4. Model-based Planning

2019 Fall

Yusung Kim
yskim525@skku.edu

Planning and Learning

Two fundamental problems in sequential decision making

- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy

What is Dynamic Programming?

- Simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.
  - Solve the sub-problems.
  - Combines sub-programs to a final solution

Requirements for Dynamic Programming Dynamic

- 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

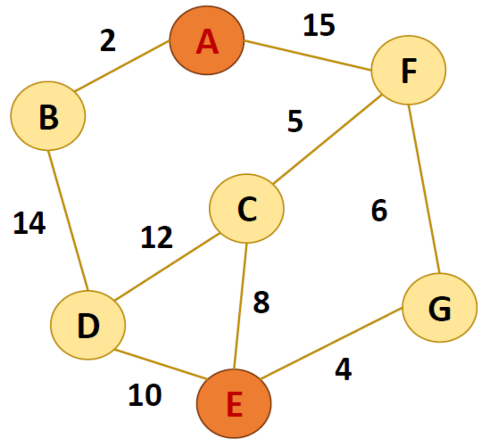
- Markov decision processes satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

Review: Shortest Path

- Definition
  - A path between two vertices in a graph such that the sum of the weights along the path is minimized

- Example: The shortest path from A to E
  - Path 1 (A-F-C-E): 15 + 5 + 8 = 28
  - Path 2 (A-F-G-E): 15 + 6 + 4 = 25
  - Path 3 (A-B-D-E): 2 + 14 + 10 = 26

  . . .



Planning by Dynamic Programming

- Assumes full knowledge of the MDP.
    - It is used for planning in an MDP
- For prediction (i.e., evaluation)
    - Input: MDP $\langle S, A, P, R, \gamma \rangle$ and policy π
    - Output: value function v_π

  어떤 바보 같은 policy여도 그에 맞춰 value function을 찾음.
- For control (i.e., improvement)
    - Input: MDP $\langle S, A, P, R, \gamma \rangle$
    - Output: optimal value function v_* and optimal policy π_*

Iterative Policy Evaluation

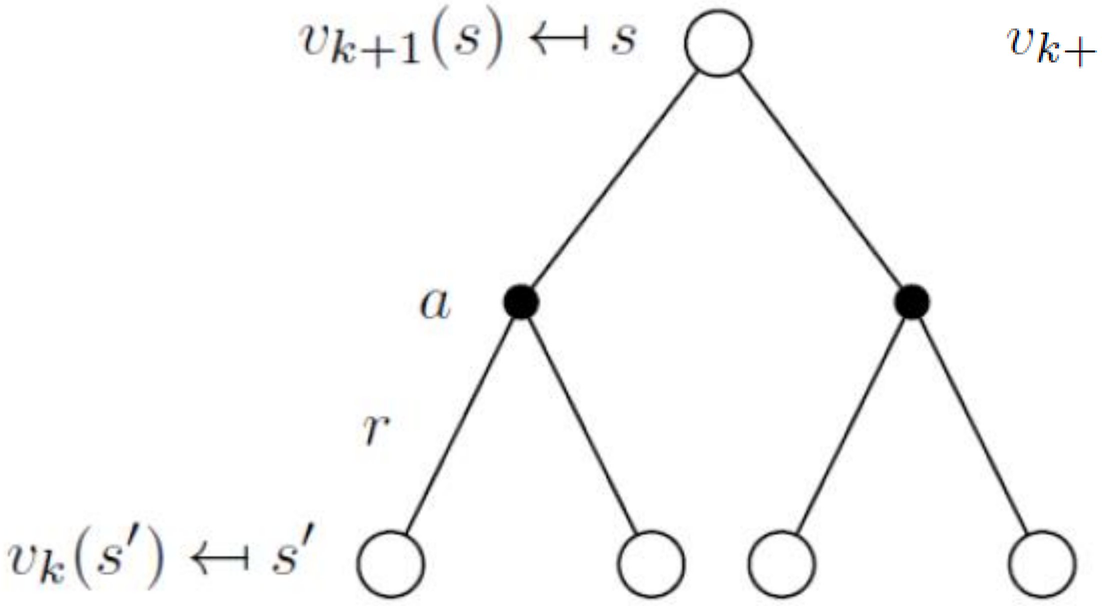
- How to evaluate a given policy π
  - iterative calculation of bellman expectation backup

- At each iteration k+1
- For all states $s \in S$
- Update $v_{k+1}(s)$ from $v_k(s')$

v파이에 수렴하게 반복함.

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

Bellman Expectation Equation

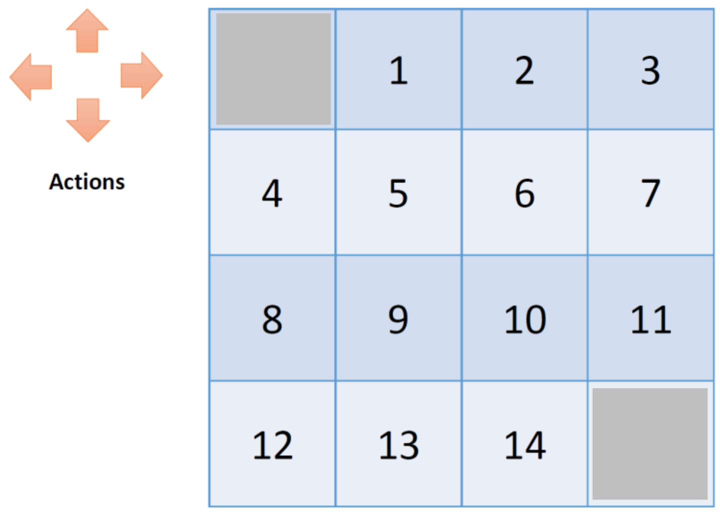


$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$

v_k **can converge to** v_π **as** $k \to \infty$

처음에는 쓰레기값이 들어있음.(초기화)
어떤 값에 수렴할때까지 반복함. 어느 순간 값이 더 바뀌지 않음.

Example: Small Grid World

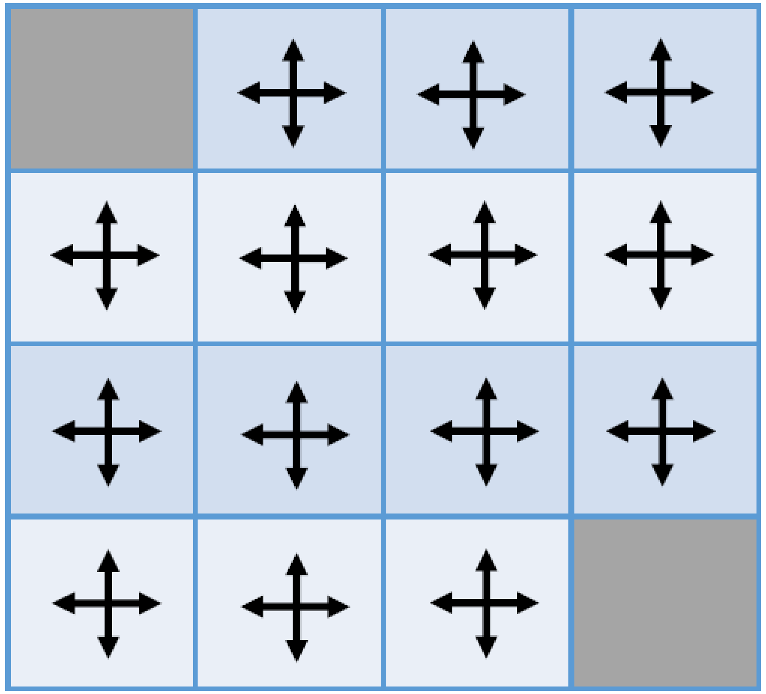


- Undiscounted episodic MDP (γ=1)
- Nonterminal states 1, ..., 14
- Terminal state at shared squares
- Actions leading out of the grid leave state unchanged.
- Reward is −1 for all movements

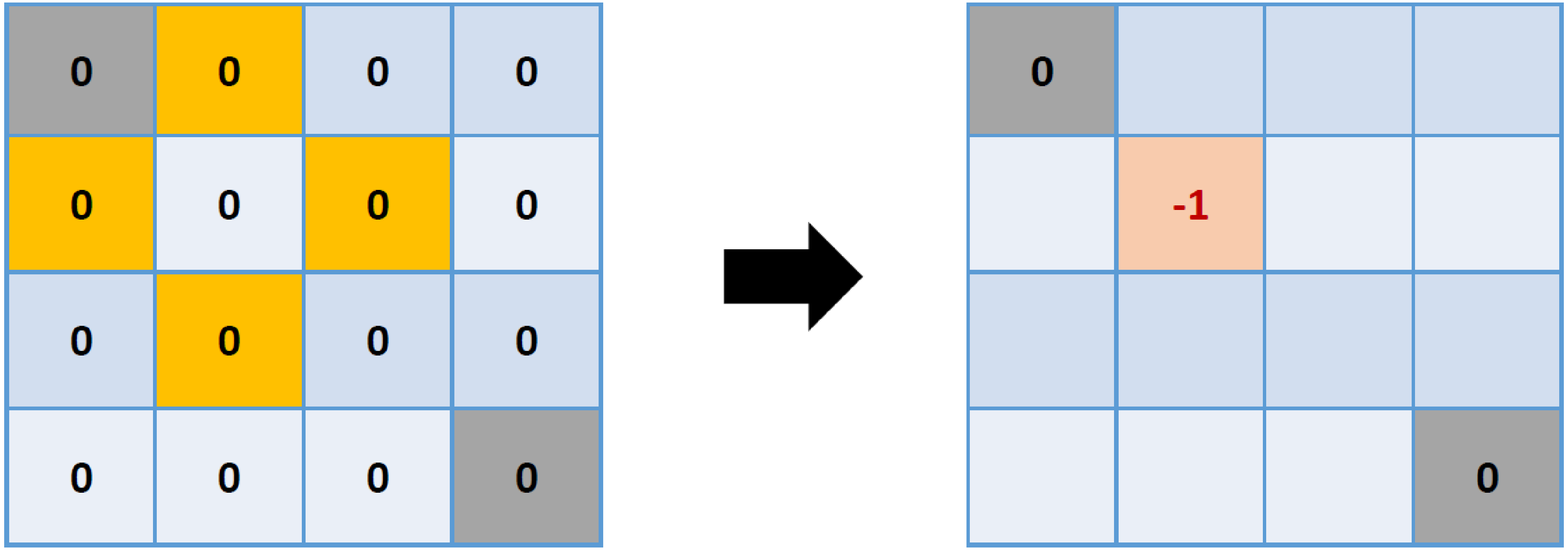
Small Grid World with Random Policy

- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$



Example: Policy Evaluation



$k = 0$ $k = 1$

Example: Policy Evaluation

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$k = 0$

0			
	-1		

Up: $v_1(s) = 0.25 \times (-1 + 0)$

Down: $v_1(s) = 0.25 \times (-1 + 0)$

Left: $v_1(s) = 0.25 \times (-1 + 0)$

Right: $v_1(s) = 0.25 \times (-1 + 0)$

$$v_1(s) = 4 \times 0.25 \times -1 = -1$$

Example: Policy Evaluation

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$k = 0$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k = 1$

After updating all states from $k = 0$

Example: Policy Evaluation

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k = 1$

	-1.75		

$k = 2$

Example: Policy Evaluation

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k = 1$

	-1.75		

Up: $v_2(s) = 0.25 \times (-1 - 1)$

Down: $v_2(s) = 0.25 \times (-1 - 1)$

Left: $v_2(s) = 0.25 \times (-1 + 0)$

Right: $v_2(s) = 0.25 \times (-1 - 1)$

$$v_2(s) = 3 \times 0.25 \times -2 + 0.25 \times -1 = -1.75$$

Example: Policy Evaluation

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k = 1$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

$k = 2$

After updating all states from $k = 1$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$k = 0$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k = 1$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

$k = 2$

0	-2.4	-2.9	-3
-2.4	-2.9	-3	-2.9
-2.9	-3	-2.9	-2.4
-3	-2.9	-2.4	0

$k = 3$

0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0

$k = 10$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

$k = \infty$

Algorithm: Iterative Policy Evaluation

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

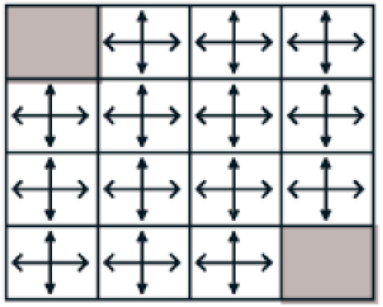
Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

v_k for the Random Policy

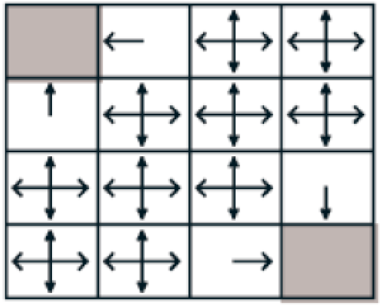
Greedy Policy w.r.t. v_k

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0



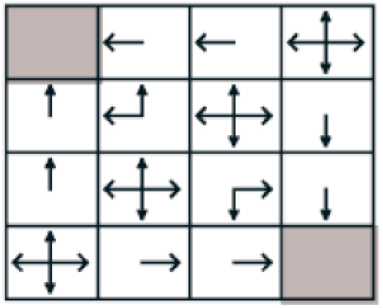
$k = 0$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0



$k = 1$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

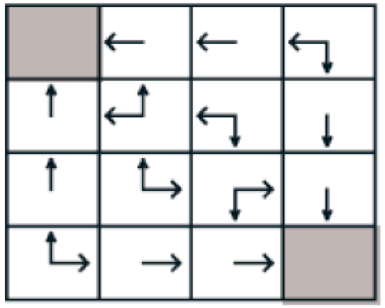


$k = 2$

v_k for the Random Policy

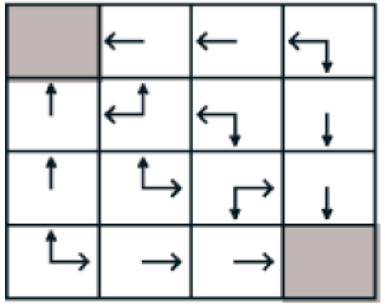
Greedy Policy w.r.t. v_k

0	-2.4	-2.9	-3
-2.4	-2.8	-3	-2.9
-2.9	-3	-2.8	-2.4
-3	-2.9	-2.4	0



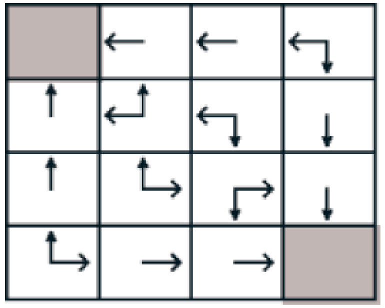
$k = 3$

0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0



$k = 10$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0



$k = \infty$

optimal policy

Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$

- We can improve the policy by acting greedily

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\, q_\pi(s, a)$$

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

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

첫 step은 파이프라임을 따라가고 나머지는 파이를 따라가는 것이
첫 step부터 전부 파이를 따라가는 것보다 좋다.

- It improves the value function,

$$v_{\pi'}(s) \geq v_\pi(s)$$

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_\pi(s')\Big].
\end{aligned} \tag{4.6}
$$

$$
q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{4.7}
$$

$$
v_{\pi'}(s) \geq v_\pi(s). \tag{4.8}
$$

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

Policy Improvement

- If improvements stop,

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

- Then the Bellman optimality equation has been satisfied

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

- Therefore, $v_\pi(s) = v_*(s)$ for all $s \in S$
- So π is an optimal policy

Policy Iteration

- Given a policy π

evaluate와 improve를 반복하면 optimal에 도달함.

  - Evaluate the policy π

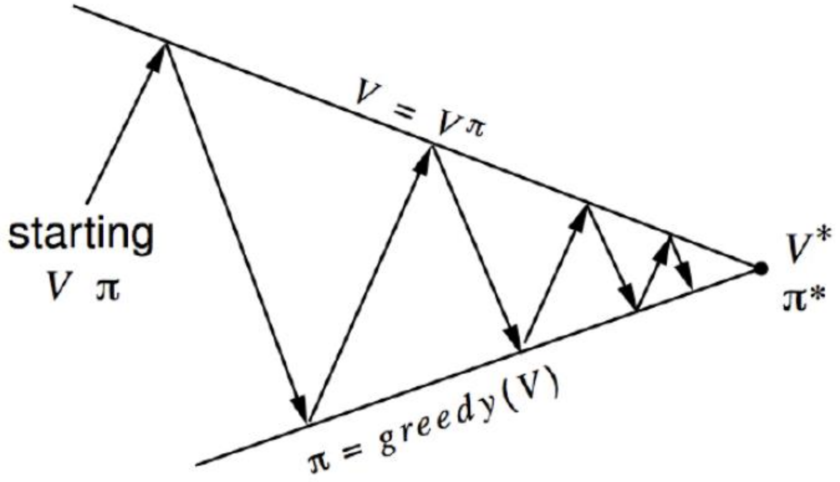
$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ \dots | S_t = s]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = greedy(v_\pi)$$

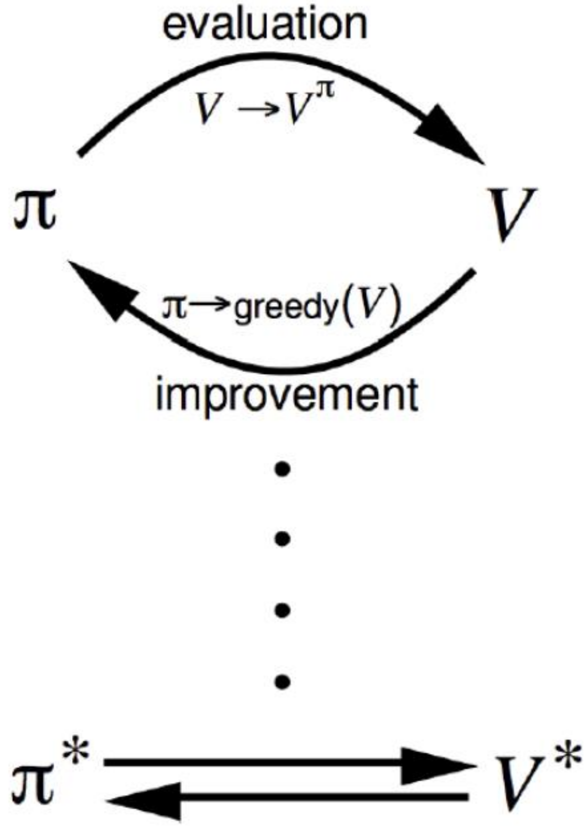
- In Small Grid World, a few times improved policy was optimal.
- In general, need more iterations of improvement.
- The policy iteration always converges to optimal policy π^*

Policy Iteration



Policy evaluation Estimate v_π
Iterative policy evaluation

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



Algorithm: Policy Iteration

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:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
   $\quad$ 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

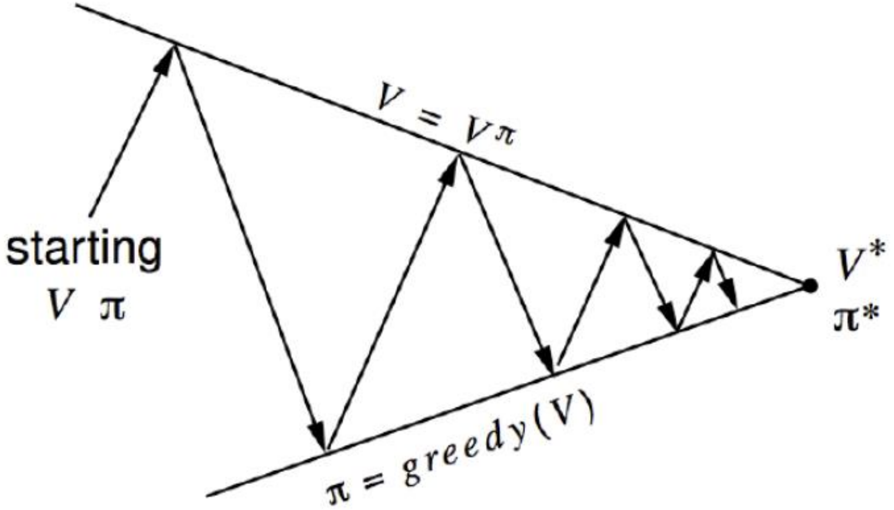
Modified Policy Iteration

- Does policy evaluation need to converge to v_π?

  policy evaluation단계에서 꼭 v파이에 수렴해야 하는가?

- Can we introduce a stopping condition ?
- Or simply stop after k iterations of iterative policy evaluation?
  - For example, in Small Grid World, $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration ? e.g. stop after $k = 1$
  - This is equivalent to value iteration (next slide)

Generalized Policy Iteration

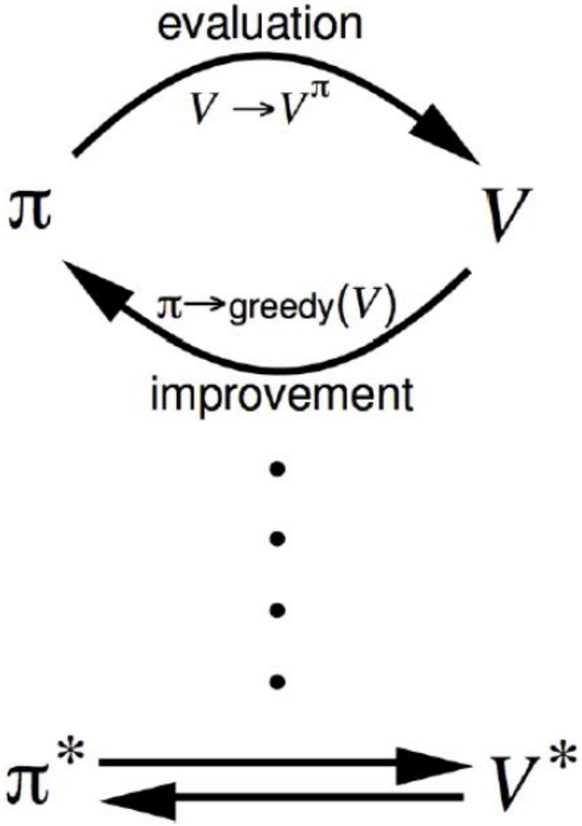


Policy evaluation Estimate v_π

Any policy evaluation algorithm

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

Any policy improvement algorithm



Principle of Optimality

- Any optimal policy can be subdivided into two components:
  - An optimal first action A
  - Followed by an optimal policy from successor state S'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

- *For any state s' reachable from s*
- *π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$*

Deterministic Value Iteration

policy없이 value만 가지고 update해나가는 것.

- If we know the solution to subproblems $v_*(s')$
- Then solution $v_*(s)$ can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

bellman optimality equation

- The idea of value iteration is to apply these updates iteratively
- Still works with loopy, stochastic MDPs

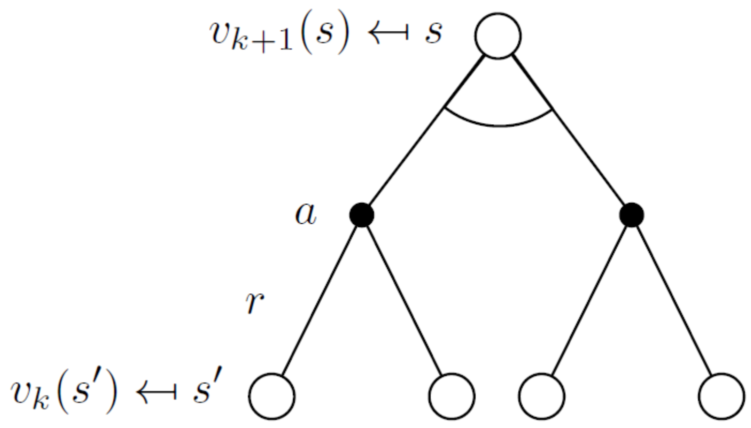
goal에서 시작해서 뒤로 가면서 구하는 것.

Value Iteration

- To find optimal policy
  - A case of policy iteration when policy evaluation is stopped after just one sweep
- Iterative application of bellman optimality backup
  - At each iteration $k+1$
  - For all states $s \in S$
  - Update $v_{k+1}(s)$ from $v_k(s')$

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

Value Iteration



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

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

Algorithm: Value Iteration

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

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

dynamic 중에 synchronous한 것만 배움.
한 타이밍에 모든 state를 update하는 것.

Summary

- 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

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**