



Reinforcement Learning

Dynamic Programming

U Kang
Seoul National University



In This Lecture

- Overview of DP
- Policy Iteration
- Value Iteration
- Generalized Policy Iteration
- Synchronous and Asynchronous DP



Overview

- Dynamic Programming (DP)
 - Collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as MDP
 - Classical DP algorithms are of limited utility in RL both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically
 - DP provides an essential foundation for the understanding of most RL methods, which are attempts to do what DP does, but with less computation and without assuming a perfect model of the environment



Overview

■ Dynamic Programming (DP)

- Key idea: use of value functions to organize and structure the search for good policies
- DP is used to compute optimal value functions
- Given the optimal value function, we can easily obtain optimal policies which satisfy the Bellman optimality equations

- $$v_*(s) = \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')]$$

- $$q_*(s, a) = E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a]$$
$$= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')]$$

- DP turns Bellman equations into update rules



Outline

- ➔ **Policy Evaluation (Prediction)**
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- Generalized Policy Iteration
- Efficiency of DP
- Conclusion



Policy Evaluation (Prediction)

- Policy evaluation (or prediction): given a policy π , compute the state-value function v_π
- $$\begin{aligned} v_\pi(s) &= E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \end{aligned}$$
- $\pi(a|s)$: probability of taking action a in state s under policy π
- v_π exists and is unique as long as either $\gamma < 1$ or eventual termination is guaranteed from all states under policy π



Policy Evaluation (Prediction)

- If the environment's dynamics are completely known, then solving for v_π is equivalent to solving $|S|$ linear equations with $|S|$ unknown
- Iterative policy evaluation
 - Consider a sequence of approximate value functions v_0, v_1, v_2, \dots each mapping state to a real number
 - The initial approximation v_0 chosen arbitrarily
 - Update rule
 - $$v_{k+1}(s) = E_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]$$
 - The sequence $\{v_k(s)\}$ converges to v_π



Policy Evaluation (Prediction)

■ Update rule

- $v_{k+1}(s) = E_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$
 $= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]$
- This is called expected updates

■ Implementation

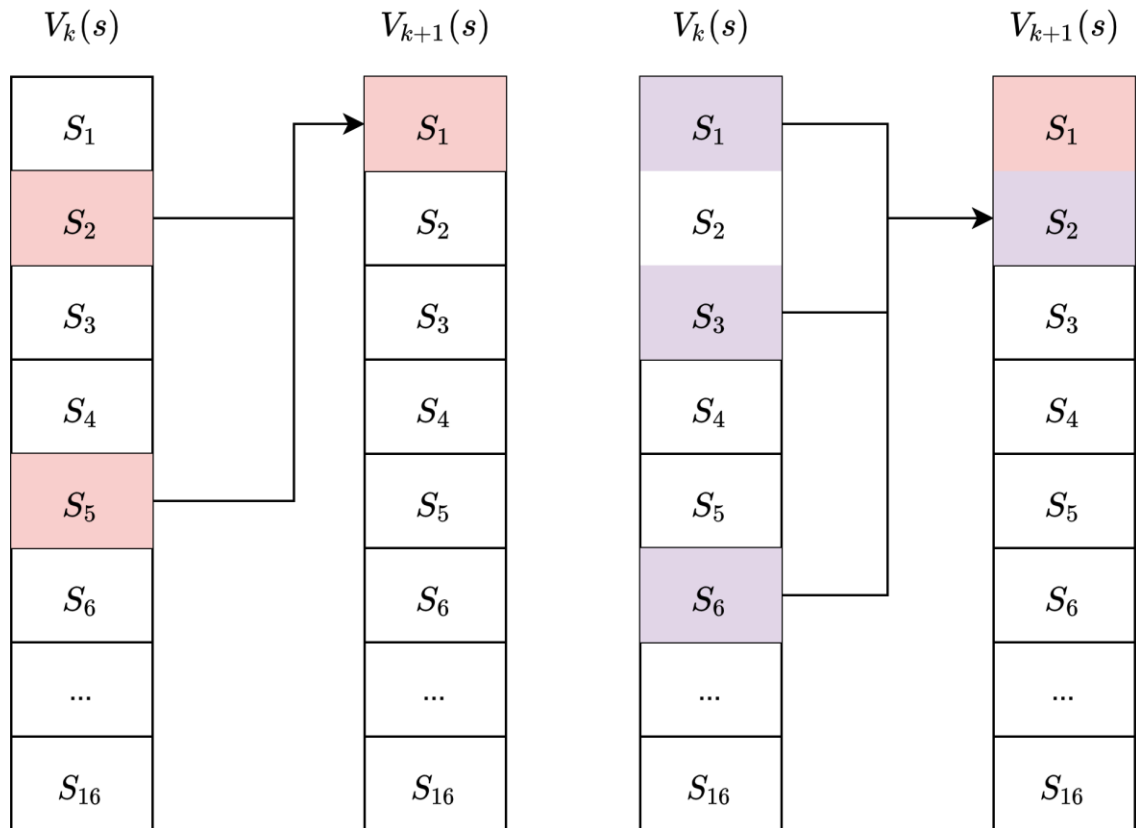
- Two-arrays version
 - One for $v_{k+1}(s)$, the other for $v_k(s)$
- In-place version
 - Use only one array
 - Faster convergence



Policy Evaluation (Prediction)

- Two-arrays version

s_1	s_2	s_3	s_4
s_5	s_6	s_7	s_8
s_9	s_{10}	s_{11}	s_{12}
s_{13}	s_{14}	s_{15}	s_{16}





Policy Evaluation (Prediction)

- In-place version

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π , the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

$\Delta \leftarrow 0$

Loop 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 $\Delta < \theta$



Example: Gridworld

- Nonterminal states $S = \{1, 2, \dots, 14\}$
- Actions $A = \{\text{up, down, right, left}\}$
 - Actions that would take the agent off the grid leave the state unchanged e.g., $p(7, -1 \mid 7, \text{right}) = 1$

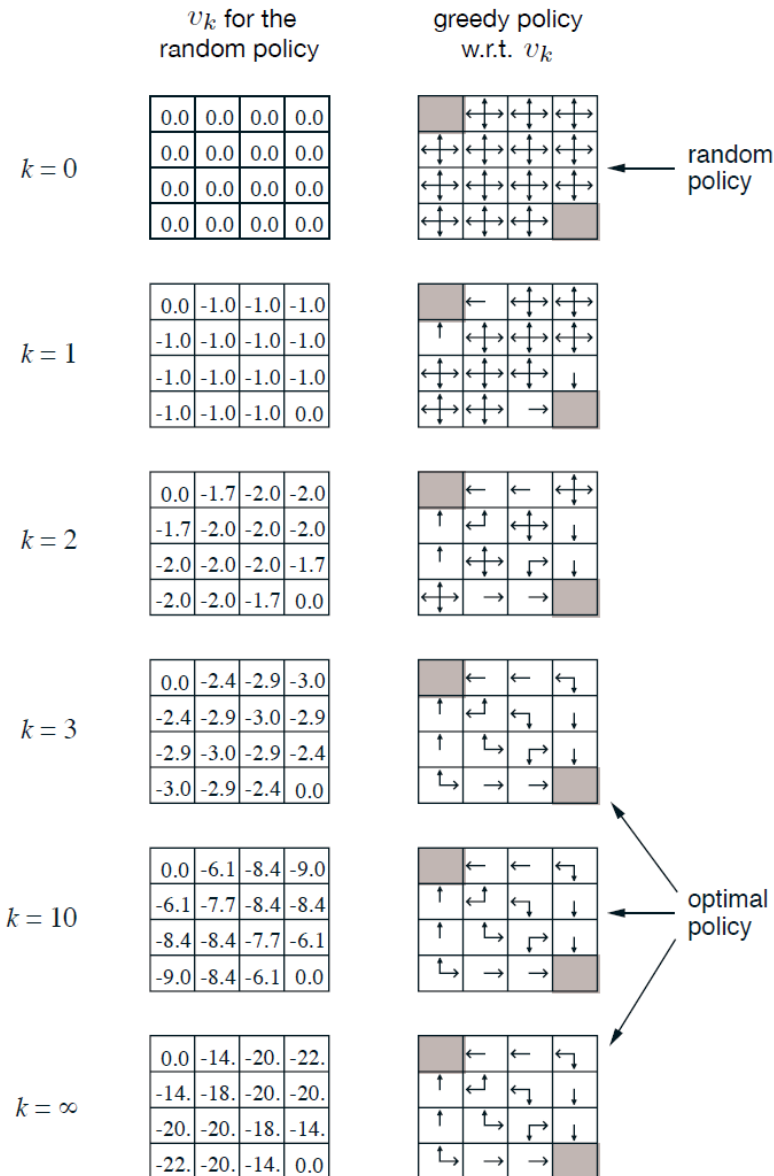


	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions



Example: Gridworld





Outline

Policy Evaluation (Prediction)

 **Policy Improvement**

Policy Iteration

Value Iteration

Asynchronous DP

Generalized Policy Iteration

Efficiency of DP

Conclusion



Policy Improvement

- Suppose we have determined the value function v_π for an arbitrary deterministic policy π
- Would it be better or worse to change to a new policy?
- Idea: evaluate action value function
 - $q_\pi(s, a) = E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$
 $= \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')]$
 - If $q_\pi(s, a) > v_\pi(s)$, then it is better to select a in s and thereafter follow π



Policy Improvement

- Policy improvement theorem
 - Let π and π' be any pair of deterministic policies such that, $q_{\pi}(s, \pi'(s)) \geq v_{\pi}(s)$, for all s
 - Then, π' is as good as, or better than π
 - That is, $v_{\pi'}(s) \geq v_{\pi}(s)$

 - Special case: π' is identical to π except that $\pi'(s) = a \neq \pi(s)$



Policy Improvement

■ Proof of policy improvement theorem

$$\begin{aligned} \square \quad & v_{\pi}(s) \leq q_{\pi}(s, \pi'(s)) \\ &= E[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\ &= E_{\pi'}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s] \\ &\leq E_{\pi'}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\ &= E_{\pi'}[R_{t+1} + \gamma E_{\pi'}[R_{t+2} + \gamma v_{\pi}(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\ &= E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_{\pi}(S_{t+2}) | S_t = s] \\ &\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_{\pi}(S_{t+3}) | S_t = s] \\ &\dots \\ &\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots | S_t = s] \\ &= v_{\pi'}(s) \end{aligned}$$



Policy Improvement

- Greedy policy
 - Consider selecting at each state the action that appears the best according to $q_{\pi}(s, a)$
 - $\pi'(s) = \operatorname{argmax}_a q_{\pi}(s, a)$
 - $= \operatorname{argmax}_a E[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s, A_t = a]$
 - $= \operatorname{argmax}_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')]$
 - By construction, the greedy policy is as good as, or better than, the original policy
 - Policy improvement: the process of making a new policy that improves on an original policy, by making it greedy wrt the value function of the original policy



Policy Improvement

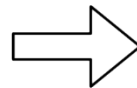
- Greedy policy

Policy evaluation

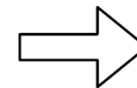
Greedy Policy

→	↓	↓
↓	X	←
→	→	★

$\pi(s)$



0.13 0.13 0.32 0.32	0.12 0.11 0.45 0.32	0.2 0.13 0.2 0.44
0.13 0.13 0.18 0.32	X	0.14 0.22 0.21 0.68
0.17 0.17 0.28 0.16	0.31 0.11 0.9 0.1	★



↖	→	↓
↓	X	↓
→	→	★

$\pi'(s)$



Policy Improvement

■ Stochastic policy

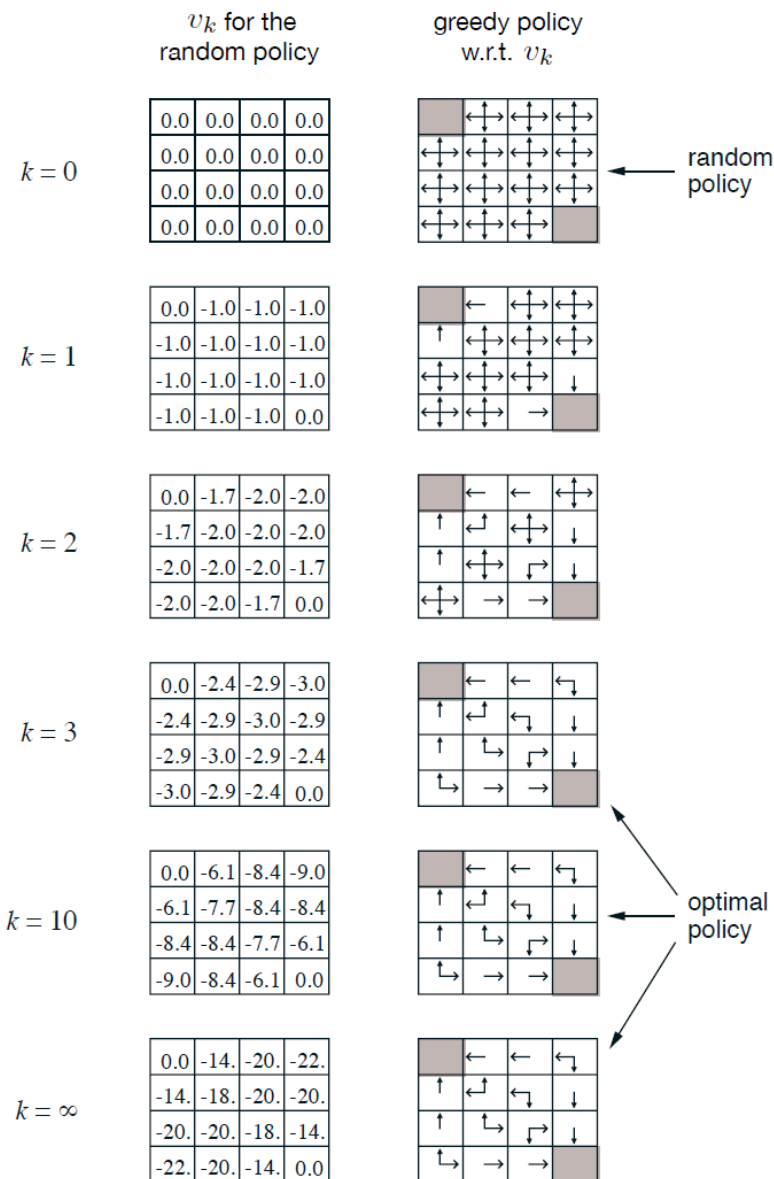
□ (Reminder: greedy policy)

- $\pi'(s) = \operatorname{argmax}_a q_\pi(s, a)$
 $= \operatorname{argmax}_a E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$
 $= \operatorname{argmax}_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')]$

- A stochastic policy π specifies probabilities, $\pi(a|s)$ for taking each action a in each state s
- All the ideas, including policy improvement theorem, we discussed extend easily to stochastic policies
- If there are ties in policy improvement steps, we select any of the maximizing actions




Example: Gridworld





Outline

- Policy Evaluation (Prediction)
- Policy Improvement
-  **Policy Iteration**
- Value Iteration
- Asynchronous DP
- Generalized Policy Iteration
- Efficiency of DP
- Conclusion



Policy Iteration

- After a policy π has been improved using v_π to yield a better policy π' , we can compute $v_{\pi'}$, and improve it again to yield an even better π''

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

- This process converges to an optimal policy and optimal value function in a finite number of iterations
- This is called policy iteration



Policy Iteration

Sutton and Barto,
Reinforcement
Learning, 2018

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

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

2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop 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 determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

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

old-action $\leftarrow \pi(s)$

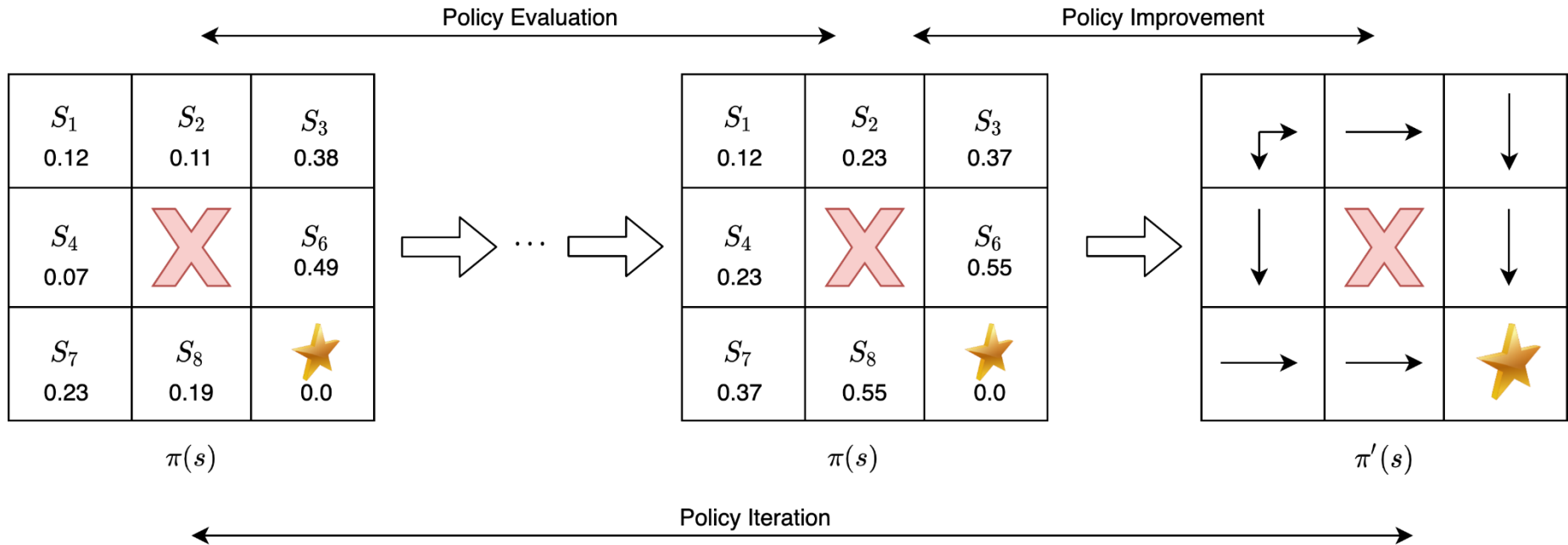
$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

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

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2



Policy Iteration





Example: Jack's Car Rental

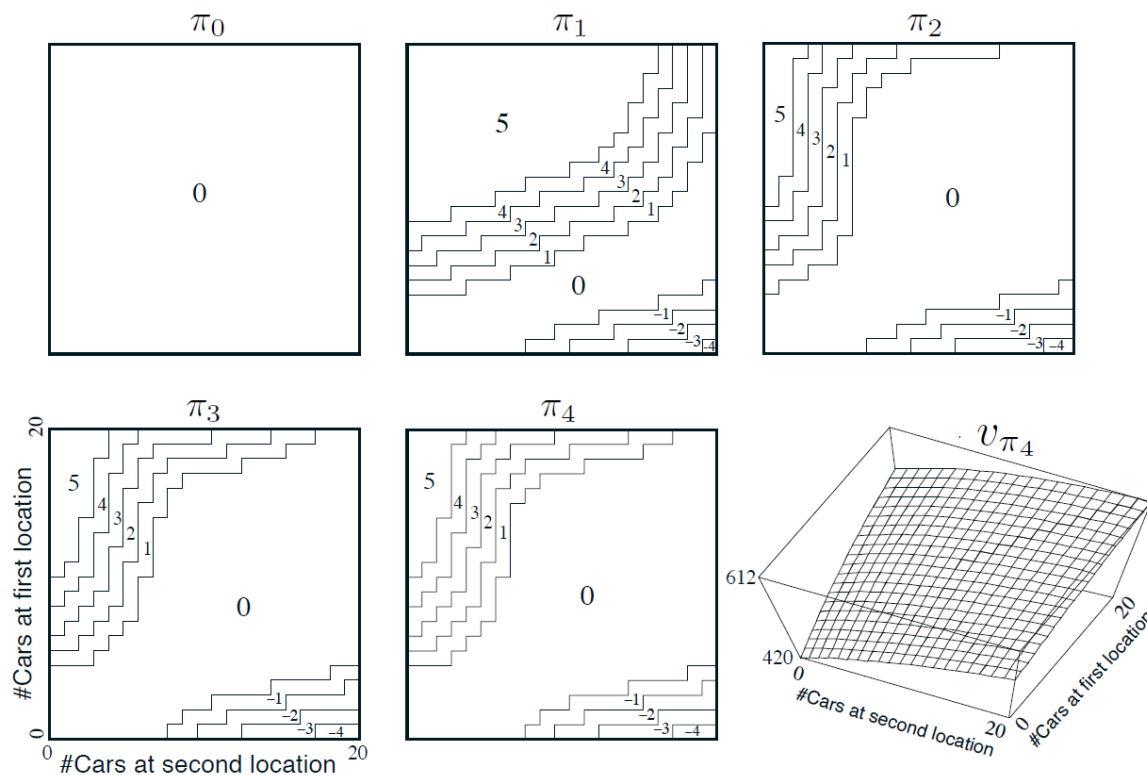
- Jack manages two locations for a car rental company
- For a customer to a location, if Jack has a car available, he rents it out and is credited \$10; no \$ if he is out of cars
- Cars become available for renting the day after they are returned
- Jack can move them between the two locations overnight, at a cost of \$2 per car moved
- Number of cars requested and returned at each location are Poisson random variables: $P(n) = \frac{\lambda^n}{n!} e^{-\lambda}$, where λ is the expected number
- Assume $\lambda_{L1,request} = 3, \lambda_{L2,request} = 4, \lambda_{L1,return} = 3, \lambda_{L2,return} = 2$
- Assume there can be no more than 20 cars at each location, maximum of five cars can be moved from one to the other location, and $\gamma = 0.9$



Example: Jack's Car Rental


■ MDP formulation

- State: the number of cars at each location at the end of the day
- Actions: net number of cars moved from location 1 to location 2 overnight





Outline

- Policy Evaluation (Prediction)
- Policy Improvement
- Policy Iteration
-  **Value Iteration**
- Asynchronous DP
- Generalized Policy Iteration
- Efficiency of DP
- Conclusion



Value Iteration

- Drawback of policy iteration: each iteration involves policy evaluation which is expensive
- Policy evaluation step can be truncated, without losing the convergence guarantees of policy iteration



Reminder: Policy Iteration

Sutton and Barto,
Reinforcement
Learning, 2018

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

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

2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop 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 determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow *true*

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

old-action $\leftarrow \pi(s)$

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

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

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2



Value Iteration

- Value iteration
 - Policy evaluation is stopped after one update of each state
 - $$v_{k+1}(s) = \max_a E[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]$$
 - For arbitrary v_0 , the sequence $\{v_k\}$ converges to v_*



Value Iteration

- Understanding value iteration
 - Value iteration is obtained by turning the Bellman optimality equation into an update rule
 - $$v_*(s) = \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')]$$
 - Value iteration is identical to the policy evaluation update, except that it requires the maximum over all actions
 - $$v_{k+1}(s) = E_\pi [R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$$
$$= \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]$$



Value Iteration

- Termination of value iteration
 - Value iteration formally requires an infinite number of iterations to converge exactly to v_*
 - In practice, we stop when the value function changes by only a small amount

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

```
|  $\Delta \leftarrow 0$   
| Loop 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 $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

Sutton and Barto,
Reinforcement
Learning, 2018

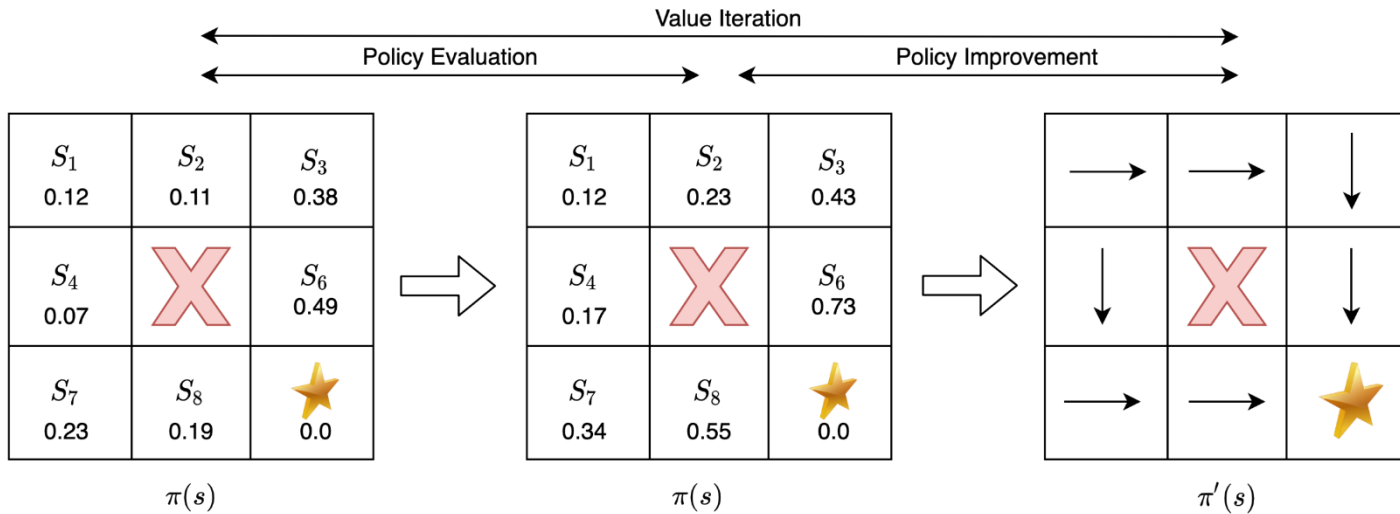
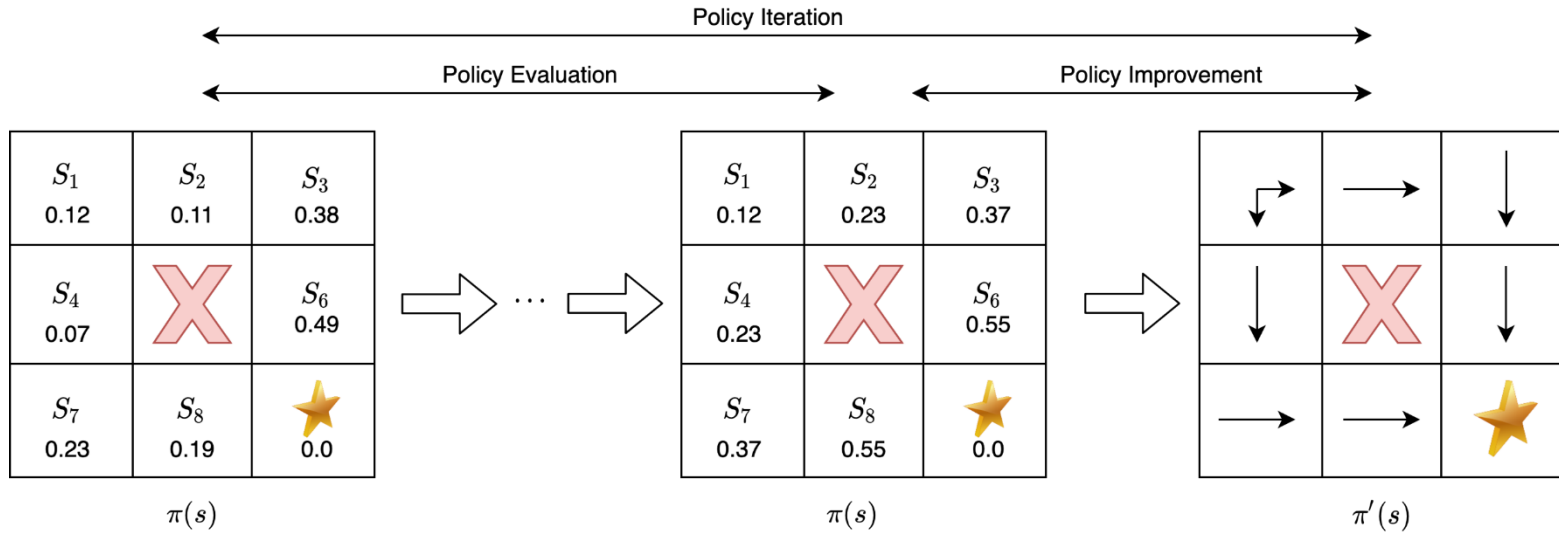


Value Iteration

- Value iteration effectively combines one sweep of policy evaluation and one sweep of policy improvement
- Faster convergence is often achieved by interposing multiple policy evaluation sweeps between each policy improvement sweep
- The entire class of truncated policy iteration algorithms can be thought of as sequences of sweeps (policy evaluation updates and value iteration updates)
- In value iteration, the max operation is added to some sweeps of policy evaluation
- All of these algorithms converge to an optimal policy for discounted finite MDPs



Value Iteration





Example: Gambler's Problem

- A gambler make bets on the outcomes of a sequence of coin flips
- If the flip result = head, he wins as many dollars as he has staked on the flip; otherwise he loses his stake
- The game ends when the gambler wins by earning \$100, or loses by running out of money
- On each flip, the gambler must decide \$\$ to bet

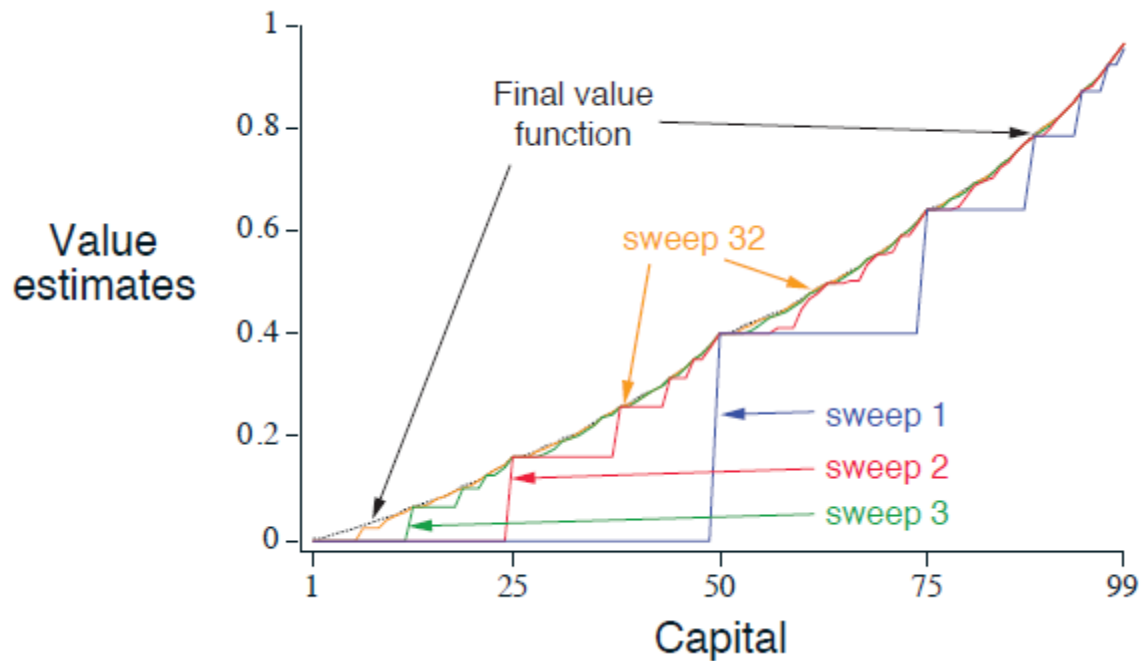


Example: Gambler's Problem

- This problem can be formulated as an undiscounted, episodic, finite MDP
- State s : the gambler's capital (1, 2, ..., 99)
- Action: stake to bet; $a \in \{0, 1, \dots, \min(s, 100 - s)\}$
- Reward: 1 when the gambler reaches the goal of \$100, or 0 on all other cases
- State-value function: gives the probability of winning from each state
- Policy: map capital to stakes
- Optimal policy: maximizes the probability of reaching the goal




Example: Gambler's Problem





Outline

- Policy Evaluation (Prediction)
- Policy Improvement
- Policy Iteration
- Value Iteration
-  **Asynchronous DP**
- Generalized Policy Iteration
- Efficiency of DP
- Conclusion



Asynchronous DP

- Synchronous DP methods require sweeps of the entire state set; if the state set is very large, then it would take too long time for the sweeps
 - Backgammon has over 10^{20} states
- Asynchronous DP: in-place iterative DP
 - Update the values of states in any order, using whatever values of other states happen to be available
 - The values of some states may be updated several times before the values of others are updated once
 - To converge correctly, asynchronous DP must continue to update the values of all the states



Asynchronous DP

- An asynchronous value iteration may update the value of only one state on each step
- If $0 \leq \gamma < 1$, asymptotic convergence to v_* is guaranteed when each state is visited an infinite number of times
- It is possible to intermix policy evaluation and value iteration updates to produce a kind of asynchronous truncated policy iteration



Asynchronous DP

- We can take advantage of this flexibility by selecting the states to which we apply updates so as to improve the algorithm's rate of progress
- We can order the updates to let value information propagate from state to state in an efficient way
- Some states may not need their values updated as often as others
- We might even try to skip updating some states entirely if they are not relevant to optimal behavior



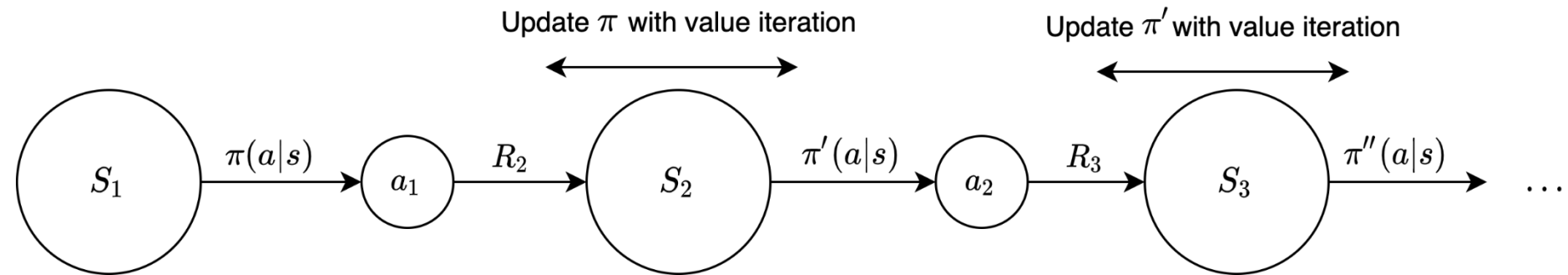
Asynchronous DP

- Intermixing computation with real-time interaction
 - To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP
 - The agent's experience can be used to determine the states to which the DP algorithm applies its updates
 - At the same time the latest value and policy information from the DP algorithm can guide the agent's decision making
 - For example, we can apply updates to states as the agent visits them
 - This makes it possible to focus the DP algorithm's updates onto parts of the state set that are most relevant to the agent




Asynchronous DP

- Real-time Dynamic Programming (RTDP)





Outline

- Policy Evaluation (Prediction)
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
-  **Generalized Policy Iteration**
- Efficiency of DP
- Conclusion



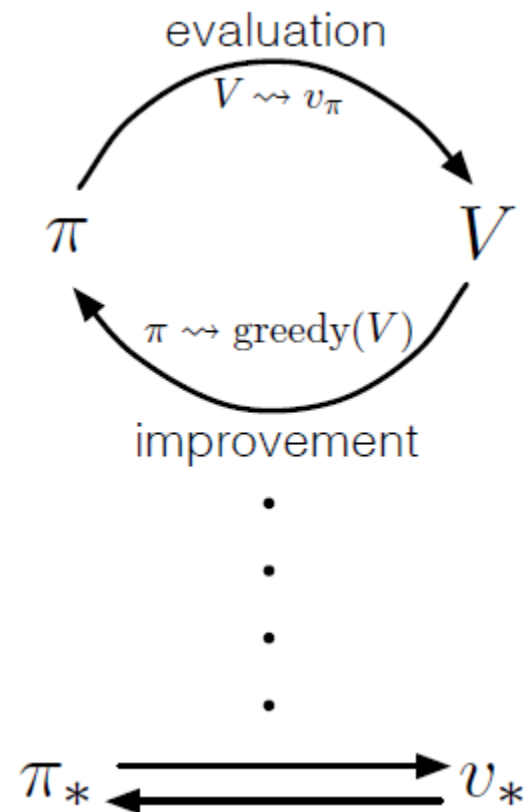
Generalized Policy Iteration

- Policy iteration consists of two simultaneous, interacting processes
 - Policy evaluation: make the value function consistent with the current policy
 - Policy improvement: make the policy greedy with respect to the current value function
- The two processes alternate, but not necessarily
 - Value iteration: only a single iteration of policy evaluation is performed in between each policy improvement
 - Asynchronous DP: the two processes are interleaved at a finer grain
 - In some cases, a single state is updated in one process before returning to the other
 - As long as both processes continue to update all states, the ultimate result typically converges to the optimal value function and policy



Generalized Policy Iteration

- Generalized Policy Iteration (GPI)
 - General idea of letting policy-evaluation and policy-improvement process interact, independent of the granularity and other details of the two processes
 - Almost all RL methods are well described as GPI; they have policies and value functions, with the policy always being improved with respect to the value function and the value function always being driven toward the value function for the policy



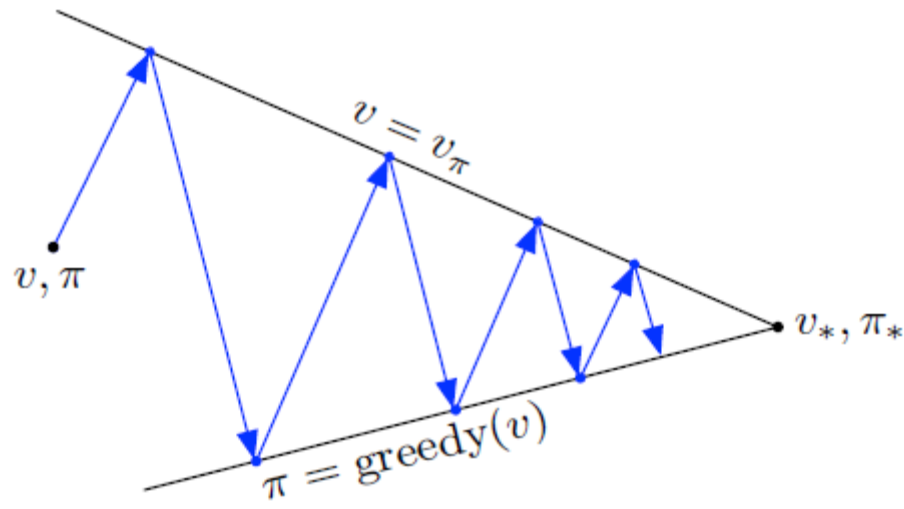


Generalized Policy Iteration

- Generalized Policy Iteration (GPI)
 - If both the evaluation process and the improvement process stabilize, then the value function and policy is optimal
 - The value function stabilizes only when it is consistent with the current policy, and the policy stabilizes only when it is greedy with respect to the current value function
 - This implies that the Bellman optimality equation holds, and thus that the policy and the value function are optimal




Generalized Policy Iteration





Outline

- Policy Evaluation (Prediction)
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous DP
- Generalized Policy Iteration
-  **Efficiency of DP**
- Conclusion



Efficiency of DP

- DP may not be practical for very large problems, but compared with other methods for solving MDPs, DP methods are actually quite efficient
- The (worst case) time DP methods take to find an optimal policy is polynomial in the numbers n of states and k of actions
 - Even though the total number of (deterministic) policies is k^n
- Thus, DP is exponentially faster than any direct search in policy space could be



Efficiency of DP

- LP (linear programming) methods can also be used to solve MDPs
- But LP methods become impractical at a much smaller number of states than do DP methods (by a factor of about 100)
- For the largest problems, only DP methods are feasible



Efficiency of DP

- DP methods can be used with today's computers to solve MDPs with millions of states
- Both policy iteration and value iteration are widely used
- These methods usually converge much faster than their theoretical worst-case run times, particularly if they are started with good initial value functions or policies.




Efficiency of DP

- On problems with large state spaces, asynchronous DP methods are often preferred, since they do not need to visit all states for each iteration
- Synchronous DP methods are too expensive for problems with large states



Outline

- Agent-Environment Interface
- Goals and Rewards
- Returns and Episodes
- Episodic and Continuing Tasks
- Policies and Value Functions
- Optimal Policies and Value Functions
- Optimality and Approximation
-  **Conclusion**



Conclusion

- Ideas and algorithms of DP to solve finite MDPs
- Policy evaluation: iterative computation of the value functions for a given policy
- Policy improvement: computation of an improved policy given the value function for that policy
- These computations combine to get policy iteration and value iteration, the two most popular DP methods
- DP methods operate in sweeps through the state set, performing an expected update operation on each state
- Each such operation updates the value of one state based on the values of all possible successor states and their probabilities of occurring.
- Expected updates are based on Bellman equations



Conclusion

- Generalized Policy Iteration (GPI)
 - The general idea of two interacting processes revolving around an approximate policy and an approximate value function
 - One process takes the policy as given and performs some form of policy evaluation, changing the value function to be more like the true value function for the policy
 - The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value function is its value function
 - A policy and value function that are unchanged by either process are optimal



Conclusion

- Asynchronous DP
 - In-place iterative methods that update states in an arbitrary order
 - Converges faster to the optimal solution
- Bootstrapping
 - DP methods update estimates of the values of states based on estimates of the values of successor states
 - This idea is called 'bootstrapping'
 - DP is a bootstrapping method that also requires a complete and accurate model of the environment
 - There are methods that do not require a model and do not bootstrap
 - There are methods that do not require a model but do bootstrap



Questions?