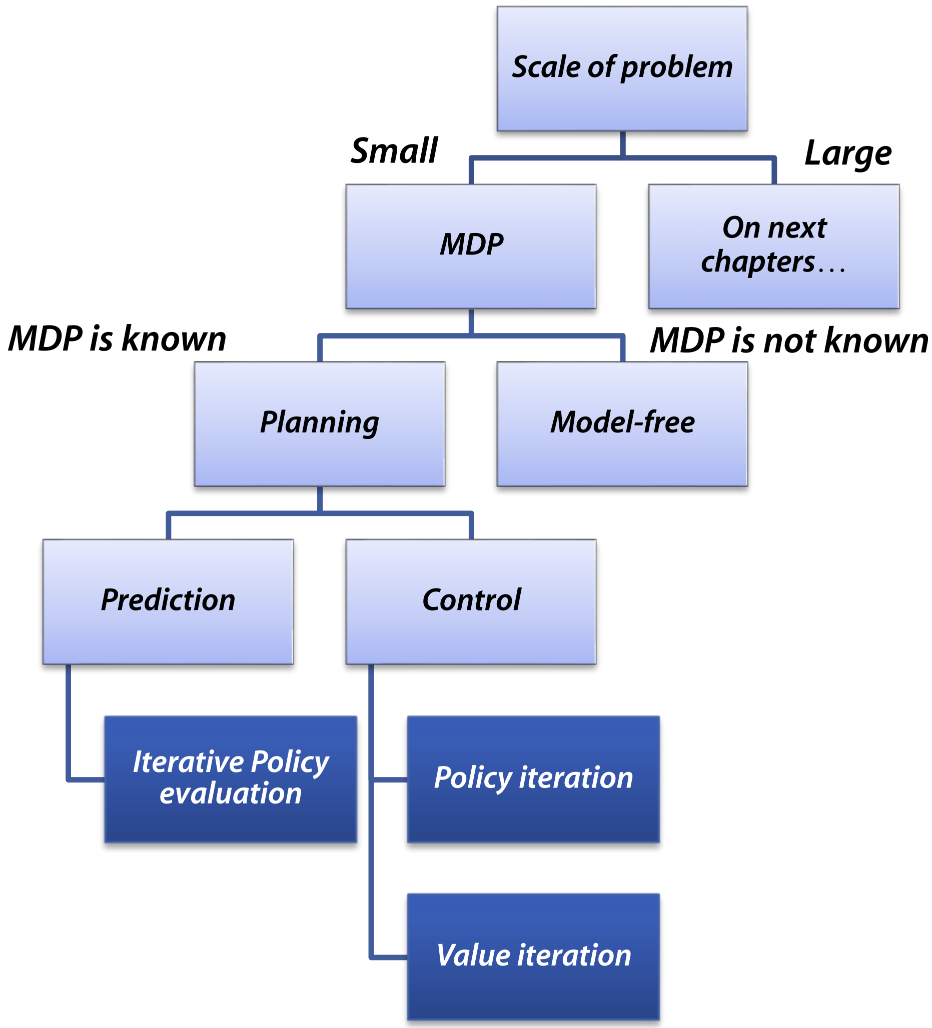
Introduction to Reinforcement Learning

2025. 1st semester

단국대학교
DANKOOK UNIVERSITY

Mobile System Engineering

Categories


Scale of problem
Small
Large
MDP
On next chapters…
MDP is known
MDP is not known
Planning
Model-free
Prediction
Control
Iterative Policy evaluation
Policy iteration
Value iteration

MDP planning

- ***When all information about MDP is known, the process of using it to improve policies is called planning.***

- ***Prediction***
  - ***Finding the state-values when π is given***
- ***Control***
  - ***Finding the optimal policy π_****

What is Dynamic Programming?

Dynamic sequential or temporal component to the problem

Programming optimising a "program", i.e. a policy

- c.f. linear programming

- A method for solving complex problems
- 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 processes satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

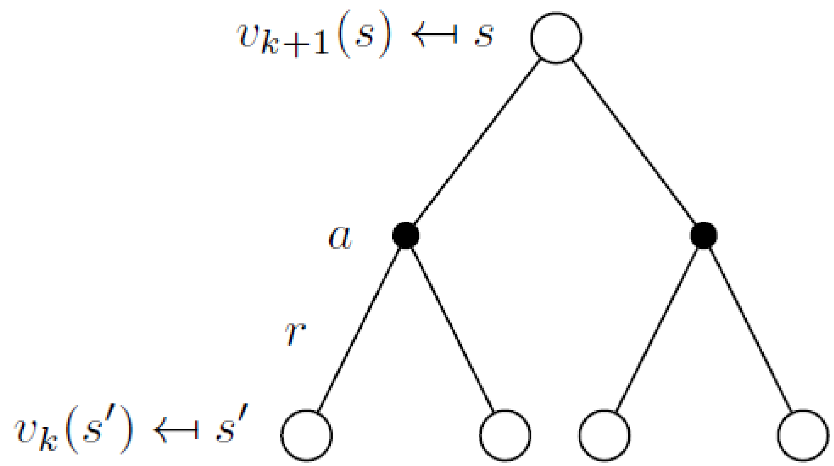
Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
  - 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
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
- 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

Iterative policy evaluation

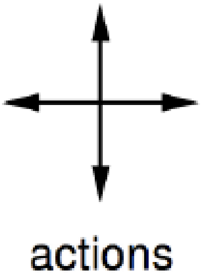


$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

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

Evaluating a Random Policy in the Small Gridworld

s0	*s1*	*s2*	*s3*
s4	*s5*	*s6*	*s7*
s8	*s9*	*s10*	*s11*
s12	*s13*	*s14*	*s15*



$r = -1$
on all transitions

- *Undiscounted episodic MDP ($\gamma = 1$)*
- *One terminal state s_{15}*
- *Actions leading out of the grid leave state unchanged*
- *Reward is -1 until the terminal state is reached (at the terminal state -> 0)*
- *Action follows uniform random policy*

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

Iterative policy evaluation

$v(s)$			
0.0	0.0	0.0	0.0
0.0	-1.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$v(s)$			
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

Iterative policy evaluation

state			
s0	***s1***	***s2***	***s3***
s4	***s5***	***s6***	***s7***
s8	***s9***	***s10***	***s11***
s12	***s13***	***s14***	***s15***

$v(s), k = 0$			
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$v(s), k = 1$			
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$v(s), k = 2$			
-2.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	***-1.75***
-2.0	-2.0	-1.75	0.0

$v(s), k = 3$			
-3.0	-3.0	-3.0	-3.0
-3.0	***-3.0***	***-3.0***	-2.94
-3.0	***-3.0***	-2.88	-2.4
-3.0	-2.94	-2.4	0.0

$v(s), k = \infty$			
-59.4	-57.4	-54.3	-51.7
-57.4	-54.6	-49.7	-45.1
-54.3	-49.7	-40.9	-30
-51.7	-45.1	-30	0.0

How to Improve a Policy

- Given a policy π
  - Evaluate the policy π

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

  - Improve the policy by acting greedily with respect to v_π

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

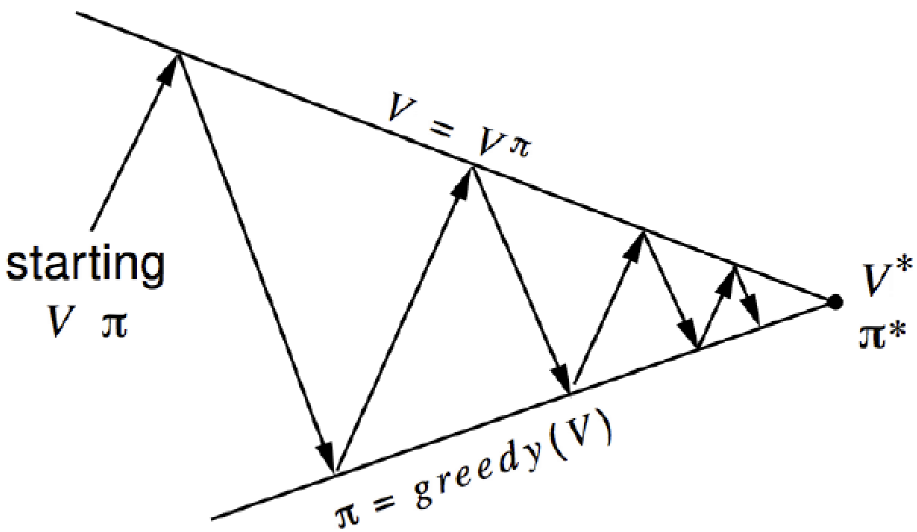
- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi *$

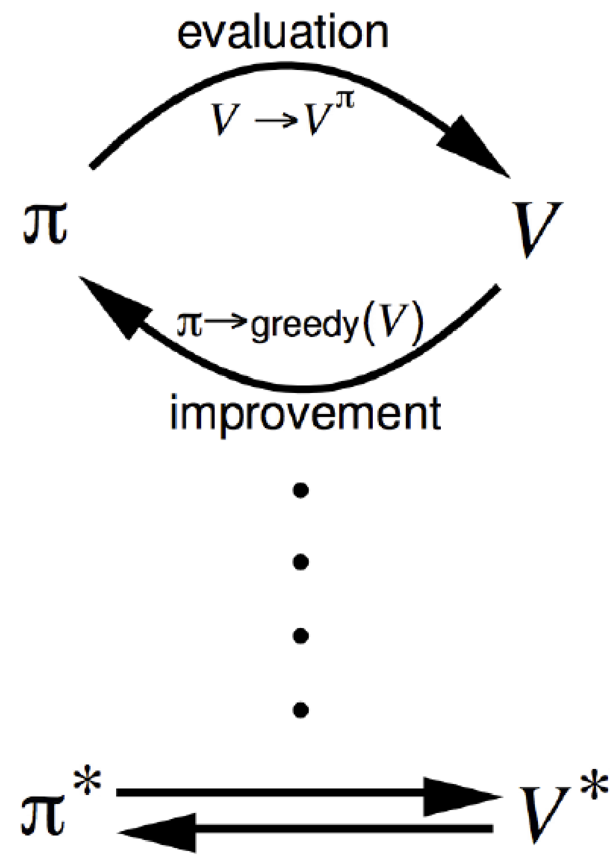
Greedy Policy

$v(s), k = \infty$			
-59.4	-57.4	-54.3	-51.7
-57.4	-54.6	-49.7	-45.1
-54.3	-49.7	-40.9	-30
-51.7	-45.1	-30	0.0

Greedy policy			
↓→	↓→	↓→	↓
↓→	↓→	↓→	↓
↓→	↓→	↓→	↓
→	→	→	F

Policy Iteration

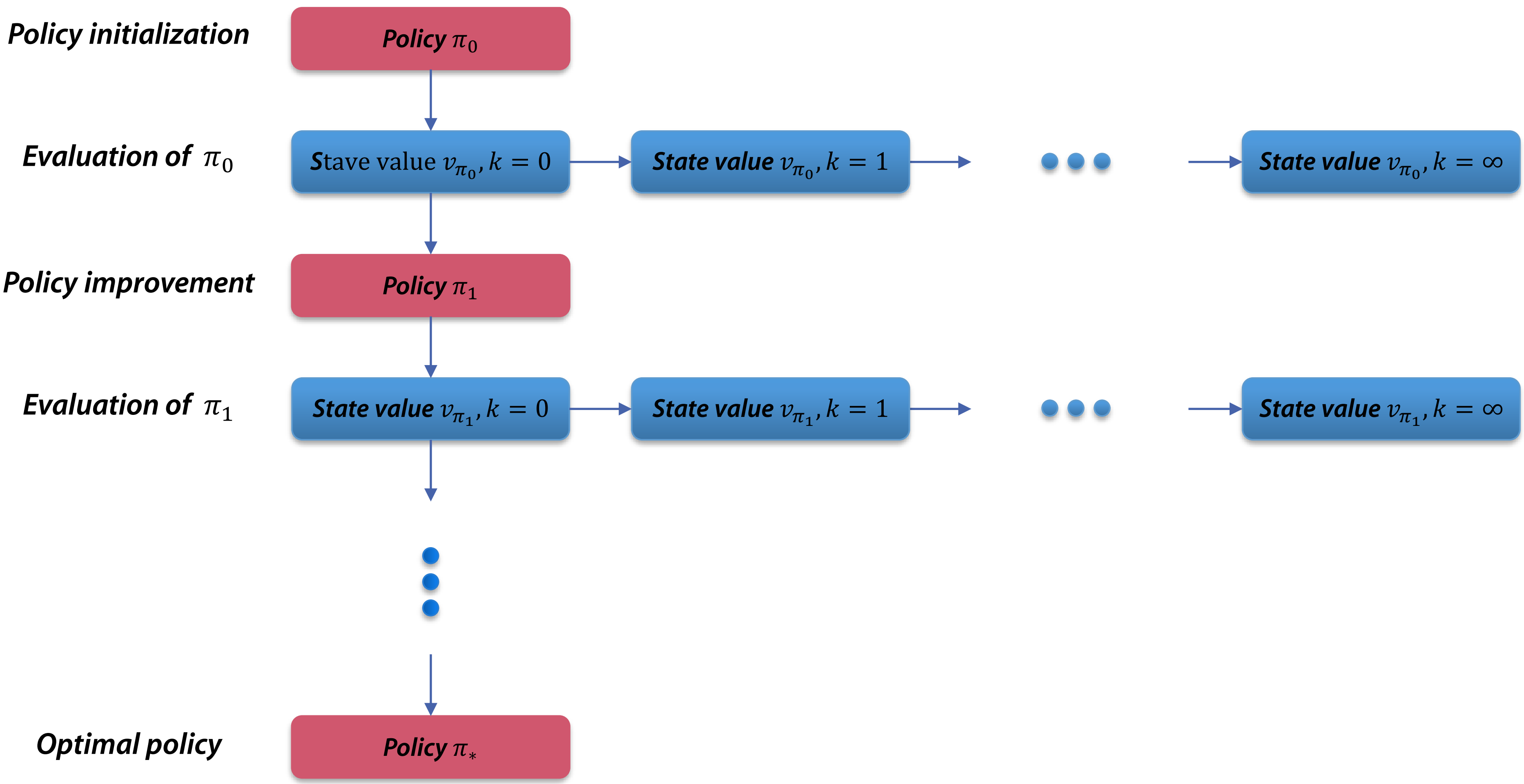




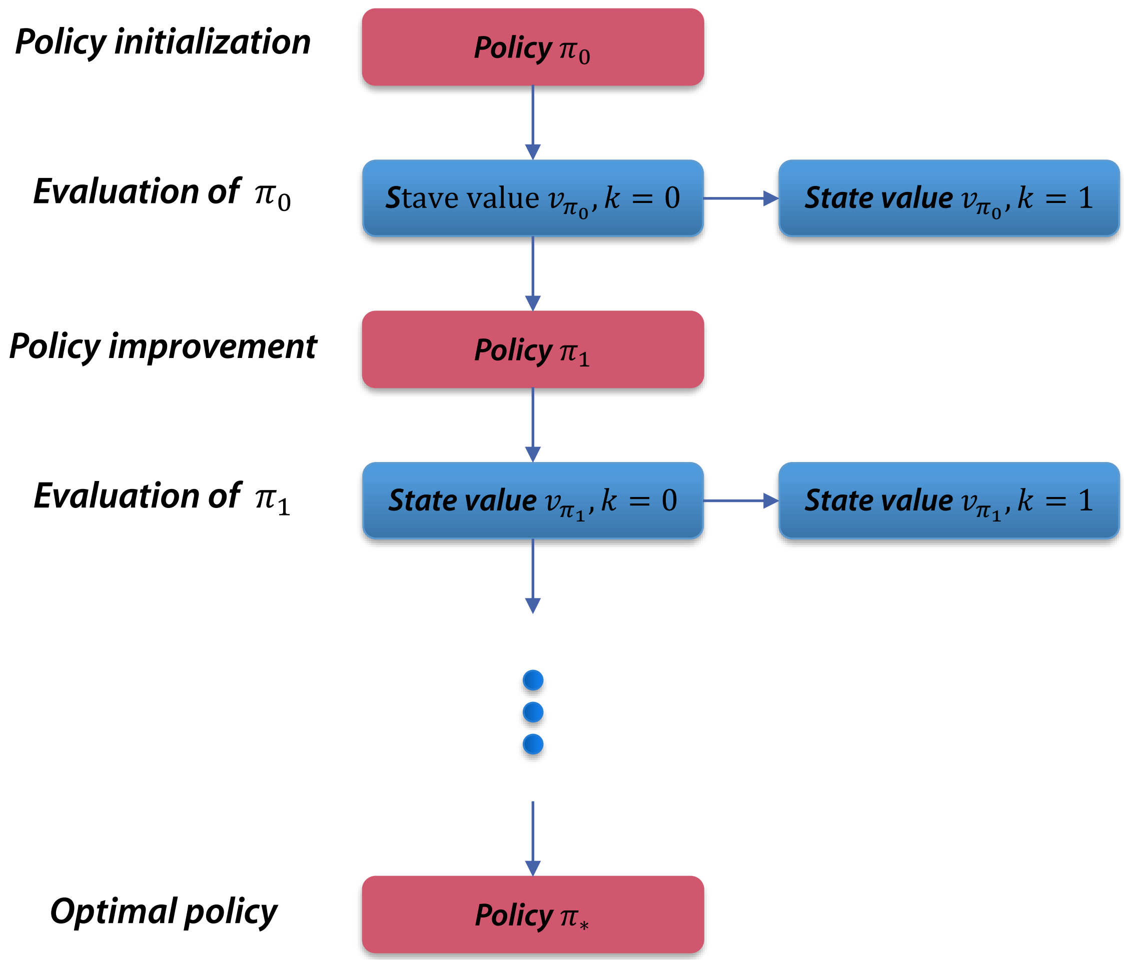
Policy evaluation Estimate v_π
Iterative policy evaluation

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

Policy iteration



Early stopping



Early stopping

$v(s), k = 6$			
-6	-5.99	-5.96	-5.90
-5.99	***-5.95***	***-5.80***	-5.55
-5.96	***-5.80***	-5.27	-4.22
-5.90	-5.55	-4.22	0.0

$v(s), k = \infty$			
-59.4	-57.4	-54.3	-51.7
-57.4	-54.6	-49.7	-45.1
-54.3	-49.7	-40.9	-30
-51.7	-45.1	-30	0.0

Policy improvement

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

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

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

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

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

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

Policy improvement

- 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 $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Deterministic Value Iteration

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

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

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards
- Still works with loopy, stochastic MDPs

Value iteration

- $v_*(s) = \max_a \left[r_S^a + \gamma \sum_{s' \in S} P_{s,s'}^a v_*(s') \right]$
  - $r_S^a = -1$ ***for all actions***
  - $\gamma = 1$
  - $P_{s,s'}^a = 1$ ***for all actions and states***
  - $v_*(s') = 0$

- $v_*(s_5) = \max(-1 + 1.0 * 0,$
  $-1 + 1.0 * 0,$
  $-1 + 1.0 * 0,$
  $-1 + 1.0 * 0,)$
  $= -1.0$

Optimal state value			
s0	***s1***	***s2***	***s3***
s4	***s5***	***s6***	***s7***
s8	***s9***	***s10***	***s11***
s12	***s13***	***s14***	***F***

Optimal state value			
0.0	0.0	0.0	0.0
0.0	-1.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Value iteration

State			
s0	***s1***	***s2***	***s3***
s4	***s5***	***s6***	***s7***
s8	***s9***	***s10***	***s11***
s12	***s13***	***s14***	종료

Optimal state value, $k = 0$			
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Optimal state value, $k = 1$			
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

Optimal state value, $k = 2$			
-2.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	***-1.0***
-2.0	-2.0	-1.0	0.0

Optimal state value, $k = 3$			
-3.0	-3.0	-3.0	-3.0
-3.0	***-3.0***	***-3.0***	-2.94
-3.0	***-3.0***	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

Optimal state value, $k = \infty$			
-6.0	-5.0	-4.0	-3.0
-5.0	-4.0	-3.0	-2.0
-4.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

Value iteration

- *Let's intuitively think about how many steps it would take to follow the optimal policy from the starting state to the ending point.*
  - *Result of multiple applications of the Bellman Optimal Equation*
- *Now that we have found the optimal value, we can find the optimal policy*
  - *You can move to the square with the highest optimal value.*
  - *That is, the greedy policy for optimal value*

Optimal state value, $k = \infty$			
-6.0	-5.0	-4.0	-3.0
-5.0	-4.0	-3.0	-2.0
-4.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0