e\big(s_1, \pi(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi(s_1)\big)^T v_\pi(\boldsymbol{s}) \\ \dots \\ R_e\big(s_N, \pi(s_N)\big) + \gamma \boldsymbol{p}\big(s_N, \pi(s_N)\big)^T v_\pi(\boldsymbol{s}) \end{bmatrix}$$

$$= R_\pi(\boldsymbol{s}) + \gamma \boldsymbol{P}_\pi v_\pi(\boldsymbol{s})$$

Policy Improvement Theorem Proof, cont'd

- We now construct matrix-form Bellman equations for π and π'
- For π'

$$v_{\pi'}(\boldsymbol{s}) = \begin{bmatrix} q_{\pi'}\big(s_1, \pi'(s_1)\big) \\ q_{\pi'}\big(s_2, \pi(s_2)\big) \\ \cdots \\ q_{\pi'}\big(s_N, \pi(s_N)\big) \end{bmatrix}$$

$$= \begin{bmatrix} R_e\big(s_1, \pi'(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi'(s_1)\big)^T v_{\pi'}(\boldsymbol{s}) \\ R_e\big(s_2, \pi(s_2)\big) + \gamma \boldsymbol{p}\big(s_2, \pi(s_2)\big)^T v_{\pi'}(\boldsymbol{s}) \\ \cdots \\ R_e\big(s_N, \pi(s_N)\big) + \gamma \boldsymbol{p}\big(s_N, \pi(s_N)\big)^T v_{\pi'}(\boldsymbol{s}) \end{bmatrix}$$

$$= R_{\pi'}(\boldsymbol{s}) + \gamma \boldsymbol{P}_{\pi'} v_{\pi'}(\boldsymbol{s})$$

  – Note that $\boldsymbol{P}_\pi$ and $\boldsymbol{P}_{\pi'}$ only differ in their first row

Policy Improvement Theorem Proof, cont'd

- Wlog, suppose π' is different from π only at s_1, i.e.,
$$q_\pi\big(s_1, \pi'(s_1)\big) \geq v_\pi(s_1)$$
- Using the Bellman equation:
$$q_\pi\big(s_1, \pi'(s_1)\big) = R_e\big(s_1, \pi'(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi'(s_1)\big)^T v_\pi(\boldsymbol{s})$$
$$v_\pi(s_1) = q_\pi\big(s_1, \pi(s_1)\big) = R_e\big(s_1, \pi(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi(s_1)\big)^T v_\pi(\boldsymbol{s})$$
- Then
$$R_e\big(s_1, \pi'(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi'(s_1)\big)^T v_\pi(\boldsymbol{s}) \geq R_e\big(s_1, \pi(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi(s_1)\big)^T v_\pi(\boldsymbol{s})$$
- Stack remaining values for π in a vector as follows:
$$\begin{bmatrix} R_e\big(s_1, \pi'(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi'(s_1)\big)^T v_\pi(\boldsymbol{s}) \\ R_e\big(s_2, \pi(s_2)\big) + \gamma \boldsymbol{p}\big(s_2, \pi(s_2)\big)^T v_\pi(\boldsymbol{s}) \\ \cdots \\ R_e\big(s_N, \pi(s_N)\big) + \gamma \boldsymbol{p}\big(s_N, \pi(s_N)\big)^T v_\pi(\boldsymbol{s}) \end{bmatrix} \geq \begin{bmatrix} R_e\big(s_1, \pi(s_1)\big) + \gamma \boldsymbol{p}\big(s_1, \pi(s_1)\big)^T v_\pi(\boldsymbol{s}) \\ R_e\big(s_2, \pi(s_2)\big) + \gamma \boldsymbol{p}\big(s_2, \pi(s_2)\big)^T v_\pi(\boldsymbol{s}) \\ \cdots \\ R_e\big(s_N, \pi(s_N)\big) + \gamma \boldsymbol{p}\big(s_N, \pi(s_N)\big)^T v_\pi(\boldsymbol{s}) \end{bmatrix}$$
  - where the inequality is interpreted element-wise

Policy Improvement Theorem Proof, cont'd

- Stack remaining values for π in a vector as follows:

$$\begin{bmatrix} R_e(s_1,\pi'(s_1)) + \gamma \boldsymbol{p}(s_1,\pi'(s_1))^T v_\pi(\boldsymbol{s}) \\ R_e(s_2,\pi(s_2)) + \gamma \boldsymbol{p}(s_2,\pi(s_2))^T v_\pi(\boldsymbol{s}) \\ \dots \\ R_e(s_N,\pi(s_N)) + \gamma \boldsymbol{p}(s_N,\pi(s_N))^T v_\pi(\boldsymbol{s}) \end{bmatrix} \geq \begin{bmatrix} R_e(s_1,\pi(s_1)) + \gamma \boldsymbol{p}(s_1,\pi(s_1))^T v_\pi(\boldsymbol{s}) \\ R_e(s_2,\pi(s_2)) + \gamma \boldsymbol{p}(s_2,\pi(s_2))^T v_\pi(\boldsymbol{s}) \\ \dots \\ R_e(s_N,\pi(s_N)) + \gamma \boldsymbol{p}(s_N,\pi(s_N))^T v_\pi(\boldsymbol{s}) \end{bmatrix}$$

- In matrix form:

$$R_{\pi'}(\boldsymbol{s}) + \gamma \boldsymbol{P}_{\pi'} v_\pi(\boldsymbol{s}) \geq R_\pi(\boldsymbol{s}) + \gamma \boldsymbol{P}_\pi v_\pi(\boldsymbol{s})$$
$$R_{\pi'}(\boldsymbol{s}) + \gamma \boldsymbol{P}_{\pi'} v_\pi(\boldsymbol{s}) \geq v_\pi(\boldsymbol{s})$$
$$R_{\pi'}(\boldsymbol{s}) \geq v_\pi(\boldsymbol{s}) - \gamma \boldsymbol{P}_{\pi'} v_\pi(\boldsymbol{s})$$
$$R_{\pi'}(\boldsymbol{s}) \geq (\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi'}) v_\pi(\boldsymbol{s})$$

- Pre-multiply both sides by $(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi'})^{-1}$
  - Inequalities don't switch sides (don't have time to prove)

$$(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi'})^{-1} R_{\pi'}(\boldsymbol{s}) \geq v_\pi(\boldsymbol{s})$$
$$v_{\pi'}(\boldsymbol{s}) \geq v_\pi(\boldsymbol{s})$$

Deterministic Policies: Greedy Policy Improvement

Rensselaer

- Suppose we are given a deterministic policy π
- We can greedily improve π for each state

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

- By the policy improvement theorem, π' is better than or equal to π
- If $\pi' = \pi$, then $\pi' = \pi^*$:

$$v_{\pi'}(S_t) = \max_a \left[\mathbb{E}_\pi[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a]\right]$$

  - Bellman optimality equation!

Policy Iteration

- The greedy policy improvement approach suggests an algorithm for finding the optimal policy through iterating
  - Start from a policy, compute its value function, improve greedily for one state, repeat…

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

  - Terminate when you find optimal policy
- Guaranteed to terminate because there are finitely many policies in a finite-state MDP
- Policy iteration is trickier in the finite-horizon case
  - Need to evaluate the policy at each time t
  - Just use dynamic programming in the finite-horizon case

Policy Iteration, Workday Example

- What are the optimal actions in the long run?
  - $\pi_*(Teach) = Relax$
  - $\pi_*(OH) = Relax$
  - $\pi_*(MLS) = Relax$
  - $\pi_*(FLE) = Work$
  - $\pi_*(Pub) = Work$
- Corresponding values are

$$v_*(s) = [9.18 \;\; 6.31 \;\; 6.31 \;\; 5.15 \;\; 7.76]$$

Policy Iteration Summary

- Start with a random policy π
- Repeat until you find the optimal policy:
  - Loop through all states
  - For each state s, loop through all actions
  - If you find an action a for which $q_\pi(s,a) > v_\pi(s)$
    - Modify π such that $\pi(s) = a$
    - Recalculate values $v_{\pi'}$ for modified policy π'
    - Go back to main loop
  - If you did not change the policy at all, terminate
    - You found the optimal!

Value Iteration

- Policy iteration requires evaluating each new policy
  - i.e., need to compute $v_\pi(s)$ for all states
  - May take significant time
- Another approach is to use the Bellman optimality equation

$$\begin{aligned} v_{\pi_*}(s) &= \max_a q_{\pi_*}(s,a) \\ &= \max_a \left[\mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_{\pi_*}(S_{t+1}) \middle| S_t = s, A_t = a\right]\right] \end{aligned}$$

- The Bellman optimality equation suggests the recursion

$$v_{k+1}(s) = \max_a \left[\mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]\right]$$

  - Starting from any v_0

Value Iteration, cont'd

- The Bellman optimality equation suggests the recursion
$$v_{k+1}(s) = \max_a \big[\mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]\big]$$
- The sequence is guaranteed to converge
  - Consider the mapping
$$Lv = \max_a \big[\mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a]\big]$$
  - Map L can be shown to be contractive
    - i.e., any 2 sequences get closer to each other after each iteration
  - The sequence v_k converges to a unique v^* for all v_0
- The unique v^* satisfies the Bellman optimality equation
$$v^*(s) = \max_a \big[\mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a]\big]$$
  - So it is the value function corresponding to the optimal policy

Value Iteration Considerations

- Given a value function, the corresponding policy is

$$\pi(s) = argmax_a \; q_\pi(s,a)$$
$$= argmax_a \; \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$
$$= arg\max_a \left[R_e(s,a) + \sum_{s'} \gamma v_\pi(s') P(s,a,s') \right]$$

- Note that a v_k may not have the actual state values of the policy it represents
  - There are finitely many policies but infinitely many v_k
  - Ultimately, we don't care what the state values are as long as the policy is optimal
  - Of course, when the v_k converge, the values will converge to the values of the optimal policy

Value Iteration Summary

- Start from an arbitrary v_0
- For each state s, update v_{k+1} as follows:

$$v_{k+1}(s) = \max_a \left[\mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \right]$$

- Iterate until $|v_{k+1} - v_k| < \epsilon$
  - Where ϵ is a hyperparameter
  - Can use your favorite norm above, e.g., L_∞
- Unlike policy iteration, no need to invert large matrices
  - Though may require many more iterations
  - For large-state-space MDPs, value iteration likely to scale better

Workday Comparison

- The workday example has 5 states and 2 actions
  - How many policies are there in total?

$$2^5 = 32$$

  - Policy iteration likely to converge in several iterations
- Value iteration takes several dozen iterations to converge to true values
  - From an initial state of all 0's
  - Though you don't need to converge fully until you uncover the optimal policy
- Grid world has a bigger policy space

$$4^{25}$$

  - Optimal policy is very simple, so policy iteration still fast

Summary

- Dynamic programming is a powerful iterative algorithm
- Very popular in some fields of computer science and engineering
  - Widely used in control, in a similar way to RL
- Vanilla algorithm only works for finite-space MDPs
  - Overall iteration idea is still mainstream RL, however
- All algorithms discussed so far also need the user to know the MDP structure
  - Not realistic in many cases