

Planning by Dynamic Programming

Prof. Junni Zou

Institute of Media, Information and Network
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University
<http://min.sjtu.edu.cn>

Spring, 2024

Outline

- 1 Policy Evaluation
- 2 Policy Iteration
- 3 Value Iteration

Table of Contents

① Policy Evaluation

② Policy Iteration

③ Value Iteration

What is Dynamic Programming?

Dynamic sequential or temporal component to the problem
Programming optimizing a "program", i.e. a policy

- c.f. linear programming

- A method for solving complex problem
- By breaking them down into subproblems
 - Solve the subproblems
 - Combine solutions to subproblems

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
 - Principle of optimality applies
 - Optimal solution can be decomposed into subproblems
- Overlapping subproblems
 - Subproblems recur many times
 - Solutions can be cached and reused
- Markov decision process satisfy both properties
 - Bellman equation gives recursive decomposition
 - Value function stores and reuses solutions

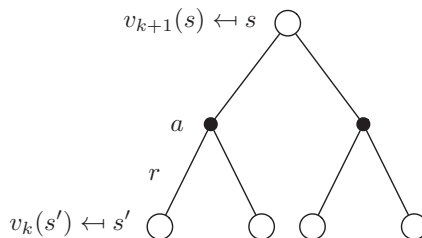
Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP
- For prediction (policy evaluation):
 - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
 - or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
 - Output: value function v_π
- Or for control:
 - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
 - Output: optimal value function v_* and optimal policy π_*

Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$
- Using *synchronous* backups,
 - At each iteration $k + 1$
 - For all states $s \in \mathcal{S}$
 - Update $v_{k+1}(s)$ from $v_k(s')$
 - where s' is a successor state of s
- We will discuss *asynchronous* backups later
- Convergence to v_π will be proven at the end of the lecture

Iterative Policy Evaluation (2)



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$$

Iterative Policy Evaluation (3)

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 $s \in \mathcal{S}$:

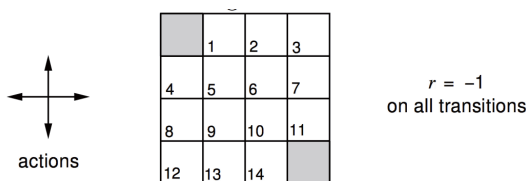
$v \leftarrow V(s)$

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

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

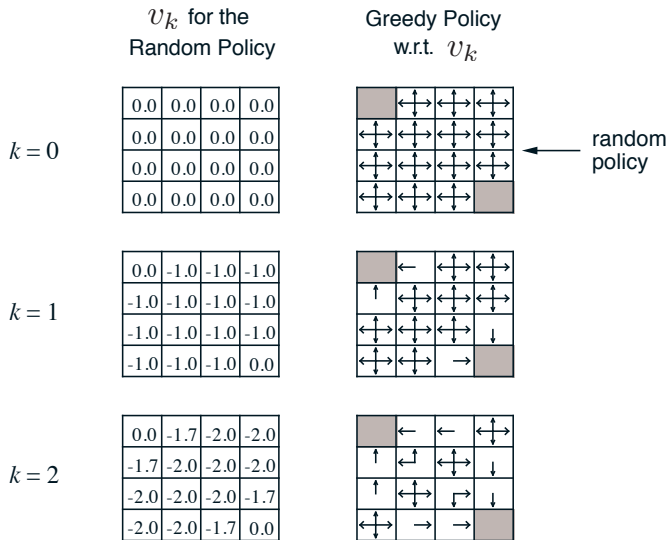
until $\Delta < \theta$

Example: Evaluating a Random Policy in the Small Gridworld



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \dots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy
 $\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$

Example: Iterative Policy Evaluation in Small Gridworld



Example: Iterative Policy Evaluation in Small Gridworld (2)

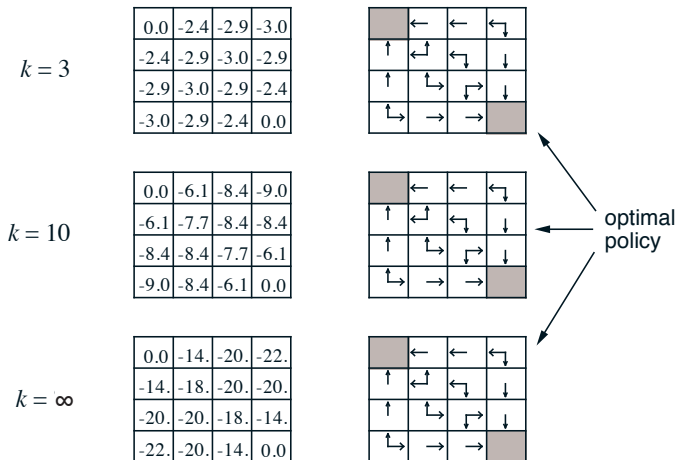


Table of Contents

- ① Policy Evaluation
- ② Policy Iteration
- ③ Value Iteration

Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \arg \max_{a \in \mathcal{A}} q_{\pi}(s, a)$$

- This improves the value from any state s over one step,

$$v_{\pi'} = q_{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_{\pi}(s, a) \geq q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- It therefore improves the value function, $v_{\pi'} \geq v_{\pi}(s)$

$$\begin{aligned} v_{\pi}(s) &\leq q_{\pi}(s, \pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_{\pi}(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s] = v_{\pi'}(s) \end{aligned}$$

Policy Improvement (2)

- If improvements stop,

$$q_{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_{\pi}(s, a) = q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- Then the Bellman optimality equation has been satisfied

$$v_{\pi}(s) = \max_{a \in \mathcal{A}} q_{\pi}(s, a)$$

- Therefore $v_{\pi}(s) = v_*(s)$, $\forall s \in \mathcal{S}$
- So π is an optimal policy

How to Improve a Policy

- Given a policy π
 - Evaluation** the policy π

$$v_{\pi}(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

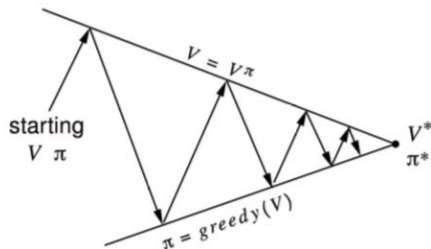
- Improvement** the policy by acting greedily with respect to v_{π}

$$\pi' = \text{greedy}(v_{\pi})$$

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of **policy iteration** always converges to π^*

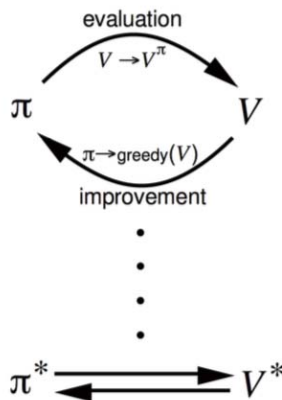
Policy Iteration

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



Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement



Policy Iteration (2)

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 $\leftarrow \pi(s)$

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

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

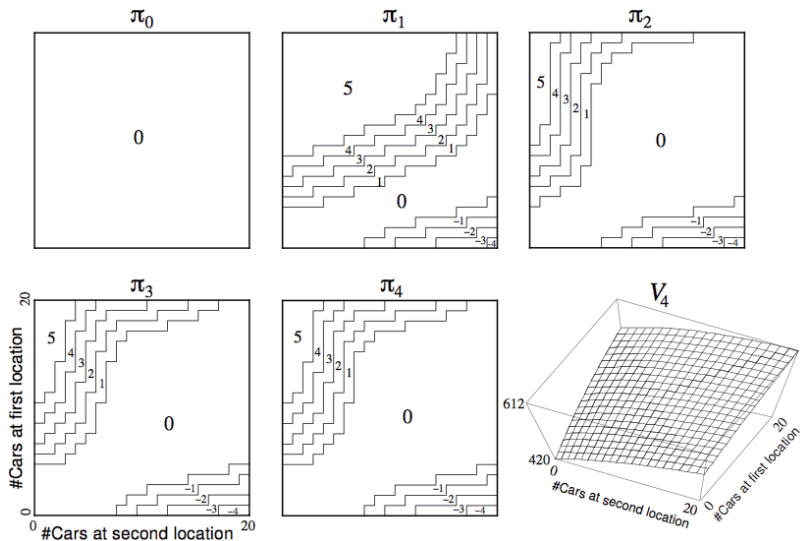
If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

e of Media,
ation, and Network

Jack's Car Rental

- States: Car numbers in two locations, maximum of 20 cars each
- Actions: Move up to 5 car between locations overnight
- Reward: \$10 for each car rented (must be available)
- Transitions: Cars returned and requested randomly
 - Poisson distribution, n returns/ requests with prob $\frac{\lambda^n}{n!} e^{-\lambda}$
 - 1st location: average requests = 3, average returns = 3
 - 2nd location: average requests = 4, average returns = 2
- Discount rate: $\gamma = 0.9$
- **Question:** What is π^* and V^* ?

Policy Iteration in Jack's Car Rental



1edra,
and Network

Modified Policy Iteration

- Does policy evaluation need to converge to v_π ?
- Or should we introduce a stopping condition
 - e.g. ϵ -convergence of value function
- Or simple stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration? i.e. stop after $k = 1$
 - This is equivalent to value *iteration* (next section)

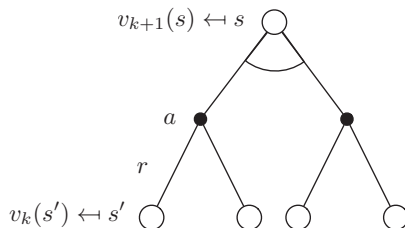
Table of Contents

- 1 Policy Evaluation
- 2 Policy Iteration
- 3 Value Iteration**

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Using synchronous backups
 - At each iteration $k + 1$
 - For all states $s \in \mathcal{S}$
 - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_* will be proven later
- Unlike policy iteration, there is no explicit policy
- intermediate value functions may not correspond to any policy

Value Iteration (2)



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a \mathbf{v}_k$$

Value Iteration (3)

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

$$\pi(s) = \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

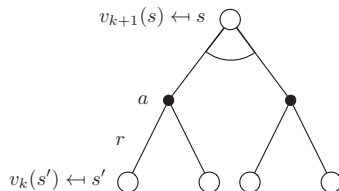
Synchronous Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_{\pi}(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_{\pi}(s, a)$ or $q_*(s, a)$
- Complexity $O(m^2n^2)$ per iteration

Full-Width Backups

- DP uses *full-width* backups
- For each backup (sync or async)
 - Every successor state and action is considered
 - Using knowledge of the MDP transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers Bellman's *curse of dimensionality*
 - Number of state/action spaces grows exponentially
- Even one backup can be too expensive



Sample Backups

- Using sample rewards and sample transitions $\langle S, A, R, S' \rangle$
- Instead of reward function \mathcal{R} and transition dynamic \mathcal{P}
- Advantages:
 - Model-free: no advance knowledge of MDP required
 - Breaks the curse of dimensionality through sampling
 - Cost of backup is constant, independent of $n = |\mathcal{S}|$



Approximate Dynamic Programming

- Approximate the value function
- Using a *function approximator* $\hat{v}(s, \mathbf{w})$
- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration k
 - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
 - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}(s', \mathbf{w}_k) \right)$$

- Train next value function $\hat{v}(\cdot, \mathbf{w}_{k+1})$ using targets $\langle s, \tilde{v}_k(s) \rangle$