Lecture 4: Value Iteration and Policy Iteration

Dr. Wen Fuxi

0. Review and two examples

1. Value iteration algorithm

2. Policy iteration algorithm

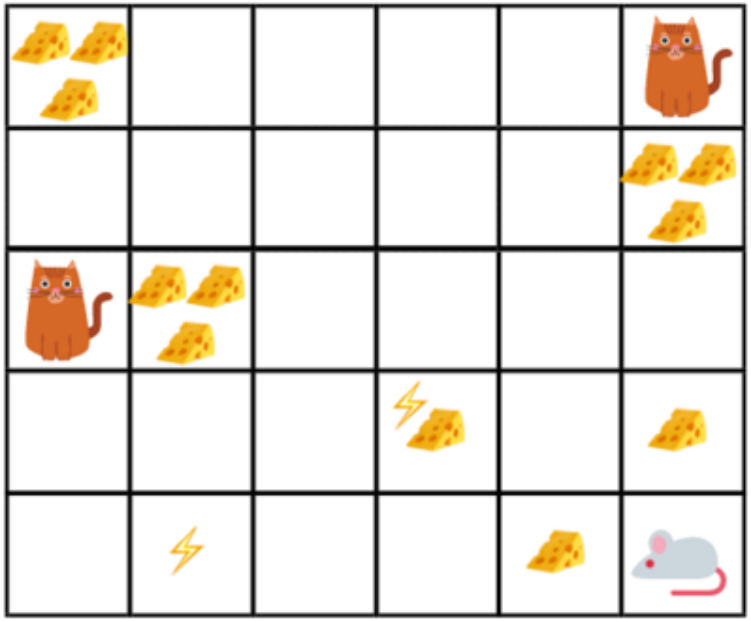
3. Truncated policy iteration algorithm

Lecture 1: Introduction and Basic Concepts

By using grid-world examples, we demonstrated the following key concepts:

- State
- Action
- State transition, state transition probability $p\left(s' \mid s, a\right)$
- Reward, reward probability $p(r \mid s, a)$
- Trajectory, episode, return, discounted return
- Markov decision process

Lecture 1: Introduction and Basic Concepts



- state space $\mathcal{S}$: positions in the maze
- action space $\mathcal{A}$: up, down, left, right
- immediate reward r: cheese $(+1)$, electricity shocks (-1), cats (-10000)
- policy $\pi(\cdot|s)$: the way to find cheese

Lecture 2: State Values and the Bellman Equation

Key concepts and results:

- State value: $v_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$
- Action value: $q_\pi(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$
- The Bellman equation (elementwise form):

$$v_\pi(s) = \sum_a \pi(a \mid s) \underbrace{\left[\sum_r p(r \mid s,a) r + \gamma \sum_{s'} p\left(s' \mid s,a\right) v_\pi\left(s'\right) \right]}_{q_\pi(s,a)}$$

$$= \sum_a \pi(a \mid s) q_\pi(s,a)$$

- The Bellman equation (matrix-vector form):

$$v_\pi = r_\pi + \gamma P_\pi v_\pi$$

- How to solve the Bellman equation: closed-form solution, iterative solution

Lecture 3: Optimal State Values and Bellman Optimality Equation

Bellman optimality equation:

- Elementwise form:

$$v(s) = \max_{\pi} \sum_{a} \pi(a \mid s) \underbrace{\left(\sum_{r} p(r \mid s, a) r + \gamma \sum_{s'} p\left(s' \mid s, a\right) v\left(s'\right) \right)}_{q(s,a)}, \quad \forall s \in \mathcal{S}$$

- Matrix-vector form:

$$v = \max_{\pi} \left(r_{\pi} + \gamma P_{\pi} v \right)$$

Lecture 3: Optimal State Values and Bellman Optimality Equation

Questions about the Bellman optimality equation:

- Does this equation have solutions? - Existence
  - *Yes, by the contraction mapping theorem*
- Is the solution to this equation unique? - Uniqueness
  - *Yes, by the contraction mapping theorem*
- How to solve this equation? - Algorithm
  - *Iterative algorithm suggested by the contraction mapping theorem*
- Why we study this equation - Optimality
  - *Because its solution corresponds to the optimal state value and optimal policy.*

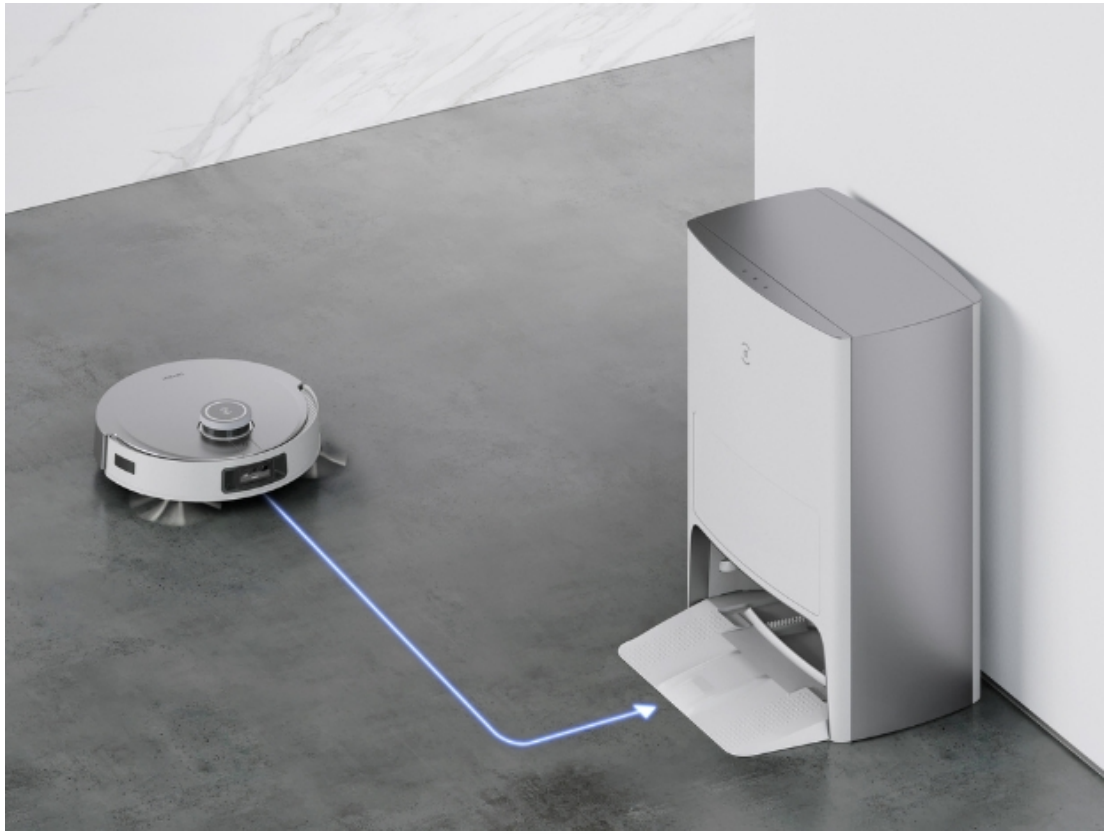
Finally, we understand why it is important to study the BOE!

Example 1: Indoor cleaning robot

Figure: Assist humans in floor cleaning

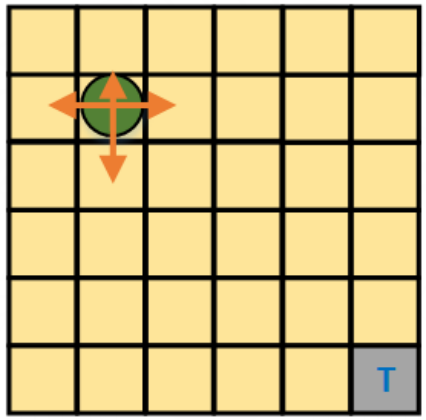
Example 1: Indoor cleaning robot

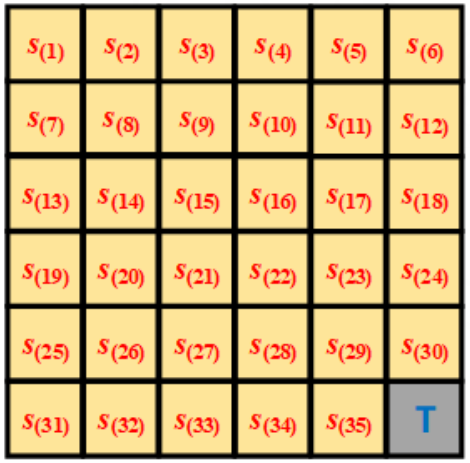
Our task is to help the auto vacuums return to their charging stations.



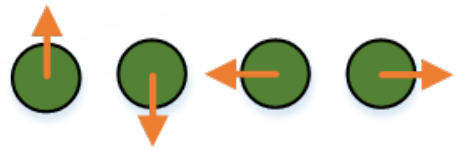
Example 1: Indoor cleaning robot

Grid environment, State space and Action space





State = $\{s_{(i)}\}$, i=1, 2,… , 35, 36



Action = {North, South, West, East}

Example 1: Indoor cleaning robot

Environment model

$$\Pr\left\{s' = \text{The cell below} \mid s, a = south\right\} = 0.8$$

$$\Pr\left\{s' = \text{Left Cell} \mid s, a = west\right\} = 0.1$$

$$\Pr\left\{s' = \text{Right Cell} \mid s, a = right\right\} = 0.1$$

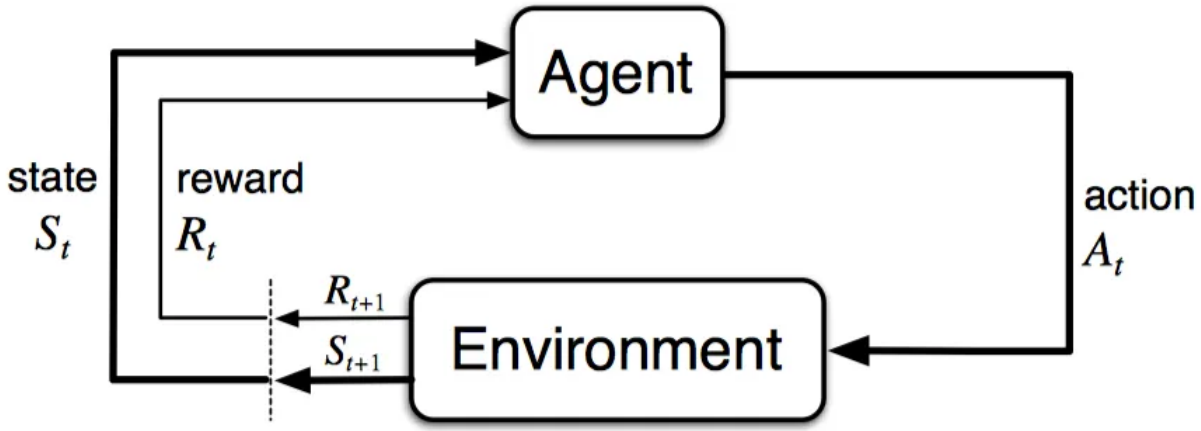
$$\Pr\left\{s' = \text{The cell above} \mid s, a = north\right\} = 0$$

Reward

$$r\left(s, a, s'\right) = \begin{cases} -1 & \text{if } s' \neq \mathrm{T} \\ +9 & \text{if } s' = \mathrm{T} \end{cases}$$

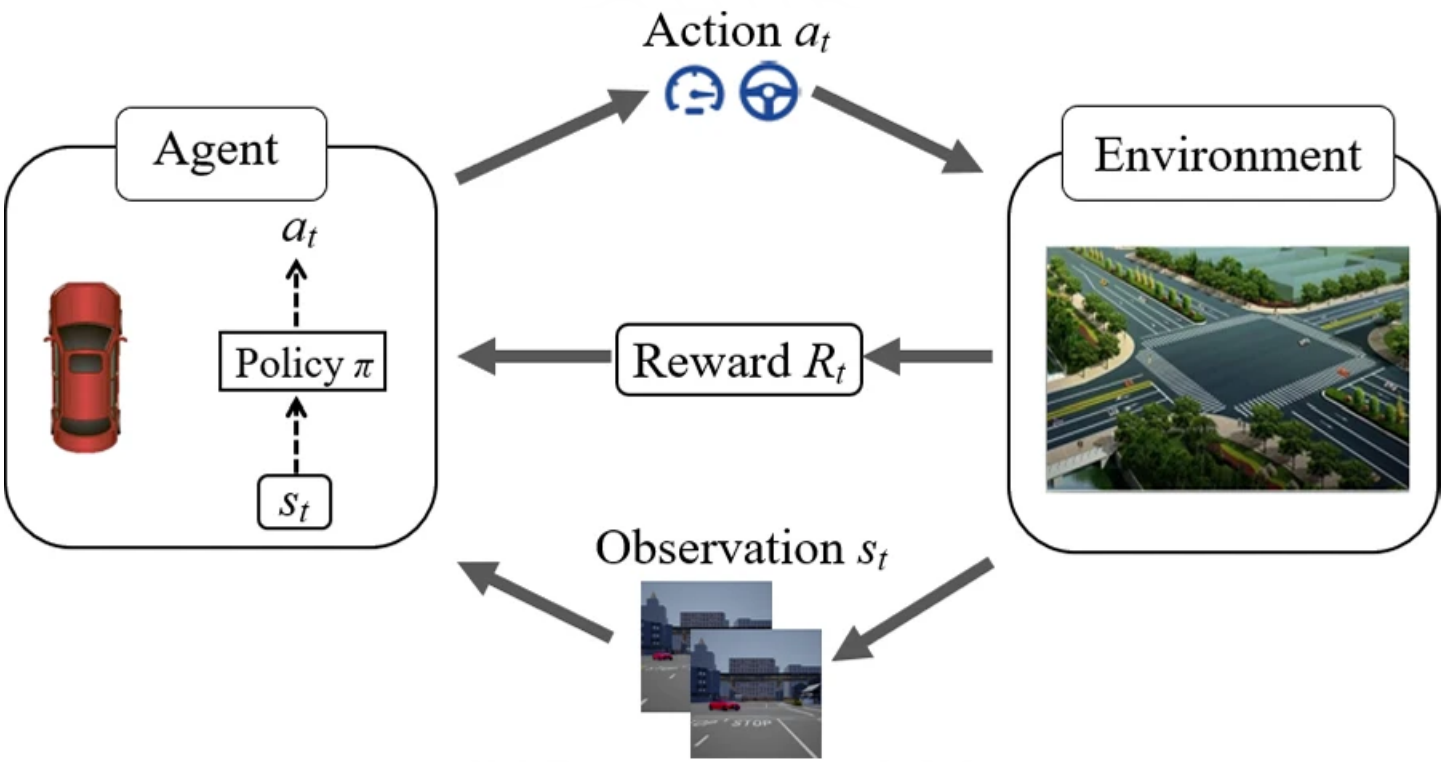
Example 2: Autonomous Driving System

Agent-Environment Feedback Loop for RL

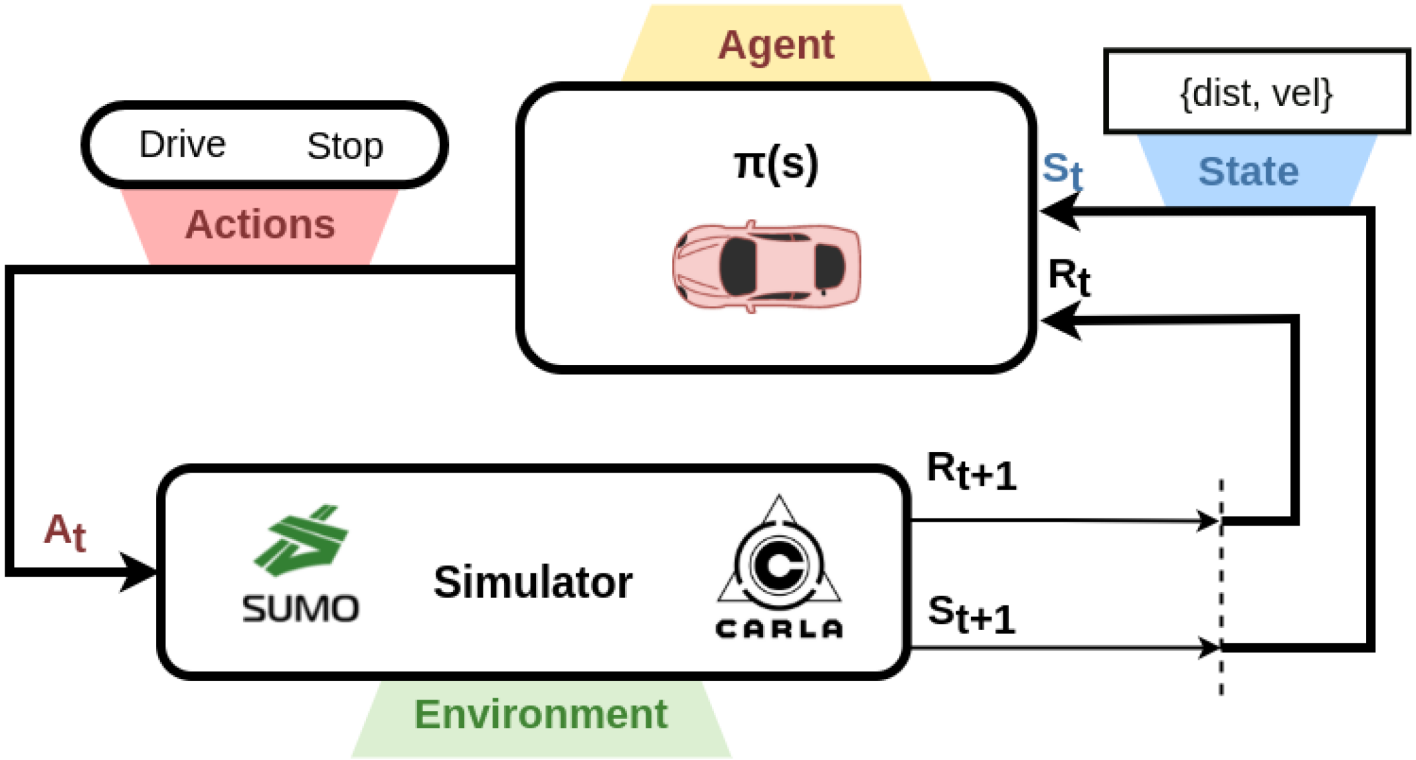


Example 2: Autonomous Driving System

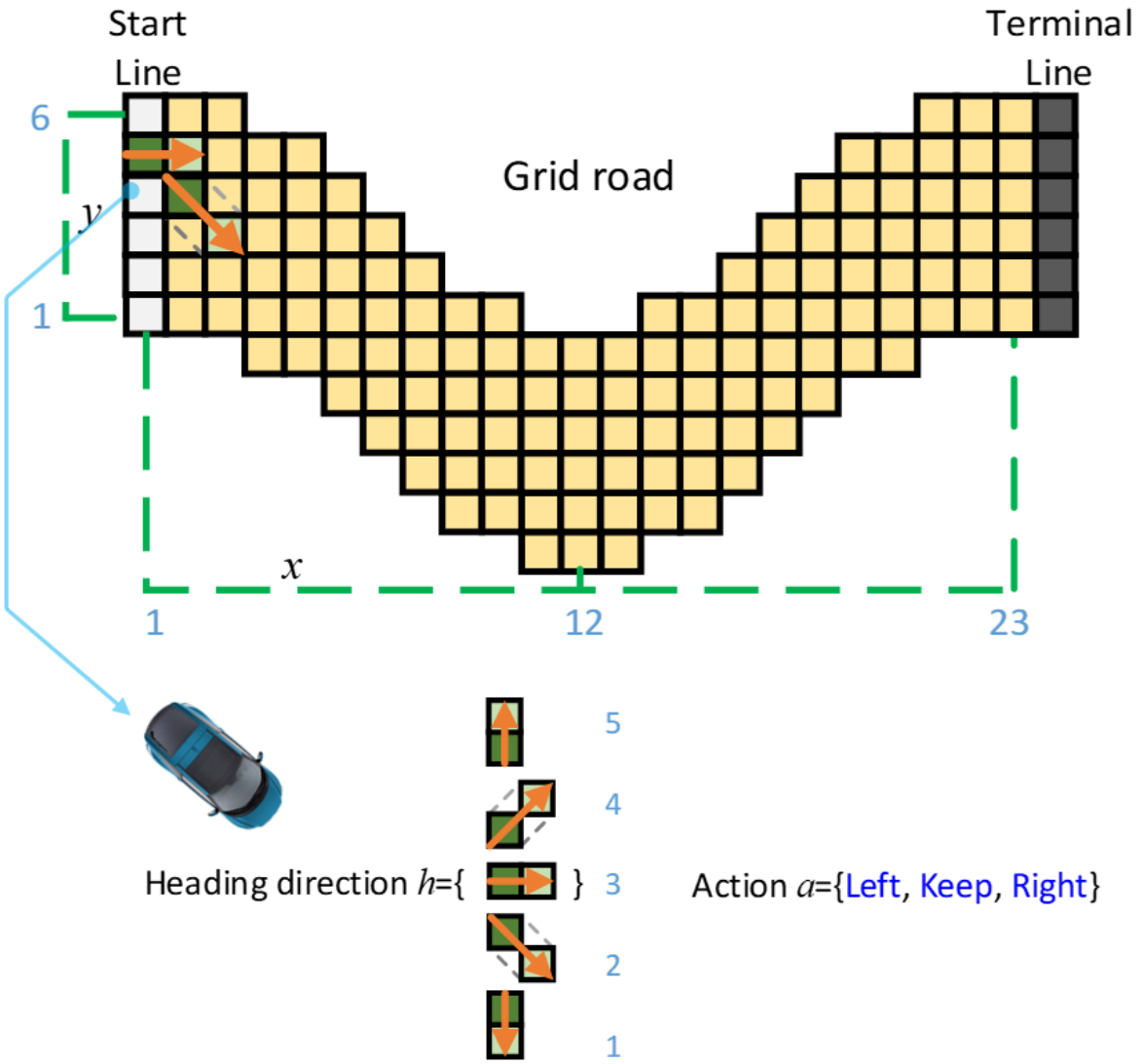
RL model for autonomous driving at intersections



Example 2: Autonomous Driving System



Example 2: An autonomous car works on a curved grid road



Example 2: An autonomous car works on a curved grid road

- Each car takes up two adjacent cells. The rear cell corresponds to two state dimensions, i.e., horizontal position (x) and vertical position (y). The relative position between the two cells represents the third state dimension, i.e., heading direction h.
- The car has three steering actions, i.e., "Right", "Keep", and "Left". Note that "Left" and "Right" are defined with respect to the heading direction (h). The steering action allows the car to adjust its heading direction. For example, if the car is heading toward the right, a "Left" steering action will turn the car toward the top right side.
- After the steering action is executed, the car will move one step forward along the new heading direction h_{new}.
- The terminal line is on the right side of the grid road, shaded as the self-driving destination. Once the front cell of the car occupies any cell in the terminal line, the current episode ends.

Example 2: An autonomous car works on a curved grid road

The state of this car-road system is a vector with three dimensions:

$$s = [x, y, h]^{\mathrm{T}} \in \mathcal{S},$$
$$\mathcal{S} = \mathcal{S}_x \times \mathcal{S}_y \times \mathcal{S}_h,$$
$$\mathcal{S}_x = \left\{x_{(1)}, x_{(2)}, \cdots, x_{(23)}\right\},$$
$$\mathcal{S}_y = \left\{y_{(1)}, y_{(2)}, \cdots, y_{(6)}\right\},$$
$$\mathcal{S}_h = \left\{h_{(1)}, h_{(2)}, \cdots, h_{(5)}\right\}.$$

The action space is

$$\mathcal{A} = \{ \text{ Left } , \text{ Keep, Right } \}.$$

Each action can deterministically turn the car to a neighboring direction and move the car one step forward along the new direction. Taking the action "Right" as an example, the car dynamics is represented at state $\left[x_{(1)}, y_{(5)}, h_{(3)}\right]^{\mathrm{T}}$:

$$\Pr\left\{s' = \left[x_{(2)}, y_{(4)}, h_{(2)}\right]^{\mathrm{T}} \mid s = \left[x_{(1)}, y_{(5)}, h_{(3)}\right]^{\mathrm{T}}, a = \text{ Right } \right\} = 1.$$

Example 2: An autonomous car works on a curved grid road

The autonomous car seeks to reach the destination as soon as possible while utilizing the least amount of steering energy.

The reward signal has two sub-elements: steering reward and moving reward.

Each steering action "Left" or "Right" results in a reward of -1 , and such a no-steering action "Keep" has a zero reward. The moving reward is either -1 or $-\sqrt{2}$, depending on whether the travel path is horizontal (vertical) or diagonal.

$$r\left(s, a, s'\right) = r_{\text{Steer}} + r_{\text{Move}} ,$$

$$r_{\text{Steer}} = \begin{cases} -1, \text{ if } a = \text{ Left} \\ 0, \text{ if } a = \text{ Keep} , \\ -1, \text{ if } a = \text{ Right} \end{cases}$$

$$r_{\text{Move}} = \begin{cases} -1, & \text{if } h' = 1 \text{ or } 3 \text{ or } 5 \\ -\sqrt{2} & , \text{ if } h' = 2 \text{ or } 4 \end{cases} .$$

The reward r_{Steer} poses a penalty to energy-wasting behaviors in the steering actuator, while r_{Move} is equivalent to penalizing the traveling distance.

Value iteration algorithm

Value Iteration

How to solve the Bellman optimality Equation (BOE)?

$$v = f(v) = \max_{\pi} (r_\pi + \gamma P_\pi v)$$

- The **contraction mapping theorem** suggests an iterative algorithm:

$$v_{k+1} = f(v_k) = \max_{\pi} (r_\pi + \gamma P_\pi v_k), \quad k = 1, 2, 3 \ldots$$

  where v_0 can be arbitrary.

- This algorithm can eventually find the optimal state value and an optimal policy.

How to implement this algorithm?

Value iteration algorithm

The algorithm (matrix-vector form)

$$v_{k+1} = f(v_k) = \max_{\pi} (r_\pi + \gamma P_\pi v_k), \quad k = 1, 2, 3 \ldots$$

It can be decomposed into two steps.

- Step 1: policy update. This step is to solve

$$\pi_{k+1} = \arg\max_{\pi} (r_\pi + \gamma P_\pi v_k)$$

  where v_k is given.

- Step 2: value update.

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$$

Question: Is v_k a state value?

No, because it is not ensured that v_k satisfies a Bellman equation.

Value iteration algorithm

Next, we need to study the elementwise form to implement the algorithm.

- Matrix-vector form is useful for theoretical analysis.
- Elementwise form is useful for implementation.

Value iteration algorithm - Elementwise form

▷ Step 1: Policy update

The elementwise form of

$$\pi_{k+1} = \arg\max_{\pi} \left(r_\pi + \gamma P_\pi v_k\right)$$

is

$$\pi_{k+1}(s) = \arg\max_{\pi} \sum_a \pi(a \mid s) \underbrace{\left(\sum_r p(r \mid s,a) r + \gamma \sum_{s'} p\left(s' \mid s,a\right) v_k\left(s'\right)\right)}_{q_k(s,a)}, \quad s \in \mathcal{S}$$

The optimal policy solving the above optimization problem is

$$\pi_{k+1}(a \mid s) = \begin{cases} 1 & a = a_k^*(s) \\ 0 & a \neq a_k^*(s) \end{cases}$$

where $a_k^*(s) = \arg\max_a q_k(a,s)$.

π_{k+1} is called a greedy policy, since it simply selects the greatest q-value.

Value iteration algorithm - Elementwise form

▷ Step 2: Value update

The elementwise form of

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$$

is

$$v_{k+1}(s) = \sum_a \pi_{k+1}(a \mid s) \underbrace{\left(\sum_r p(r \mid s, a) r + \gamma \sum_{s'} p\left(s' \mid s, a\right) v_k\left(s'\right) \right)}_{q_k(s,a)}, \quad s \in \mathcal{S}$$

Since π_{k+1} is greedy, the above equation is simply

$$v_{k+1}(s) = \max_a q_k(a, s)$$

Value iteration algorithm - *Pseudocode*

▷ Procedure summary:

$v_k(s) \rightarrow q_k(s,a) \rightarrow$ greedy policy $\pi_{k+1}(a \mid s) \rightarrow$ new value $v_{k+1} = \max_a q_k(s,a)$

Initialization: The probability model $p(r \mid s,a)$ and $p(s' \mid s,a)$ for all (s,a) are known. Initial guess v_0.

Aim: Search the optimal state value and an optimal policy solving the BOE.

While v_k has not converged in the sense that $\|v_k - v_{k-1}\|$ is greater than a predefined small threshold, for the k th iteration, do

For every state $s \in \mathcal{S}$, do

For every action $a \in \mathcal{A}(s)$, do

q-value: $q_k(s,a) = \sum_r p(r \mid s,a) r + \gamma \sum_{s'} p(s' \mid s,a) v_k(s')$

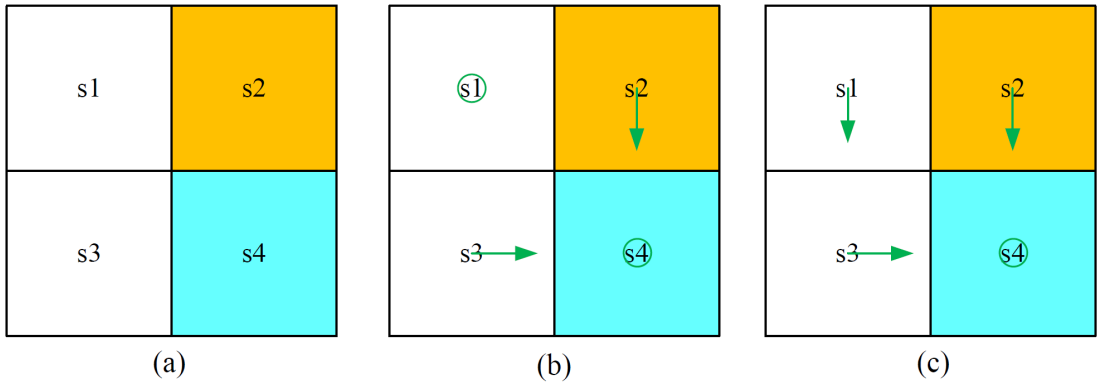
Maximum action value: $a_k^*(s) = \arg\max_a q_k(a,s)$

Policy update: $\pi_{k+1}(a \mid s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a \mid s) = 0$ otherwise

Value update: $v_{k+1}(s) = \max_a q_k(a,s)$

Value iteration algorithm - Example

▷ The reward setting is $r_{\text{boundary}} = r_{\text{forbidden}} = -1, r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.



q-table: The expression of $q(s, a)$.

q-value	a_1	a_2	a_3	a_4	a_5
s_1	$-1+\gamma v(s_1)$	$-1+\gamma v(s_2)$	$0+\gamma v(s_3)$	$-1+\gamma v(s_1)$	$0+\gamma v(s_1)$
s_2	$-1+\gamma v(s_2)$	$-1+\gamma v(s_2)$	$1+\gamma v(s_4)$	$0+\gamma v(s_1)$	$-1+\gamma v(s_2)$
s_3	$0+\gamma v(s_1)$	$1+\gamma v(s_4)$	$-1+\gamma v(s_3)$	$-1+\gamma v(s_3)$	$0+\gamma v(s_3)$
s_4	$-1+\gamma v(s_2)$	$-1+\gamma v(s_4)$	$-1+\gamma v(s_4)$	$0+\gamma v(s_3)$	$1+\gamma v(s_4)$

Value iteration algorithm - Example

- $k = 0$: let $v_0(s_1) = v_0(s_2) = v_0(s_3) = v_0(s_4) = 0$

q-value	a_1	a_2	a_3	a_4	a_5
s_1	-1	-1	0	-1	0
s_2	-1	-1	1	0	-1
s_3	0	1	-1	-1	0
s_4	-1	-1	-1	0	1

Step 1: Policy update:

$$\pi_1(a_5 \mid s_1) = 1, \pi_1(a_3 \mid s_2) = 1, \pi_1(a_2 \mid s_3) = 1, \pi_1(a_5 \mid s_4) = 1$$

Step 2: Value update:

$$v_1(s_1) = 0, v_1(s_2) = 1, v_1(s_3) = 1, v_1(s_4) = 1$$

Hints: $v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$

This policy is visualized in Figure (b).

Value iteration algorithm - Example

- $k = 1$: since $v_1(s_1) = 0, v_1(s_2) = 1, v_1(s_3) = 1, v_1(s_4) = 1$, we have

q-table	a_1	a_2	a_3	a_4	a_5
s_1	$-1+\gamma 0$	$-1+\gamma 1$	$0+\gamma 1$	$-1+\gamma 0$	$0+\gamma 0$
s_2	$-1+\gamma 1$	$-1+\gamma 1$	$1+\gamma 1$	$0+\gamma 0$	$-1+\gamma 1$
s_3	$0+\gamma 0$	$1+\gamma 1$	$-1+\gamma 1$	$-1+\gamma 1$	$0+\gamma 1$
s_4	$-1+\gamma 1$	$-1+\gamma 1$	$-1+\gamma 1$	$0+\gamma 1$	$1+\gamma 1$

Step 1: Policy update:

$$\pi_2(a_3 \mid s_1) = 1, \pi_2(a_3 \mid s_2) = 1, \pi_2(a_2 \mid s_3) = 1, \pi_2(a_5 \mid s_4) = 1$$

Step 2: Value update:

$$v_2(s_1) = \gamma 1, v_2(s_2) = 1+\gamma 1, v_2(s_3) = 1+\gamma 1, v_2(s_4) = 1+\gamma 1$$

This policy is visualized in Figure (c). The policy is already **optimal**!!

- $k = 2, 3, \ldots$. Stop when $\|v_k - v_{k+1}\|$ is smaller than a predefined threshold θ.

Policy iteration algorithm

▷ Algorithm description: *Given a random initial policy* π_0,

- Step 1: policy evaluation (PE)

This step is to calculate the state value of π_k :

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k} \text{ (Bellman Equation)}$$

Note that v_{π_k} is a state value function.

- Step 2: policy improvement (PI)

$$\pi_{k+1} = \arg\max_{\pi} (r_\pi + \gamma P_\pi v_{\pi_k})$$

The maximization is componentwise!

Similar to the value iteration algorithm?

Policy iteration algorithm

- The algorithm leads to a sequence

$$\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \dots$$

$\mathrm{PE} =$ *policy evaluation*, $\mathrm{PI} =$ *policy improvement*

Questions

Q1: In the policy evaluation step, how to get the state value v_{π_k} by solving the Bellman equation?

Q2: In the policy improvement step, why is the new policy π_{k+1} better than π_k ?

Q3: Why can such an iterative algorithm finally reach an optimal policy?

Q4: What is the relationship between this policy iteration algorithm and the previous value iteration algorithm?

Policy iteration algorithm

▷ Q1: In the policy evaluation step, how to get the state value v_{π_k} by solving the Bellman equation?

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$$

- 1. Closed-form solution:

$$v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$$

- 2. Iterative solution:

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \ldots$$

Already studied in the lecture about Bellman equation.

▷ Policy iteration is an iterative algorithm with another iterative algorithm embedded in the policy evaluation step!

Policy iteration algorithm

▷ Q2: In the policy improvement step, why is the new policy π_{k+1} better than π_k?

Lemma (Policy Improvement)

If $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$, then $v_{\pi_{k+1}} \geq v_{\pi_k}$ for any k.

Policy iteration algorithm

▷ Q3: Why can such an iterative algorithm finally reach an optimal policy?

Since every iteration would improve the policy, we know

$$v_{\pi_0} \leq v_{\pi_1} \leq v_{\pi_2} \leq \cdots \leq v_{\pi_k} \leq \cdots \leq v^*$$

As a result, v_{π_k} keeps increasing and will converge.

Still need to prove what value it converges to.

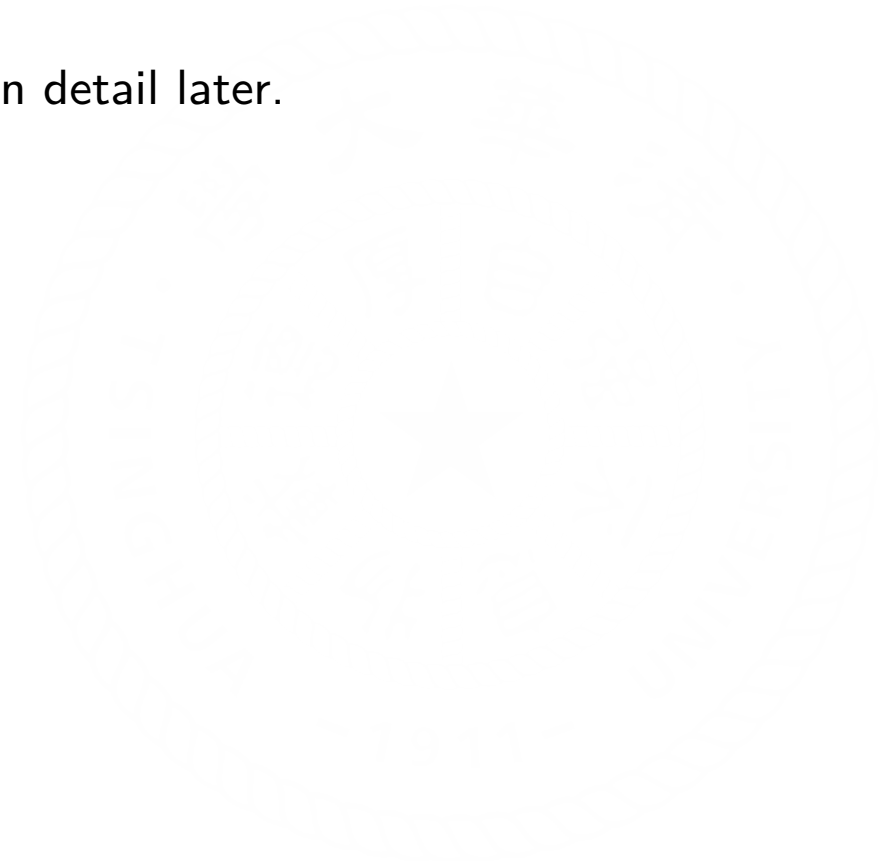
Theorem (Convergence of Policy Iteration)

The state value sequence $\{v_{\pi_k}\}_{k=0}^{\infty}$ generated by the policy iteration algorithm converges to the optimal state value v^*. As a result, the policy sequence $\{\pi_k\}_{k=0}^{\infty}$ converges to an optimal policy.

Policy iteration algorithm

- ▷ Q4: What is the relationship between policy iteration and value iteration?

Will be explained in detail later.



Policy iteration algorithm - Elementwise form

Step 1: Policy evaluation

▷ Matrix-vector form:

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \ldots$$

▷ Elementwise form:

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a \mid s) \left(\sum_r p(r \mid s, a) r + \gamma \sum_{s'} p\left(s' \mid s, a\right) v_{\pi_k}^{(j)}\left(s'\right) \right), \quad s \in \mathcal{S}$$

Stop when j is sufficiently large or $\left\| v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)} \right\|$ is sufficiently small.

Policy iteration algorithm - Elementwise form

Step 2: Policy improvement

Matrix-vector form: $\pi_{k+1} = \arg\max_{\pi} \left(r_\pi + \gamma P_\pi v_{\pi_k}\right)$

▷ Elementwise form

$$\pi_{k+1}(s) = \arg\max_{\pi} \sum_a \pi(a \mid s) \underbrace{\left(\sum_r p(r \mid s,a) r + \gamma \sum_{s'} p\left(s' \mid s,a\right) v_{\pi_k}\left(s'\right) \right)}_{q_{\pi_k}(s,a)}, \quad s \in \mathcal{S}.$$

Here, $q_{\pi_k}(s,a)$ is the action value under policy π_k. Let

$$a_k^*(s) = \arg\max_a q_{\pi_k}(a,s)$$

Then, the greedy policy is

$$\pi_{k+1}(a \mid s) = \begin{cases} 1 & a = a_k^*(s) \\ 0 & a \neq a_k^*(s) \end{cases}$$

Policy iteration algorithm - Implementation

Pseudocode: Policy iteration algorithm

Initialization: The probability model $p(r \mid s, a)$ and $p(s' \mid s, a)$ for all (s, a) are known. Initial guess π_0.

Aim: Search for the optimal state value and the corresponding optimal policy.

While v_{π_k} has not converged, for the k th iteration, do

Policy evaluation:

Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$

While $v_{\pi_k}^{(j)}$ has not converged, for the jth iteration, do

For every state $s \in \mathcal{S}$, do

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}^{(j)}(s') \right]$$

Policy improvement:

For every state $s \in \mathcal{S}$, do

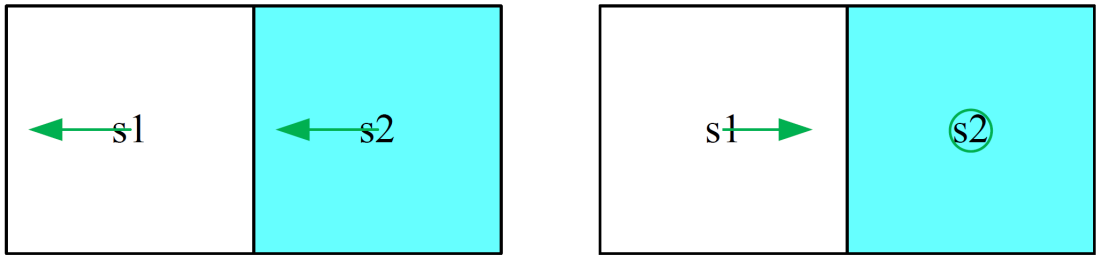
For every action $a \in \mathcal{A}$, do

$$q_{\pi_k}(s,a) = \sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s')$$

$a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$

$\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

Policy iteration algorithm - Simple example



(a) initial policy (b) optimal policy

- The reward setting is $r_{\text{boundary}} = -1$ and $r_{\text{target}} = 1$.
- The discount rate is $\gamma = 0.9$.

Actions: a_ℓ, a_0, a_r represent go left, stay unchanged, and go right.

Aim: use policy iteration to determine the optimal policy.

Policy iteration algorithm

▷ Iteration $k = 0$: Step 1: policy evaluation

π_0 is selected as the policy in Figure (a). The Bellman equation is

$$v_{\pi_0}(s_1) = -1 + \gamma v_{\pi_0}(s_1),$$
$$v_{\pi_0}(s_2) = 0 + \gamma v_{\pi_0}(s_1)$$

- **Direct method**:

$$v_{\pi_0}(s_1) = -10, \quad v_{\pi_0}(s_2) = -9$$

- **Iterative method**. Select the initial guess as $v_{\pi_0}^{(0)}(s_1) = v_{\pi_0}^{(0)}(s_2) = 0$:

$$\begin{cases} v_{\pi_0}^{(1)}(s_1) = -1 + \gamma v_{\pi_0}^{(0)}(s_1) = -1, \\ v_{\pi_0}^{(1)}(s_2) = 0 + \gamma v_{\pi_0}^{(0)}(s_1) = 0, \end{cases}$$

$$\begin{cases} v_{\pi_0}^{(2)}(s_1) = -1 + \gamma v_{\pi_0}^{(1)}(s_1) = -1.9, \\ v_{\pi_0}^{(2)}(s_2) = 0 + \gamma v_{\pi_0}^{(1)}(s_1) = -0.9, \end{cases}$$

$$\begin{cases} v_{\pi_0(s_1)}^{(3)}(s_1) - \gamma v_{\pi_0}^{(2)}(s_1) = -2.71, \\ v_{\pi_0}^{(3)}(s_2) = 0 + \gamma v_{\pi_0}^{(2)}(s_1) = -1.71, \end{cases}$$

Policy iteration algorithm

▷ Iteration $k = 0$: Step 2: policy improvement

The expression of $q_{\pi_k}(s, a)$:

$q_{\pi_k}(s, a)$	a_ℓ	a_0	a_r
s_1	$-1 + \gamma v_{\pi_k}(s_1)$	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$
s_2	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$	$-1 + \gamma v_{\pi_k}(s_2)$

Substituting $v_{\pi_0}(s_1) = -10, v_{\pi_0}(s_2) = -9$ and $\gamma = 0.9$ gives

$q_{\pi_0}(s, a)$	a_ℓ	a_0	a_r
s_1	-10	-9	-7.1
s_2	-9	-7.1	-9.1

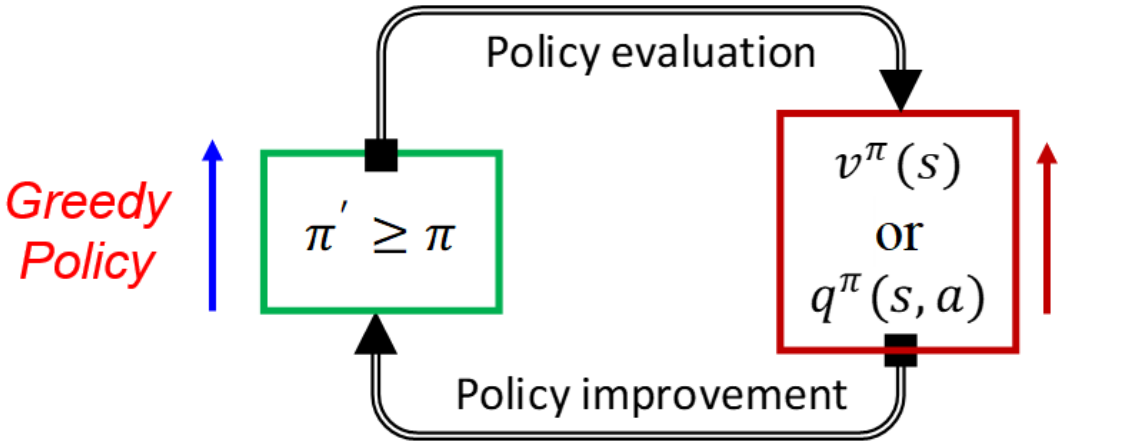
By seeking the greatest value of q_{π_0}, the improved policy is:

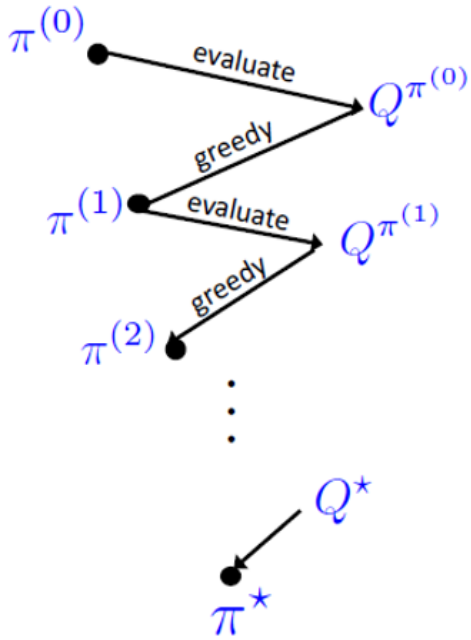
$$\pi_1(a_r \mid s_1) = 1, \quad \pi_1(a_0 \mid s_2) = 1$$

This policy is optimal after one iteration! The process should continue in your programming until the stopping criterion is satisfied.

Policy iteration algorithm

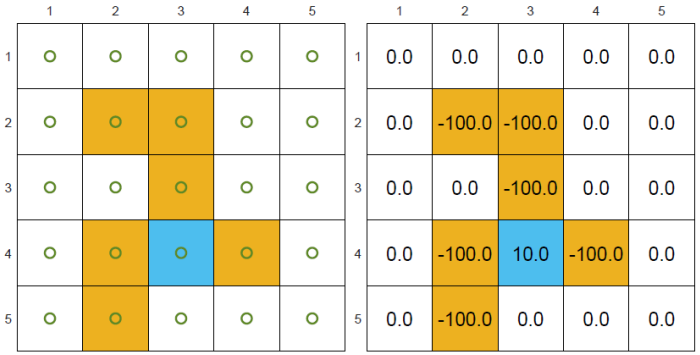
Excise! Set the left cell as the target area.



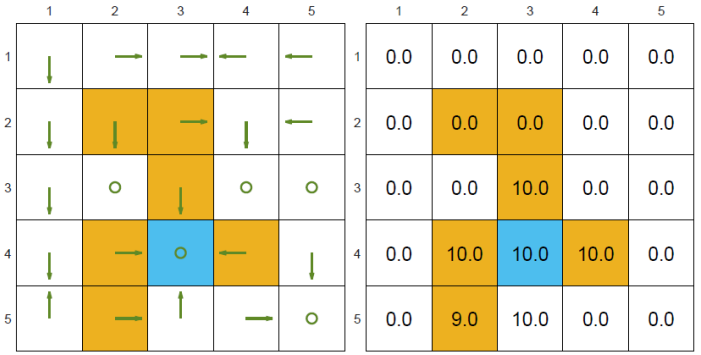


Policy iteration algorithm - Complicated example

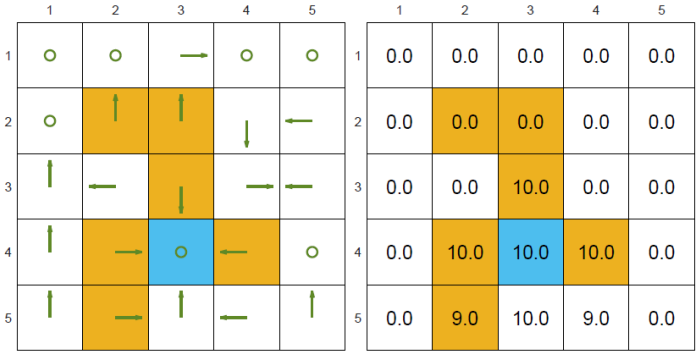
- Setting: $r_{\text{boundary}} = -1, r_{\text{forbidden}} = -10, r_{\text{target}} = 1, \gamma = 0.9$.
- Let's check out the intermediate policies and state values.



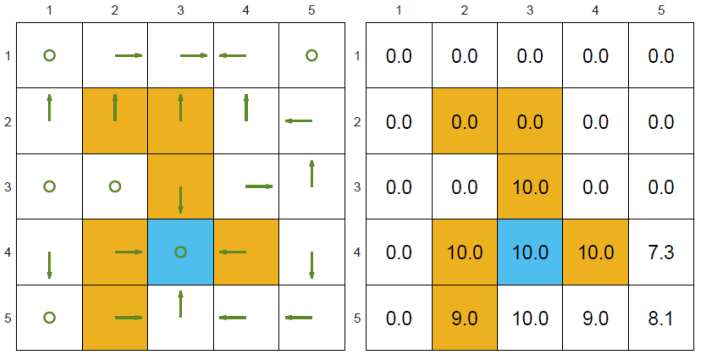
π_0 and v_{π_0}



π_1 and v_{π_1}



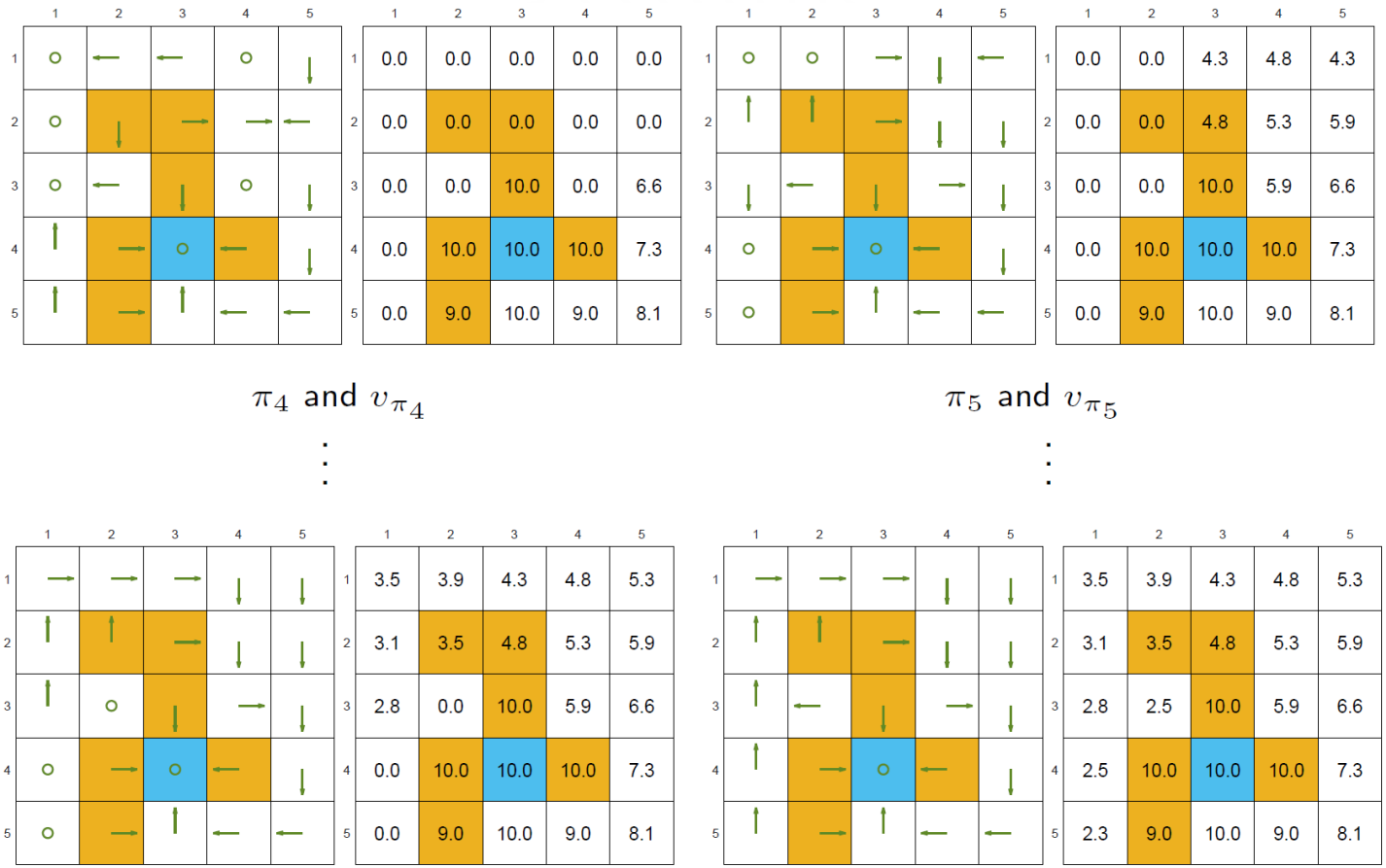
π_2 and v_{π_2}



π_3 and v_{π_3}

Policy iteration algorithm - complicated example

- Interesting pattern of the policies and state values



Compare value iteration and policy iteration

Policy Iteration: Start from π_0

- Policy Evaluation (PE):

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$$

- Policy Improvement (PI):

$$\pi_{k+1} = \arg\max_{\pi} \left(r_\pi + \gamma P_\pi v_{\pi_k} \right)$$

Value Iteration: Start from v_0

- Policy Update (PU):

$$\pi_{k+1} = \arg\max_{\pi} \left(r_\pi + \gamma P_\pi v_k \right)$$

- Value Update (VU):

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$$

Compare value iteration and policy iteration

The two algorithms are very similar

- Policy iteration:

$$\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \dots$$

- Value iteration:

$$v_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} v_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} v_2 \xrightarrow{PU} \dots$$

- $\mathrm{PE} =$ policy evaluation. $\mathrm{PI} =$ policy improvement.
- $\mathrm{PU} =$ policy update. $\mathrm{VU} =$ value update.

Compare value iteration and policy iteration

▷ Let's compare the steps carefully:

	Policy iteration algorithm	Value iteration algorithm	Comments
1) P	π_0	N/A	
2) V	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 \doteq v_{\pi_0}$	
3) P	$\pi_1 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_0}\right)$	$\pi_1 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_0\right)$	Same policies
4) V	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$	$v_{\pi_1} \geq v_1$, $v_{\pi_1} \geq v_{\pi_0}$
5) P	$\pi_2 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_1}\right)$	$\pi_2' = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_1\right)$	
⋮	⋮	⋮	⋮

- They start from the same initial condition.
- The first three steps are the same.
- The fourth step becomes different:

  In policy iteration, solving $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$ requires an iterative algorithm (an infinite number of iterations)

  In value iteration, $v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$ is a one-step iteration

Compare value iteration and policy iteration

Consider the step of solving $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$:

$$
\begin{aligned}
& & v_{\pi_1}^{(0)} &= v_0 \\
\text{value iteration} \leftarrow v_1 \longleftarrow & & v_{\pi_1}^{(1)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)} \\
& & v_{\pi_1}^{(2)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(1)} \\
& & &\vdots \\
\textit{truncated policy iteration} \leftarrow \bar{v}_1 \longleftarrow & & v_{\pi_1}^{(j)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)} \\
& & &\vdots \\
\text{policy iteration} \leftarrow v_{\pi_1} \longleftarrow & & v_{\pi_1}^{(\infty)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)}
\end{aligned}
$$

- The value iteration algorithm computes once.
- The policy iteration algorithm computes an infinite number of iterations.
- The truncated policy iteration algorithm computes a finite number of iterations (say j). The remaining iterations from j to ∞ are truncated.

Truncated policy iteration - Pseudocode

Pseudocode: Truncated policy iteration algorithm

Initialization: The probability model $p(r \mid s, a)$ and $p\left(s' \mid s, a\right)$ for all (s, a) are known. Initial guess π_0.

Aim: Search for the optimal state value and an optimal policy.

While v_k has not converged, for the k th iteration, do

Policy evaluation:

Initialization: select the initial guess as $v_k^{(0)} = v_{k-1}$. The maximum iteration is set to be j_{truncate} .

While $j < j_{\text{truncate}}$, do

For every state $s \in \mathcal{S}$, do

$$v_k^{(j+1)}(s) = \sum_a \pi_k(a \mid s) \left[\sum_r p(r \mid s, a) r + \gamma \sum_{s'} p\left(s' \mid s, a\right) v_k^{(j)}\left(s'\right) \right]$$

Set $v_k = v_k^{(j_{\text{truncate}})}$

Policy improvement:

For every state $s \in \mathcal{S}$, do

For every action $a \in \mathcal{A}(s)$, do

$$q_k(s, a) = \sum_r p(r \mid s, a) r + \gamma \sum_{s'} p\left(s' \mid s, a\right) v_k\left(s'\right)$$

$$a_k^*(s) = \arg\max_a q_k(s, a)$$

$$\pi_{k+1}(a \mid s) = 1 \text{ if } a = a_k^*, \text{ and } \pi_{k+1}(a \mid s) = 0 \text{ otherwise}$$

Truncated policy iteration - Convergence

▷ Will the truncation undermine convergence?

Proposition (Value Improvement)

Consider the iterative algorithm for solving the policy evaluation step:

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \ldots$$

If the initial guess is selected as $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$, it holds that

$$v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$$

for every $j = 0, 1, 2, \ldots$.

Truncated policy iteration - Convergence

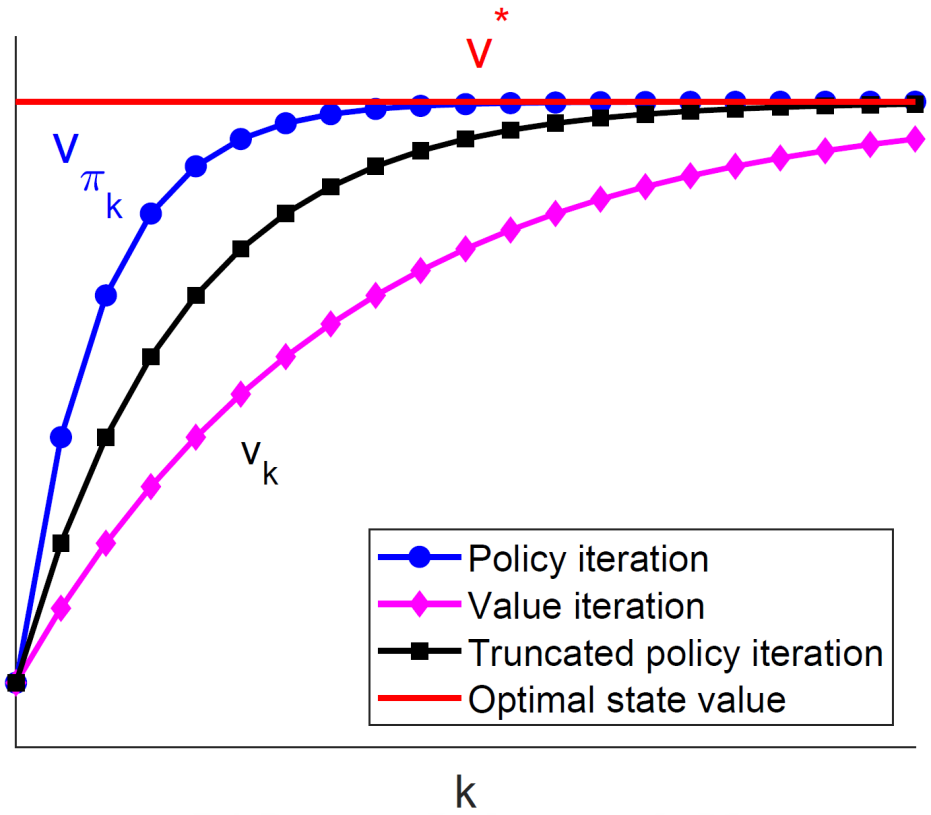


Figure: Illustration of the relationship among value iteration, policy iteration, and truncated policy iteration.

The convergence proof of PI is based on that of VI. Since VI converges, we know PI converges.

Summary

- Value Iteration: it is the iterative algorithm solving the Bellman optimality equation: given an initial value v_0,

$$v_{k+1} = \max_{\pi} (r_\pi + \gamma P_\pi v_k)$$

$$\begin{cases} \text{Policy update: } \pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_k) \\ \text{Value update: } v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k \end{cases}$$

- Policy Iteration: given an initial policy π_0,

$$\begin{cases} \text{Policy evaluation: } v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k} \\ \text{Policy improvement: } \pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k}) \end{cases}$$

- Truncated Policy Iteration