Dynamic Programming

Reinforcement Learning

Chapter 4

Dev3B 권요한

Contents

❖ Warm Up

- Before Beginning
- Review
- Introduction

❖ Part 1

- 4.1 Policy Evaluation
- 4.2 Policy Improvement
- 4.3 Policy Iteration
- 4.4 Value Iteration

❖ Part 2

- 4.5 Asynchronous Dynamic Programming
- 4.6 Policy Improvement
- 4.7 Efficiency of Dynamic Programming
- 4.8 Summary

Warm Up

Before Beginning

❖ Mathematical Facts Check

- Expectation

$$\mu_X = E[X] = \begin{cases} \sum_k x_k P_X(x_k) & X \text{ discrete} \\ \int_{-\infty}^{\infty} x f_X(x) dx & X \text{ continuous} \end{cases}$$

X : random variable
x : available value
$P_X(X = x_k)$: probability mass function

- Maximum

$$\max_a f(s,a) \qquad \arg\max_a f(s,a)$$

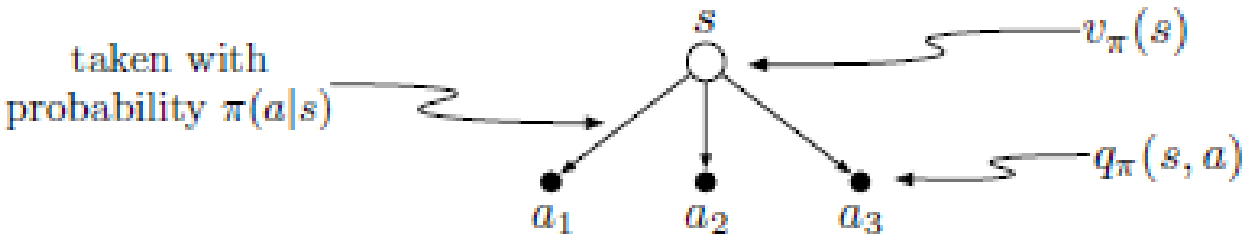
- Poisson distribution

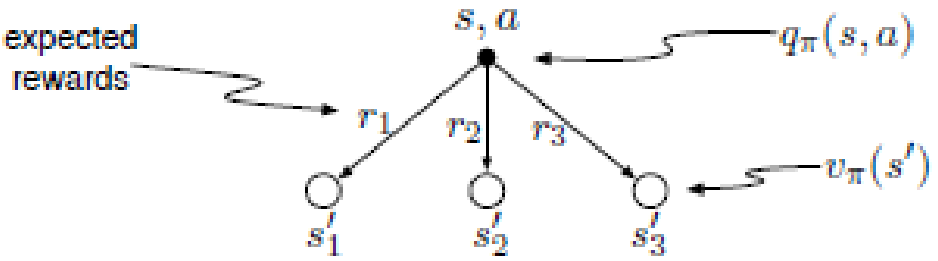
$$P_K(k) = \begin{cases} \frac{\alpha^k}{k!} e^{-\alpha} & x = 0, 1, \cdots \\ 0 & \text{otherwise} \end{cases}$$

K : number of events occur in T
k : number of events
α : event rate

Review

❖ Markov Decision Process





- G_t : *return*; sum of the rewards
- $v_\pi(s)$: *state-value function*; expected return of a state *s* under a policy π
- $q_\pi(s,a)$: *action-value function*; given an action *a*
- π_* : *optimal policies*; better (expected return) than or equal to all policies

- Bellman equation

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

$$v_*(s) \doteq \max_\pi v_\pi(s)$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$

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

Introduction

❖ Dynamic programming(DP)

: a collection of algorithms that can be used to compute optimal policies given a model of environment as a *Markov decision process(MDP).*

- Limitation : assumption of a perfect model, great computational expense

-> but still important theoretically!

- Key idea : use of value functions to organize and structure the search for good policies

- In this chapter, we show

how DP can be used to compute the value functions defined in Chapter 3.

Part 1

4.1 Policy Evaluation

❖ Iterative policy evaluation

- A sequence of approximate value functions : $v_0,\ v_1,\ v_2,\ \ldots$
- The initial approximation, v_0, is chosen arbitrarily except the terminal state.
- And, each successive approximation is obtained by using the *Bellman equation* for v_π as an update rule:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned} \tag{4.5}$$

for all $s \in S$. $\lim_{k \to \infty} v_k(s) = v_\pi(s)$

- *Expected update* : based on an expectation over all possible next states rather than on a sample next state.

4.2 Policy Improvement

❖ Policy improvement theorem

- $q_\pi(s, \pi'(s)) \geq v_\pi(s).$ (4.7)
- $v_{\pi'}(s) \geq v_\pi(s).$ (4.8)
- $$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] && \text{(by (4.6))} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] && \text{(by (4.7))} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s).
\end{aligned}$$

- $\pi'(s) \doteq \arg\max_a q_\pi(s, a)$ → optimal policy

4.2 Policy Improvement

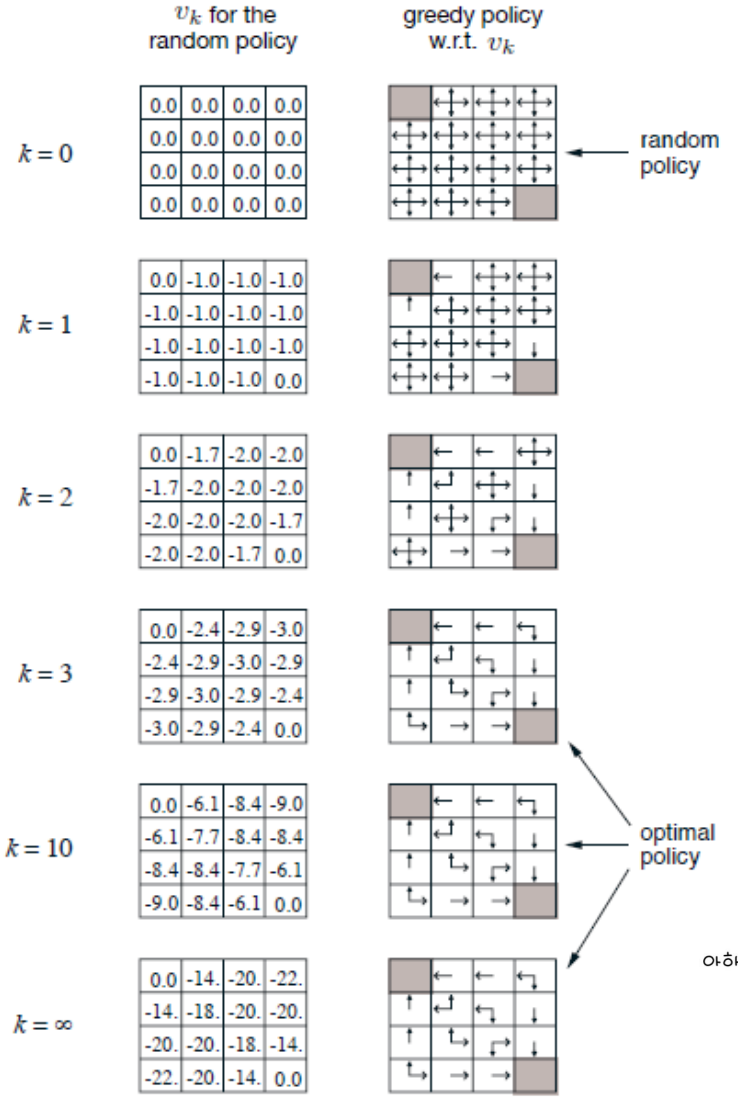
❖ Example 4.1:

actions

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

$R_t = -1$
on all transitions

- $S = \{1,2,\ldots,14\}, A = \{up, down, right, left\}$



4.3 Policy Iteration

❖ Policy iteration

- a way of finding an optimal policy

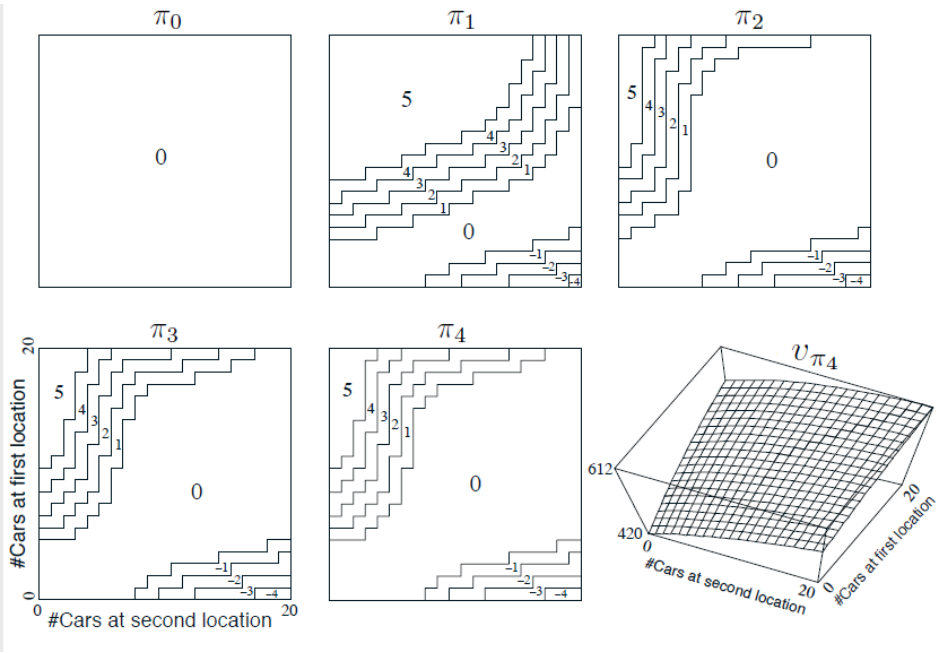
$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

❖ Example 4.2: Jack's Car Rental

- Time step : days
- State : number of cars at each location
- Actions : numbers of cars moved overnight
- Rewards : 10$ for rent, -2$ for each car move
- Number of cars requested and returned

-> Poisson random variables

$\gamma = 0.9$



4.4 Value Iteration

❖ Value iteration

- Just one sweep for policy evaluation
- Combines the policy improvement and truncated policy evaluation.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_k(s') \Big], \end{aligned} \tag{4.10}$$

- Compare with

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_*(s') \Big], \text{ or} \end{aligned} \tag{4.1}$$

4.4 Value Iteration

❖ Compare

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 $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

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 $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

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

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 $s \in \mathcal{S}$:
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
     $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
     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

4.4 Value Iteration

❖ Example 4.3: Gambler's problem

- State : gambler's capital,

  $s \in \{1,2,\ldots,99\}$

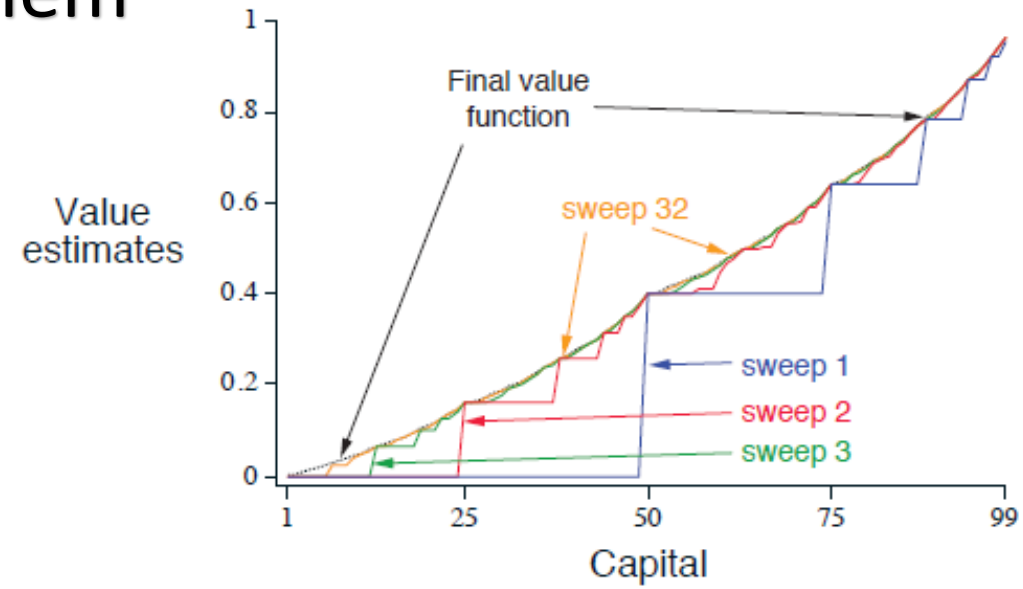
- Actions : stakes,

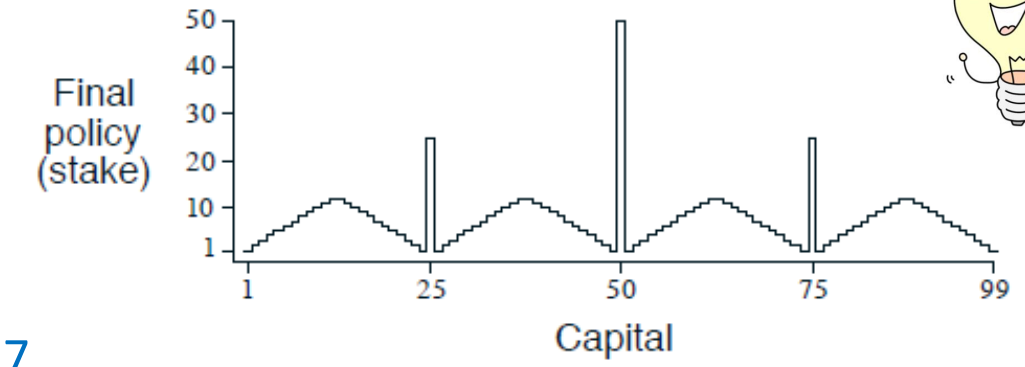
  $a \in \{0,1,\ldots,\min(s, 100-s)\}$

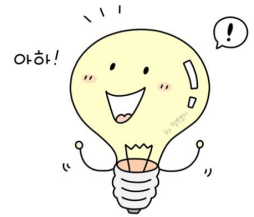
- Rewards : 0 except he reaches

  his goal when it is +1

- p_h = 0.4

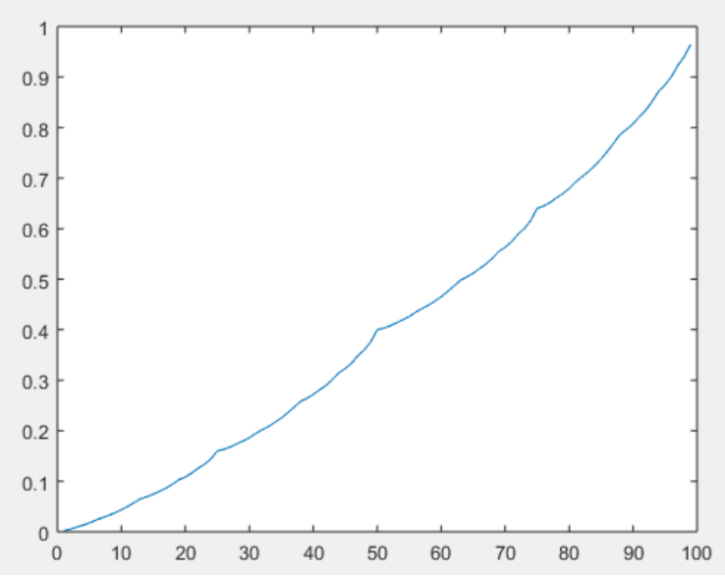


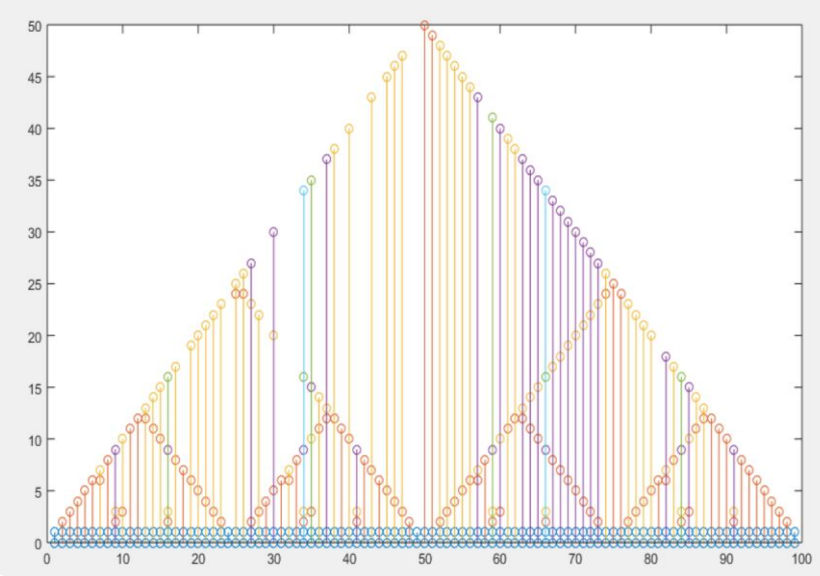


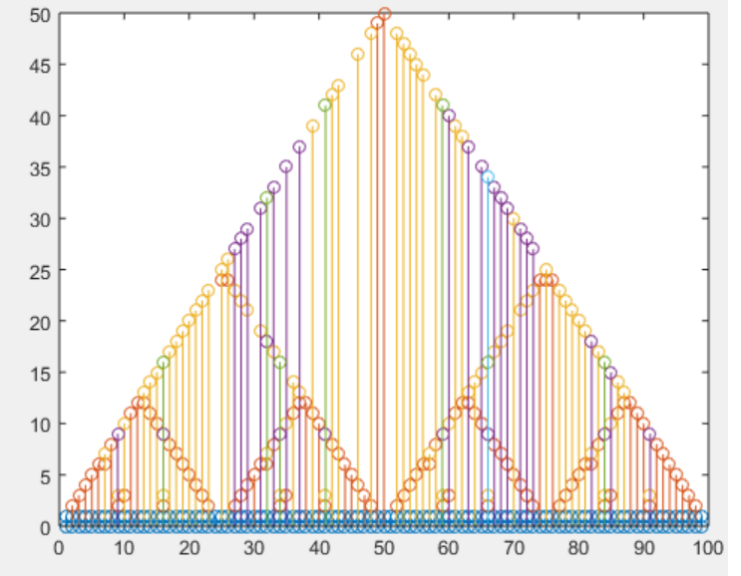


Evaluation

• 0	1	2	5	10	13	25	35	50	60	75	80	90	100
• 0	0	0	0	0	0	0	0	0	0	0	0	0	0
• 0	0(1)	0	0	0	0	0	0	0.4(50)	0.4(40)	0.4(25)	0.4(10)	0.4(10)	0
• 0	0(1)	0	0	0	0	0.16(25)	0.16(15~35)	0.4(50)	0.4(0~40)	0.64(25)	0.64(20)	0.64(10)	0
• 0	0(1)	0	0	0	0.064(12,13)	0.16(25)	0.16(15~35)						







Part 2

4.5 Asynchronous Dynamic Programming

- Drawback of the DP methods : they involve operations over the entire state set of the MDP -> If the state is very large, it can be prohibitively expensive

❖ Asynchronous DP algorithms

- : in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set
- update values of states in any order whatsoever

-> flexibility, real-time, focusing

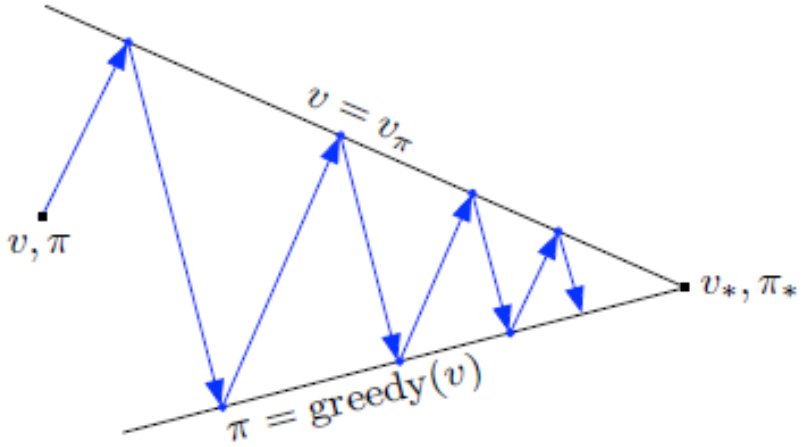
- Some ideas for doing this are discussed in Chapter 8

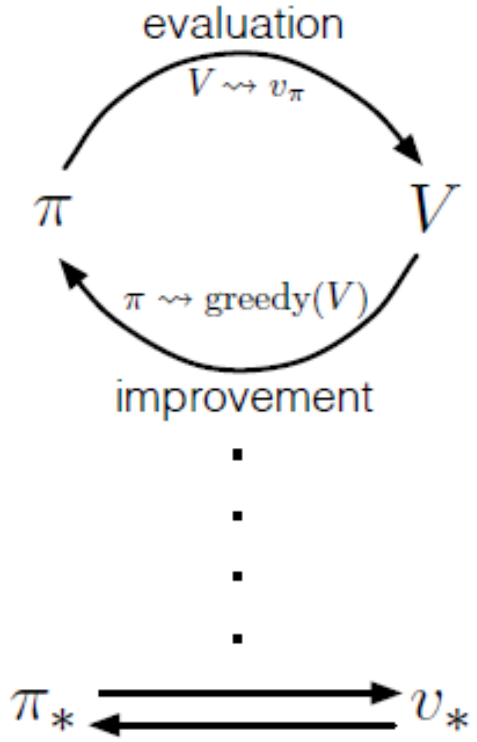
4.6 Generalized Policy Iteration

❖ Generalized policy iteration(GPI)

: general idea of letting policy-evaluation and policy-improvement processes interact, independent of the granularity and other details of the two processes.

- Both processes stabilize only when a policy has been found that is greedy with respect to its own evaluation function.





4.7 Efficiency of Dynamic Programming

❖ Efficiency of Dynamic Programming

- Worst case : polynomial in the number of states and actions.
- DP is exponentially faster than any direct search in policy space could be.
- In practice, DP methods can be used with today's computers to solve MPDs with millions of states.
- Both *policy iteration* and *value iteration* methods usually converge much faster than their theoretical worst-case run times, particularly if they are started with good initial value functions or polices.
- On problems with large state spaces, *asynchronous* DP methods are often preferred.

4.8 Summary

❖ Solutions for finite MPDs

- *Policy evaluation* : computation of the value functions for a given policy
- *Policy improvement* : computation of an improved policy given the value function for that policy.
- The most popular methods : *policy iteration* and value iteration
- Classical DP methods operate in sweeps, 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 (*bootstrapping*).
- Expected updates are closely related to Bellman equations.

4.8 Summary

❖ Reinforcement learning methods as GPI

- GPI is 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.
- In some cases, GPI can be proved to converge, most notably for the classical DP methods presented in this chapter. In other cases convergence has not been proved, but still the idea of GPI improves our understanding of the methods.
- In the next chapter, we will explore reinforcement learning methods that do not require a model and do not bootstrap. Look forward to it!

Thank you everyone and me!!

- Don’t you have any Questions?