L14. Markov Decision Process
(Dynamic Programming Approach)

L14. Markov Decision Process
(Dynamic Programming Approach)

1. Policy Evaluation
2. Policy Improvement
3. Policy iteration
4. Value Iteration

Summary

$$V^\pi(s) = \mathrm{E}\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\}$$
$$= \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^\pi(s')\}$$

$$Q^\pi(s, a) = \mathrm{E}\{r_{t+1} + \gamma Q^\pi(s, \pi(s_{t+1})) | s_t = s, a_t = a\}$$
$$= \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma Q^\pi(s, \pi(s'))]$$
$$= \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi(s')]$$

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

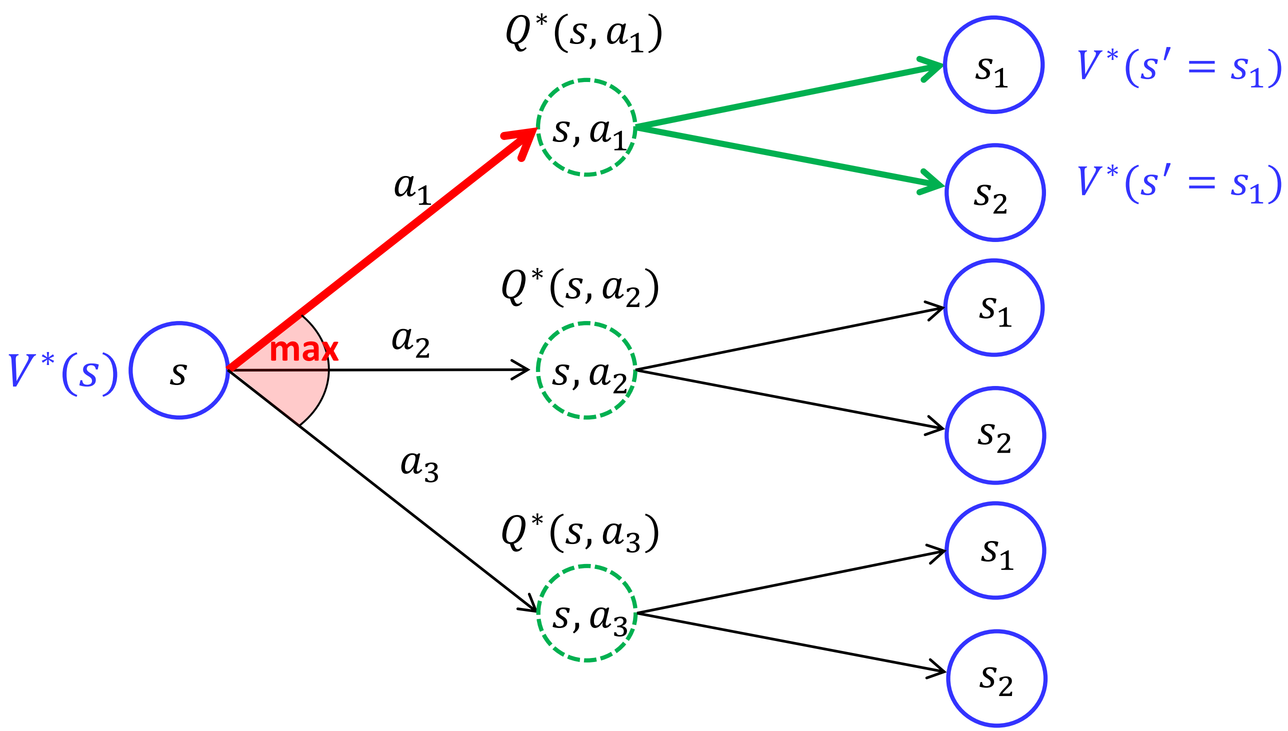
$$V^*(s) = \max_{a \in \mathcal{A}(s)} \mathrm{E}\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s\}$$
$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V^*(s')\}$$

$$Q^*(s, a) = \mathrm{E}\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a\right\}$$
$$= \sum_{s'} T(s, a, s')\left\{R(s, a, s') + \gamma \max_{a'} Q^*(s', a')\right\}$$
$$= \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V^*(s')\}$$

$$\because V^*(s') = \max_{a'} Q^*(s', a')$$

Summary

Bellman optimality equation for $V^*(s)$

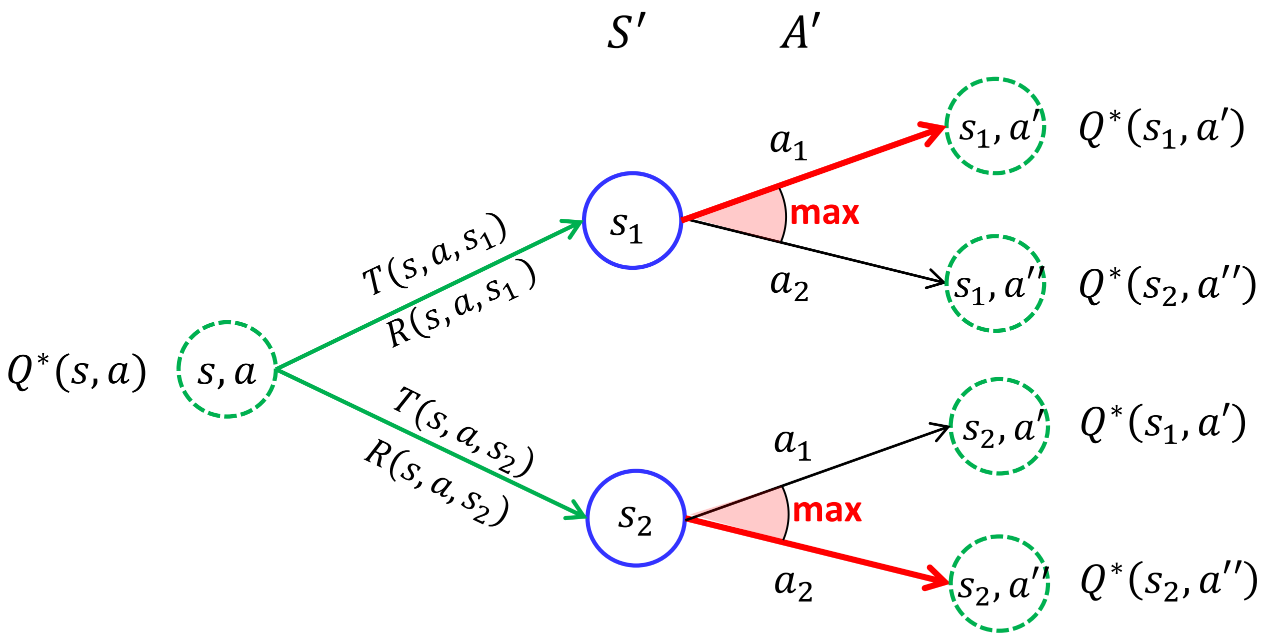


$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V^*(s')\}$$

$$\because V^*(s) = \max_{a'} Q^*(s, a')$$

Summary

Bellman optimality equation for $Q^*(s,a)$



$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right\}$$

$$= \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\} \qquad \because V^*(s') = \max_{a'} Q^*(s',a')$$

Dynamic Programming

- The term **dynamic programming (DP)** refers to a collection of algorithms that can be used to compute optimal polices given a perfect model of the environment as a Markov decision process (MDP)
- The key idea of DP (and reinforcement learning) is the use of value functions to organize and structure the search for good policies
- Optimal policies can be derived from the optimal value functions that satisfy the Bellman optimality equations

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\}$$

$$\begin{aligned} Q^*(s,a) &= \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \max_{a'} Q^*(s',a')0\right\} \\ &= \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\} \qquad \because V^*(s') = \max_{a'} Q^*(s',a') \end{aligned}$$

Optimal policy

$$\begin{aligned} \pi^*(s) &= \operatorname*{argmax}_a Q^*(s,a) \\ &= \operatorname*{argmax}_a \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\} \end{aligned}$$

Recycling Robot Example
Systems of equations for the optimum Bellman function

Policy evaluation :
A method to compute the state-value function $V^{\pi}(s)$ for an arbitrary policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^{\pi}(s')\}$$

- A system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknown

Algorithm

Initialize $V^{\pi}_{t=0}(s) \leftarrow 0$ for all states $s \in S$

Repeat (iteration $t = 0, \dots$):

For each state s:

$$V^{\pi}_{t+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^{\pi}_{t}(s')\}$$

Until $\max_{s \in \mathcal{S}} |V^{\pi}_{t+1} - V^{\pi}_{t}(s)| \leq e$

Full backup:
Each iteration of iterative policy evaluation backs up the value of every state once to produce the new approximate value function V^{π}_{t+1}

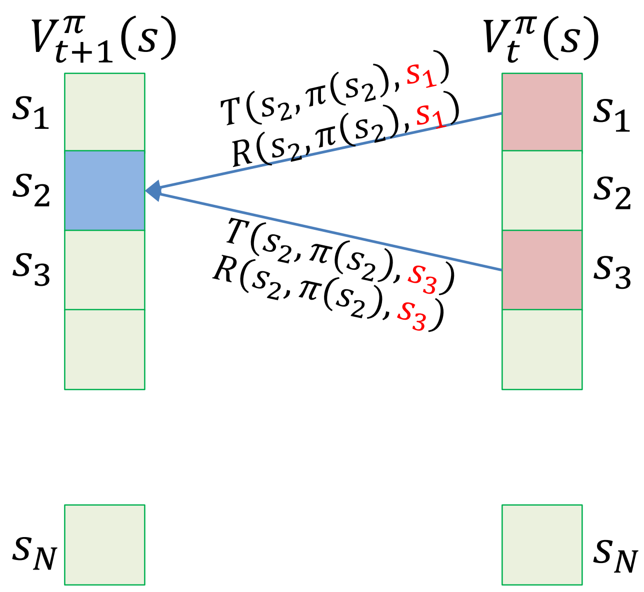
Policy Evaluation

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma V_t^{\pi}(s')\}$$

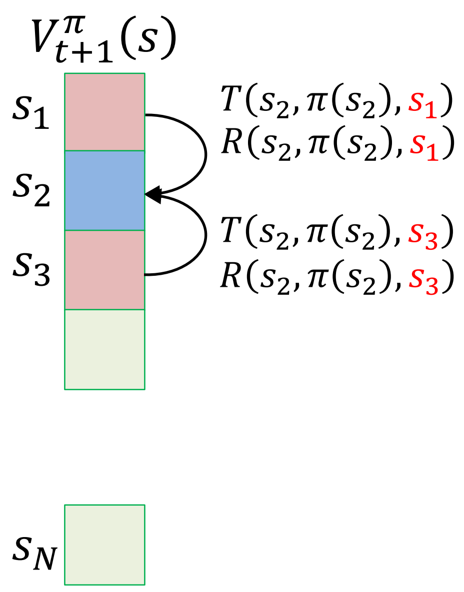
Example:

$$V_{t+1}^{\pi}(s_2) = \sum_{s'} T(s_2,\pi(s_2),s')\{R(s_2,\pi(s),s') + \gamma V_t^{\pi}(s')\}$$
$$= T(s_2,\pi(s_2),s_1)\{R(s_2,\pi(s),s_1) + \gamma V_t^{\pi}(s_1)\} + T(s_2,\pi(s_2),s_3)\{R(s_2,\pi(s_2),s_3) + \gamma V_t^{\pi}(s_3)\}$$

"Two-arrays" update



"In place" update



Usually faster!
Less memory

Example : Grid world

★	1	2	3
4	5	6	7
8	9	10	11
12	13	14	★

$MDP = \{\mathcal{S}, \mathcal{A}, T, R, \gamma\}$

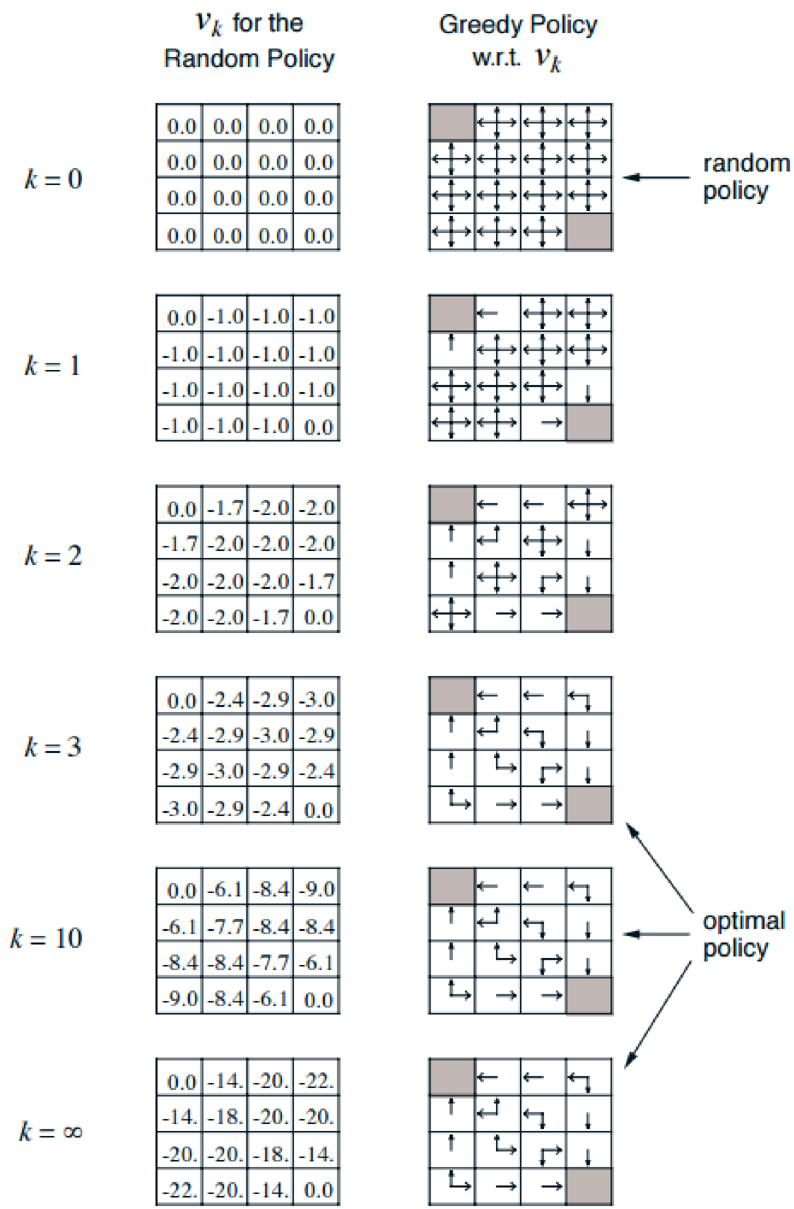
- $\mathcal{S} = \{1, 2, \dots, 14\}$
- $\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$
- $T(s, s', a) = \begin{cases} 1, \text{if move is allowed} \\ 0, \text{if move is not allowed} \end{cases}$

  $T(5,6,\rightarrow) = 1 \quad T(5,10,\rightarrow) = 0 \quad T(7,7,\rightarrow) = 1$

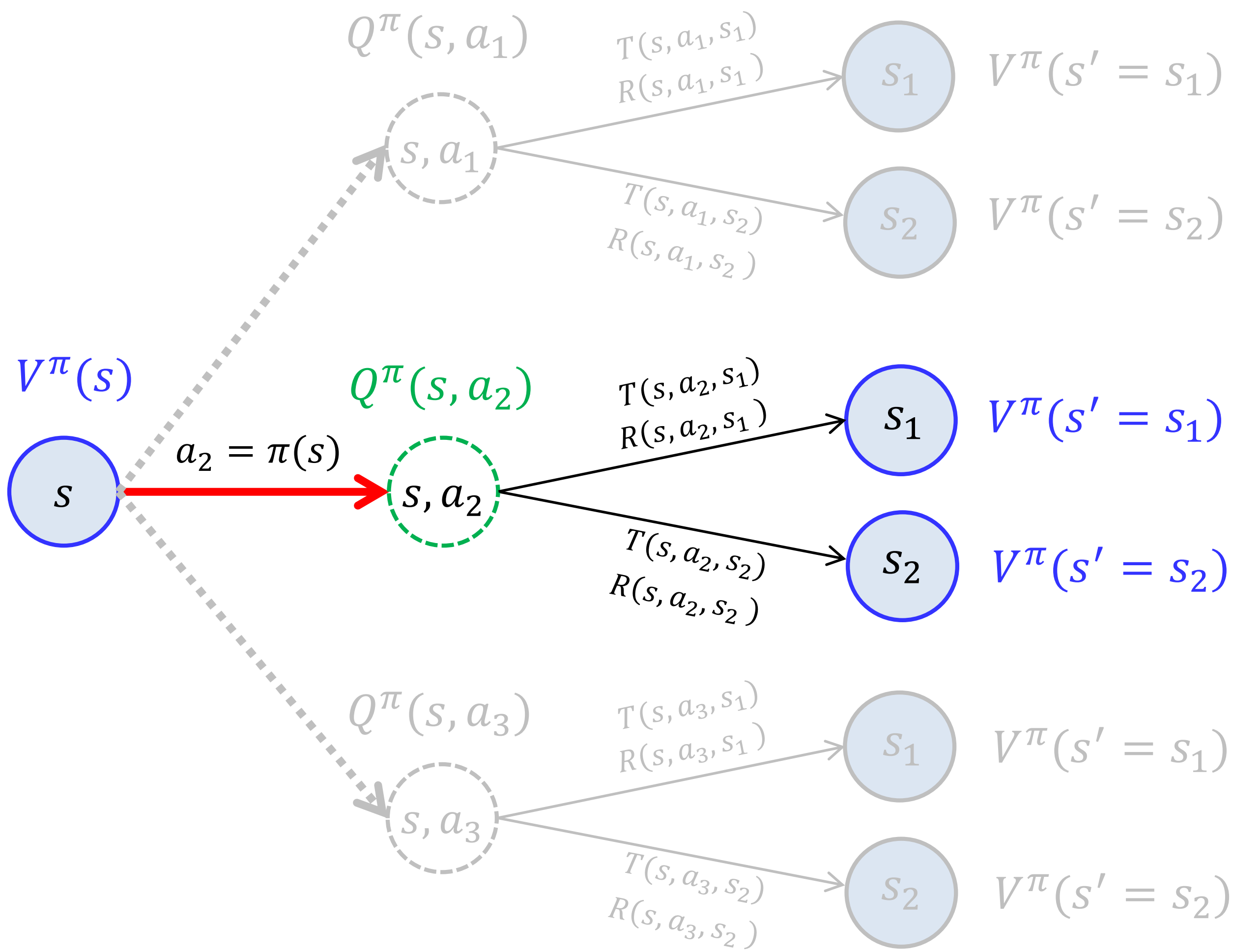
  The actions that would take the agent off the grid leave the state unchanged
- $R(s, s', a) = -1$ for all s, s', a
- $\gamma = 1$

Suppose the agent follows the equiprobable random policy (all actions equality likely), what is the value function?

Policy Evaluation



Policy Improvement

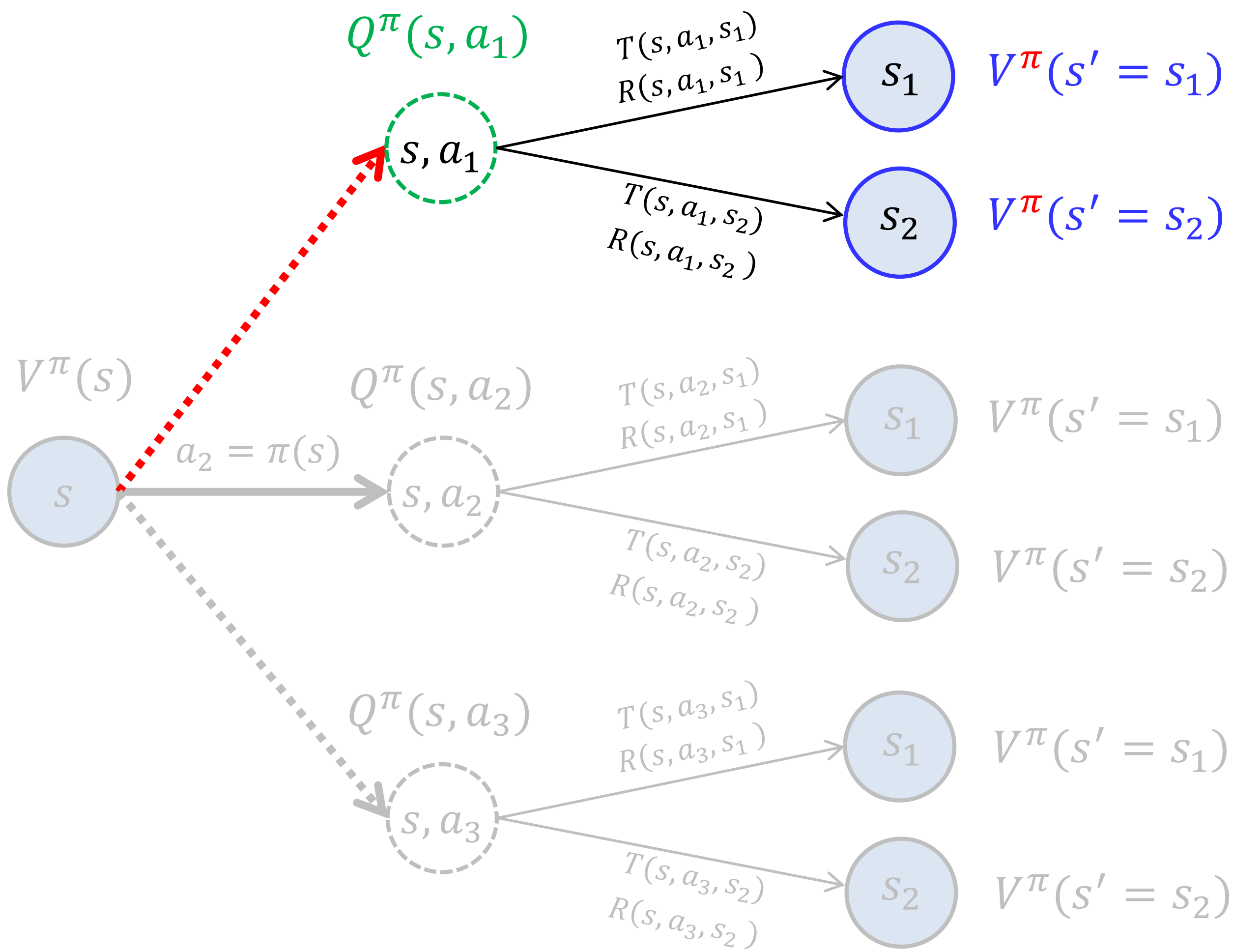


$$V^\pi(s) = Q^\pi(s, \pi(s))$$

We know how good it is to follow the current policy from s based on $V^\pi(s)$
Would it be better or worse to change to the new policy?

→ Select a given s and thereafter following the existing policy π (a single step change)

Policy Improvement

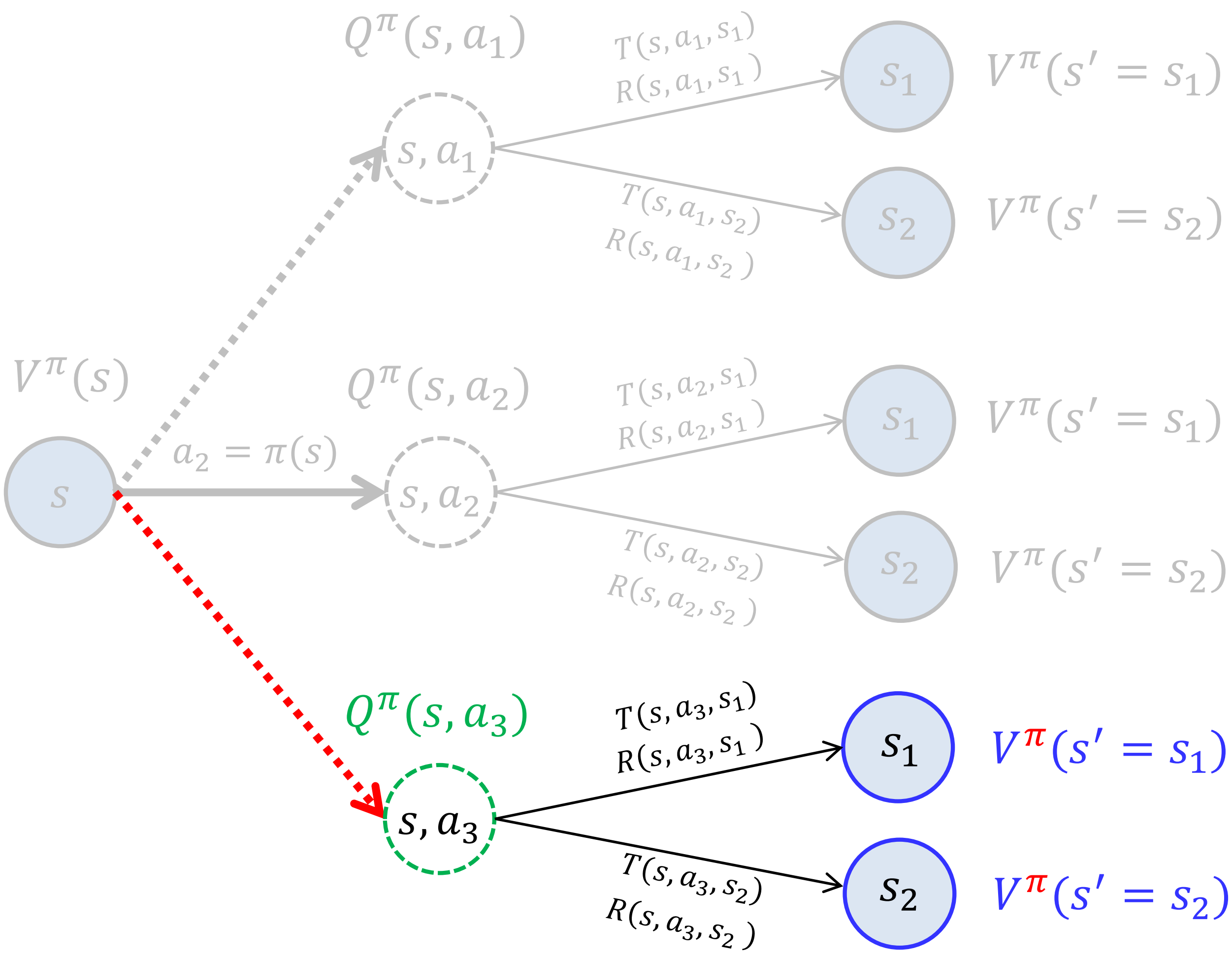


$Q^{\pi}(s,a_1)$

$V^{\pi}(s)=Q^{\pi}(s,\pi(s))$

$Q^{\pi}(s,a_1) \geq V^{\pi}(s) = Q^{\pi}(s,\pi(s))?$

Policy Improvement



$V^\pi(s) = Q^\pi(s, \pi(s))$

$Q^\pi(s, a_3)$

$$Q^\pi(s, a_3) \geq V^\pi(s) = Q^\pi\ (s, \pi(s))?$$

Policy Improvement

$$\pi'(s) = \operatorname*{argmax}_{a \in \mathcal{A}(s)} Q^{\pi}(s, a)$$

$$\rightarrow Q^{\pi}(s, \pi'(s)) \geq Q^{\pi}\big(s, \pi(s)\big) = V^{\pi}(s)$$

$$x^* = \operatorname*{argmax}_{x} f(x)$$

$\rightarrow f(x^*) \geq f(x)$ for all x

Improvement criterion =
Expected reward provided by changing one step action and following the original policy

If it is better to select $a = \pi'(s)$ once in s and thereafter follow π than it would be to follow π all the time,

?

It is better still to select $a = \pi'(s)$ whenever s is encountered

(The new policy $\pi'(s)$ is a better policy overall)

Policy Improvement

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

$$\rightarrow Q^{\pi}(s,\pi'(s)) \geq Q^{\pi}\big(s,\pi(s)\big) = V^{\pi}(s)$$

$$x^* = \underset{x}{\operatorname{argmax}}\, f(x)$$

$\rightarrow f(x^*) \geq f(x)$ for all x

Improvement criterion =
Expected reward provided by changing one step action and following the original policy

Proof (Policy improvement Theorem)

Policy improvement must give us a strictly better policy $\pi'(s)$ than the older policy $\pi(s)$ except when the original policy is already optimal $\pi(s) = \pi^*(s)$

$$Q^{\pi}\big(s,\pi'(s)\big) \geq V^{\pi}(s) \quad \rightarrow \quad \underbrace{V^{\pi'}(s) \geq V^{\pi}(s) \text{ for all states } s \in \mathcal{S}}_{\pi' \geq \pi}$$

Proof (Policy improvement Theorem)

Policy improvement must give us a strictly better policy $\pi'(s)$ than the older policy $\pi(s)$ except when the original policy is already optimal $\pi(s) = \pi^*(s)$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \rightarrow V^{\pi'}(s) \geq V^\pi(s)$$ for all states $s \in \mathcal{S}$

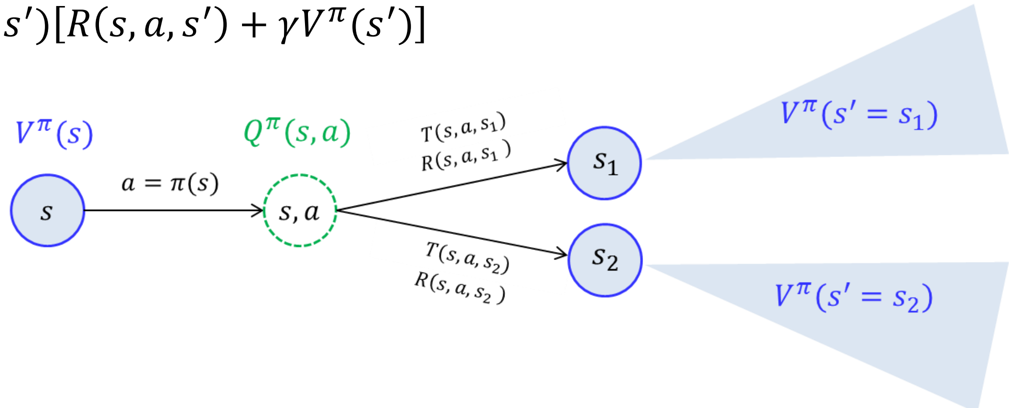
$$
\begin{aligned}
V^\pi(s) &\leq Q^\pi(s, \pi'(s)) && \text{Given} \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s] && \mathbb{E}_{\pi'} \text{ is expectation over } s_{t+1} \text{ induced by } \pi' \\
&\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s] && \because V^\pi(s_{t+1}) \leq Q^\pi(s_{t+1}, \pi'(s_{t+1})) \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \mathbb{E}_{\pi'}[r_{t+2} + \gamma V^\pi(s_{t+2})] | s_t = s] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) | s_t = s] \\
&\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q^\pi(s_{t+2}, \pi'(s_{t+2})) | s_t = s] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 \mathbb{E}_{\pi'}[r_{t+3} + \gamma V^\pi(s_{t+3})] | s_t = s] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) | s_t = s] \\
&\;\;\vdots \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots | s_t = s] \\
&= V^{\pi'}(s)
\end{aligned}
$$

Thus, $\pi \leq \pi'$

Policy Improvement

Policy improvement :
The process of making a new policy π^{new} that improves the original policy π, by making it greedy or nearly greedy with respect ot the value function of the original policy

Recall: $$Q^{\pi}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$$



Algorithm

Input : value of policy $V^{\pi}(s)$

Output: new policy π'

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

~~1. Compute $Q^{\pi}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$ for each a~~

2. Compute $\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} Q^{\pi}(s,a)$

$$= \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$$

Policy Iteration

Policy iteration :
Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement

$$\pi_0 \xrightarrow{\text{PE}} V^{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} V^{\pi_1} \xrightarrow{\text{PI}} \pi_2 \xrightarrow{\text{PE}} V^{\pi_2} \xrightarrow{\text{PI}}$$

Algorithm

$\pi \leftarrow$ arbitrary
For $t = 1, \ldots, t_{PI}$ (or until π stops changing)
Run policy evaluation to compute V^{π}
Run policy improvement to get new improved policy π'
$\pi \leftarrow \pi'$

- Policy evaluation require iterative computation, requiring multiple sweeps through the state set
- policy evaluation starts with the value function for the *previous policy*

→ *Fast convergence*

Policy Iteration

Policy iteration :
Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement

Iteration

For $t = 0, \ldots$ until convergence

For each state s:

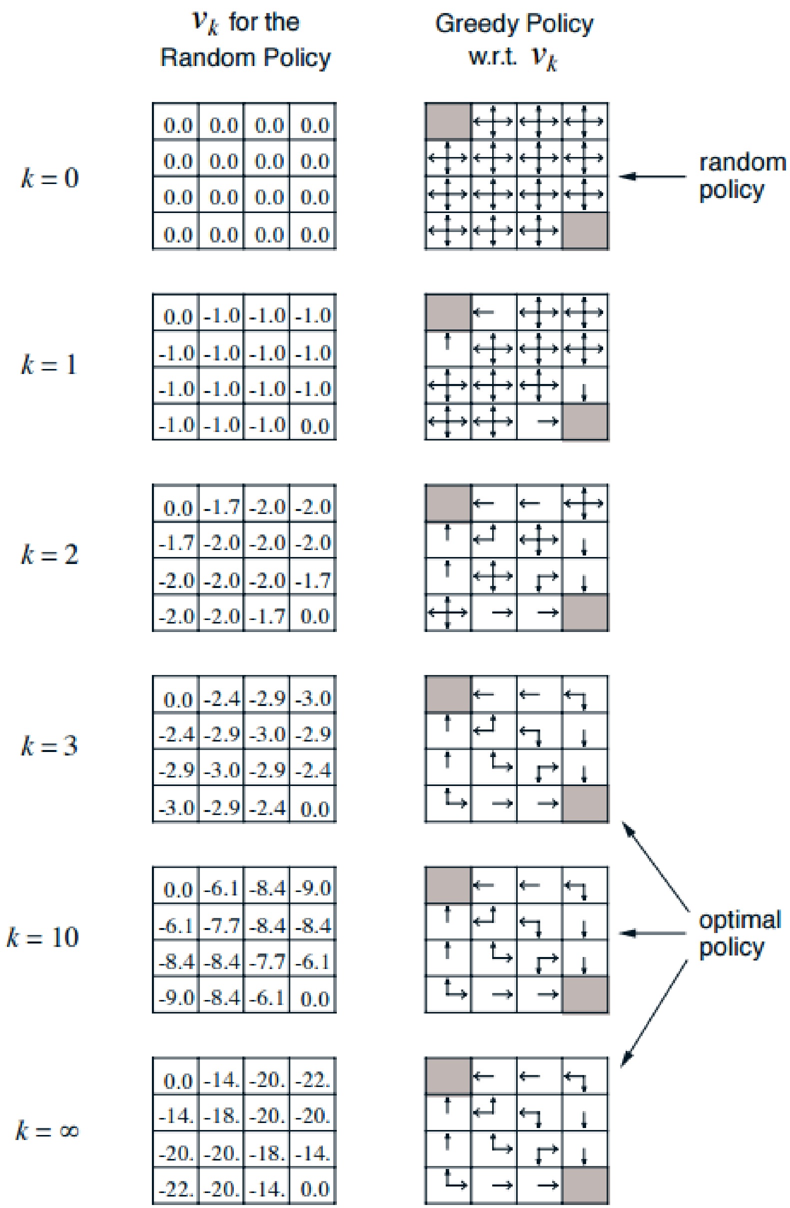
$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

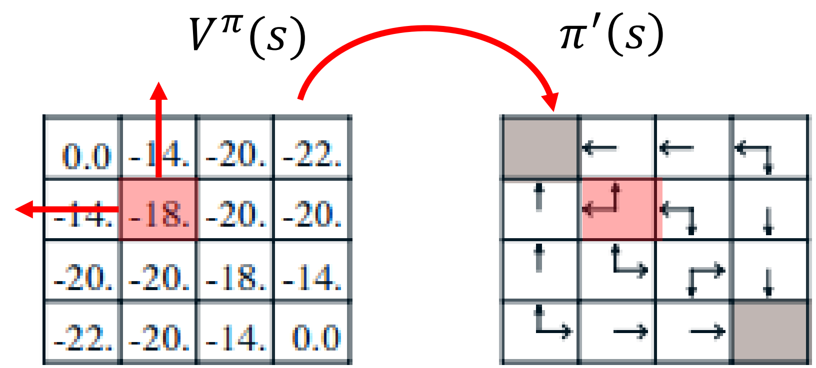
Converged state value function $V^{\pi}(s)$

For each state s:

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} Q^{\pi}(s, a)$$
$$= \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')]$$

Policy Iteration





Policy Iteration

Policy iteration :
Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement

Iteration

For $t = 0, \ldots$ until convergence

For each state s:

$$V^{\pi}_{t+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^{\pi}_{t}(s')\}$$

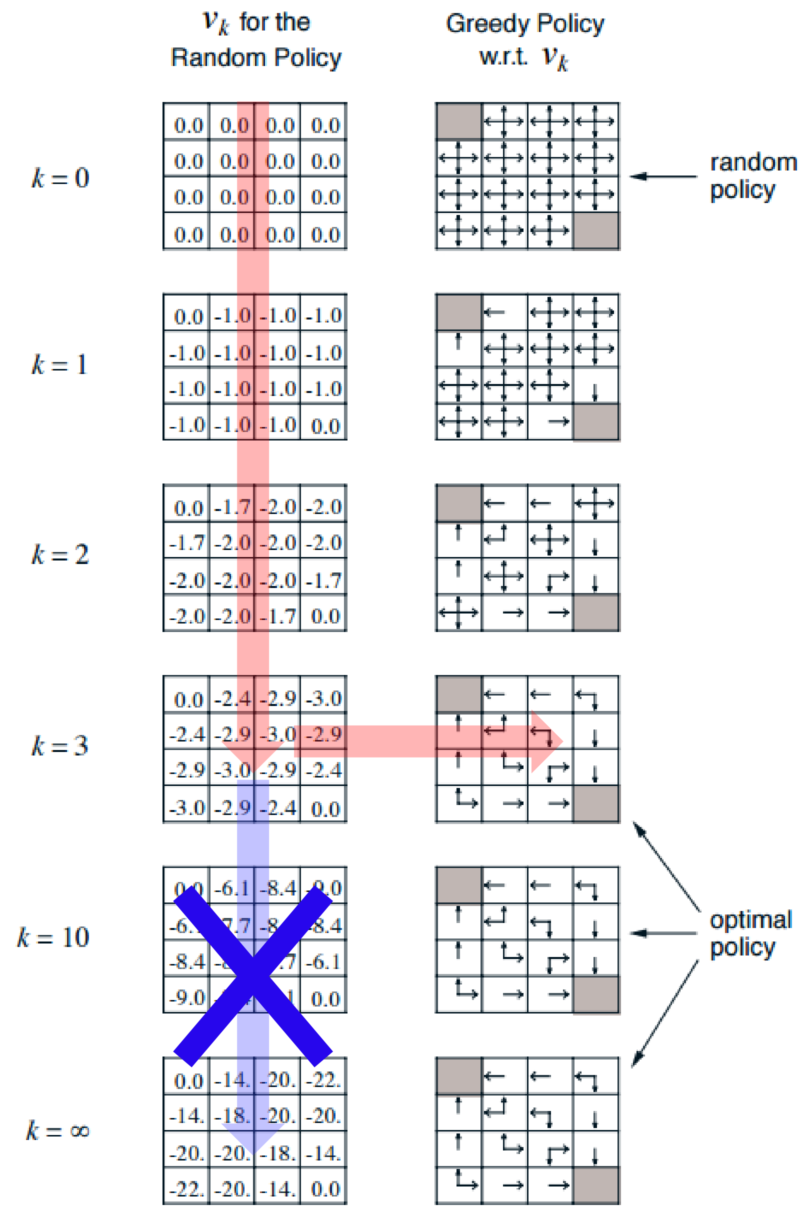
Converged state value function $V^{\pi}(s)$

For each state s:

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}}\, Q^{\pi}(s, a)$$
$$= \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')]$$

Issues :
Policy evaluation requires iterative computation, requiring multiple sweeps through the state set → slow to converge

Policy Iteration



Value Iteration

Solution:

- Stop policy evaluation after just one sweep (one backup of each state)
- Combine one sweep of policy evaluation and one sweep of policy improvement

For each state s:

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma V_t^{\pi}(s')\}$$

+

For each state s:

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{t+1}^{\pi}(s')]$$

↓

For each state s: **Value Iteration**

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V_t(s)\}$$

or, can be obtained simply by turning the Bellman optimality equation into an update rule :

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\}$$

Value Iteration:
A method to compute the optimum state-value function $V^*(s)$ by combining one sweep of policy evaluation and one sweep of policy improvement

Algorithm

Initialize $V(s) \leftarrow 0$ for all states $s \in S$

Repeat

For each state s:

$$V(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V(s)\}$$

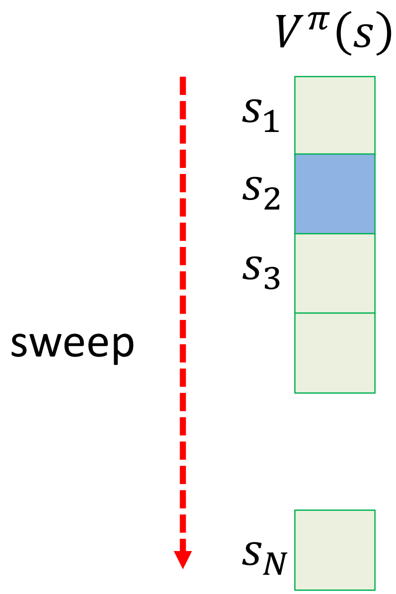
Until $\max_{s \in S} |V_t(s) - V_{t-1}\,(s)| \le e$

Optimum policy can be obtained from the converged $V^*(s)$:

$$\pi^*(\mathrm{s}) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s)\}$$

Asynchronous DP Algorithms

A major drawback to the DP methods is that they involve operations over the entire state set of the MDP

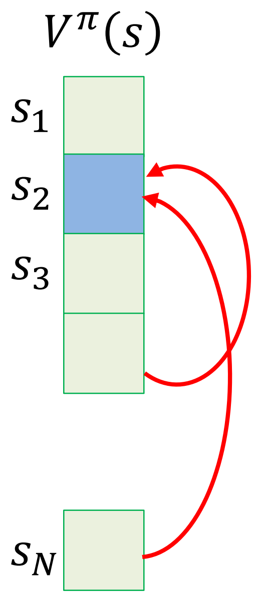


Conventional DP Algorithms

- Black mon game has 10^{20} states
- Go game has $3^{(19\times19)}$ states

…

- Take forever to sweep all states
- Does not improve policy until value functions are full backed up



Asynchronous DP Algorithms

- Back up the values of states in any order whatsoever, using whatever values of other states happen to be available
- Allow great flexibility in selecting states to which backup operations are applied
- Make it easier to intermix computation with real-time interaction: To solve a given MDP, we can run iterative DP algorithm at the same time that an agent is actually experiencing the MDP (Reinforcement Learning !!!!)

Value Iteration

Policy Evaluation

For $t = 1, \ldots$

For each state s:

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

Policy Iteration

Policy Improvement

For each state s:

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')]$$

Value Iteration

For each state s:

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V_t(s)\}$$

Asynchronous Value iteration

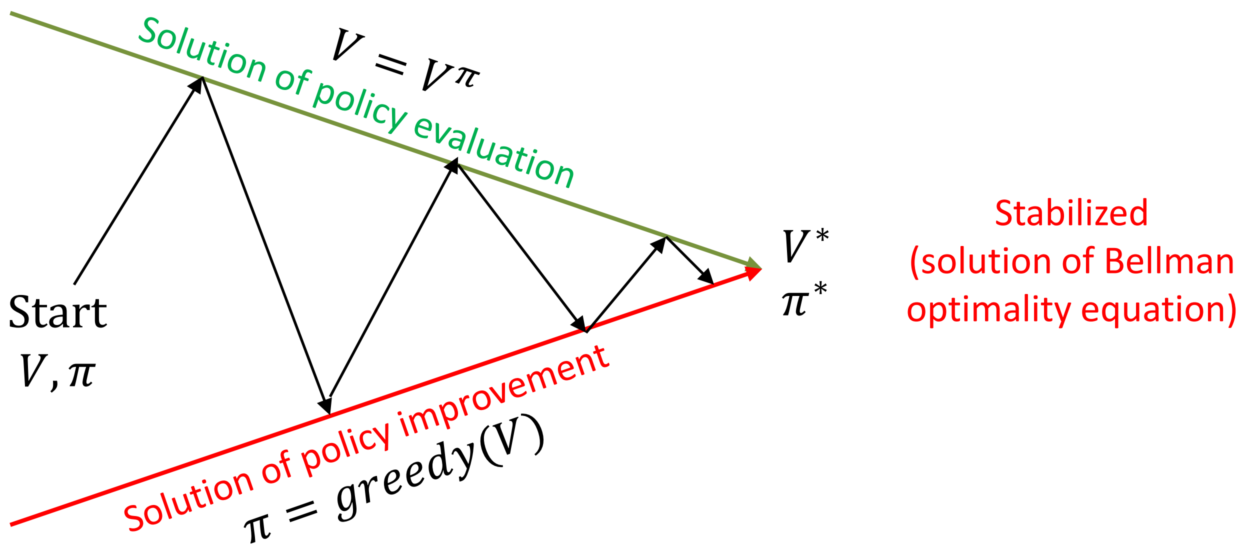
For any single state s:

$$V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V(s)\}$$

As long as both processes continue to update all states, the ultimate result is typically the same-convergence to the optimal value function and an optimal policy

Generalized Policy Iteration

Generalized Policy Iteration :
an general idea of interaction between policy evaluation and policy improvement processes



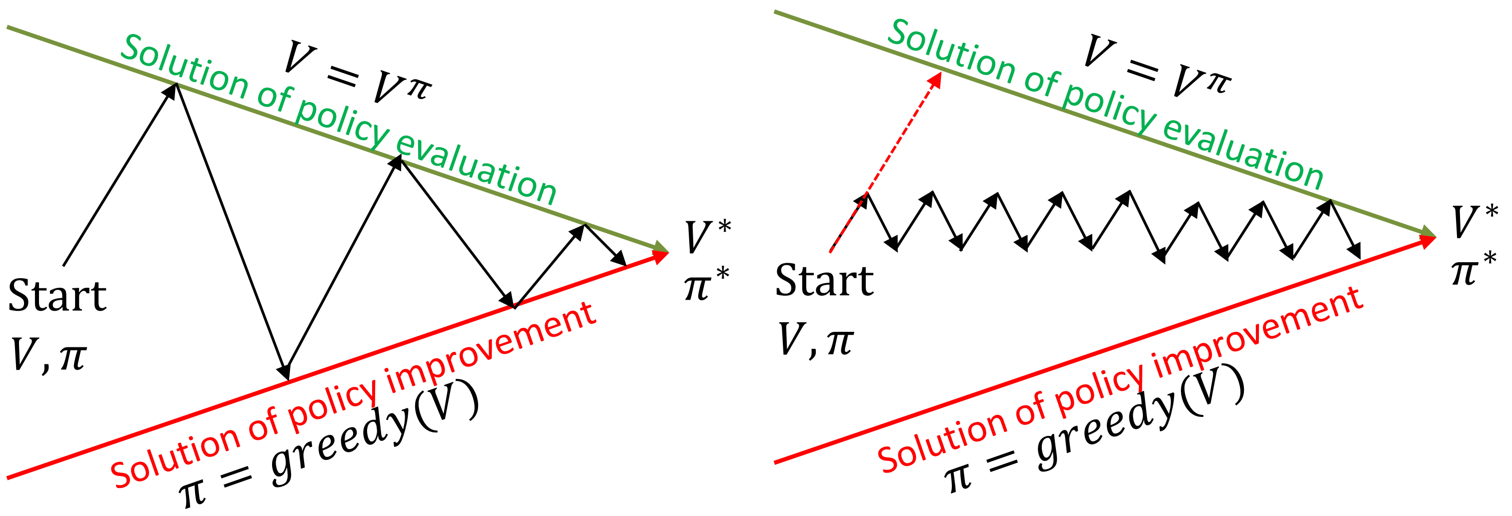
- Making policy greedy with respect to the value function typically makes the value function incorrect for the changed policy
- Making the value function consistent with the policy typically causes that policy no longer to be greedy
- In the long run, however, these two processes interact to find a single joint solution: the optimal value function and optimal policy

Generalized Policy Iteration

Generalized Policy Improvement (GPI)

Dynamic Programming
“full backup”

Asynchronous
Dynamic Programming



Asynchronous Dynamic Programming is a core concept in Reinforcement learning

Efficiency of Dynamic Programming

n : Number of states
m : Number of actions

- A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of deterministic policies is m^n

- Linear programming methods can also be used to solve MDPs, and in some case their worst-case convergence are better than those of DP methods. But linear programming methods become impractical at a much smaller number of states than do DP methods

- A DP method is guaranteed to find an optimal

- For a large problem, asynchronous DP is more efficient

- DP methods update estimates of the values of states based on estimates of the values of successor sates→ **update is based on other update (bootstrapping)**