In the Name of God

Iran University of Science and Technology

Reinforcement Learning in Control

Dr. Saeed Shamaghdari

Electrical Engineering Department
Control Group

Fall 2025 | 4041

Dynamic Programming

Dynamic Programming

Classic Dynamic Programming Methods:

Need to know the full environment model
Problem formulation in the form of a finite MDP
High computational cost

→ Used to compute the optimal policy

Application of Dynamic Programming in Reinforcement Learning:

Value function approximation
Computing the optimal policy (over time)

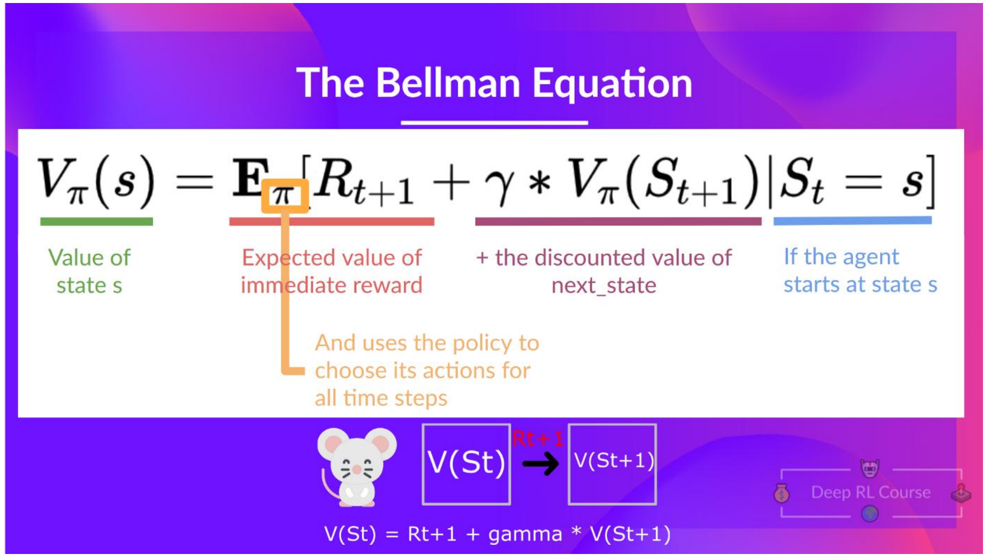
Discrete and Continuous Systems

Bellman Optimality Equation –State Value Function

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right].$$

How to determine V_*?
How to determine the optimal policy?

Reminder Box



Lecture 3

Bellman Optimality Equation –Action Value Function

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \,\Big|\, S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s',r \,|\, s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big].
\end{aligned}
$$

How to determine q_*?
How to determine the optimal policy?

Determining V_π and q_π using Dynamic Programming

Value function approximation using **iterative algorithms** based on the Bellman equation

Convergence proof to the true value function

Given a policy π: determine $V_\pi(s)$ using the Bellman equation

$\rightarrow$ **Policy Evaluation**

With an approximation of $V_\pi(s)$: determine an improved policy π'

$\rightarrow$ **Policy Improvement**

Objective: For a given policy π, compute V_π

Reminder Box: Bellman Equation

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

Bellman Equation:
Describes the relationship between the **value function at state s** and the **value function at state s'**

Metric Spaces and Banach Fixed Point

Definition (Metric Space)

A ***metric space*** is an ordered pair (X, d) consists of an *underlying set* X and a real-valued function $d(x, y)$, called *metric*, defined for $x, y \in X$ such that for any $x, y, z \in X$ the following conditions are satisfied:

1. $d(x, y) \geq 0$ [*non-negativity*]
2. $d(x, y) = 0 \Leftrightarrow x = y$ [*identity of indiscernibles*]
3. $d(x, y) = d(y, x)$ [*symmetry*]
4. $d(x, y) \leq d(x, z) + d(z, y)$ [*triangle inequality*]

Metric Spaces and Banach Fixed Point

Definition (Contraction)

Let (X, d) be a metric space and $f: X \to X$. We say that f is a *contraction*, or a *contraction mapping*, if there is a real number $k \in [0,1)$, such that

$$d(f(x), f(y)) \leq kd(x, y)$$

for all x and y in X, where the term k is called a *Lipschitz coefficent* for f.

Metric Spaces and Banach Fixed Point

Theorem (Contraction Mapping)

Let (X, d) be a complete metric space and let $f: X \to X$ be a contraction. Then there is one and only one fixed point x^* such that

$$f(x^*) = x^*.$$

Moreover, if x is any point in X and $fn(x)$ is inductively defined by

$$f_2(x) = f\big(f(x)\big), f_3(x) = f\big(f_2(x)\big), \dots, f_n(x) = f\big(f_n - 1(x)\big),$$

then $f_n(x) \to x^*$ as $n \to \infty$.

Banach Fixed-Point Perspective:

$\boldsymbol{V_\pi}$ is a fixed point of the Bellman equation: $V_\pi(s) = T(V_\pi(s'))$

Policy Evaluation: Method

Start with an initial guess for $\boldsymbol{V_\pi}$: arbitrary $\boldsymbol{v_0}$
Update the value function approximation using the following rule:

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

Iterative Policy Evaluation

$k \to \infty \quad \longrightarrow \quad V_k \to V_\pi$

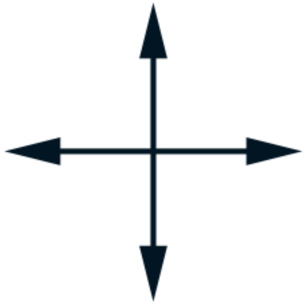
Algorithm

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

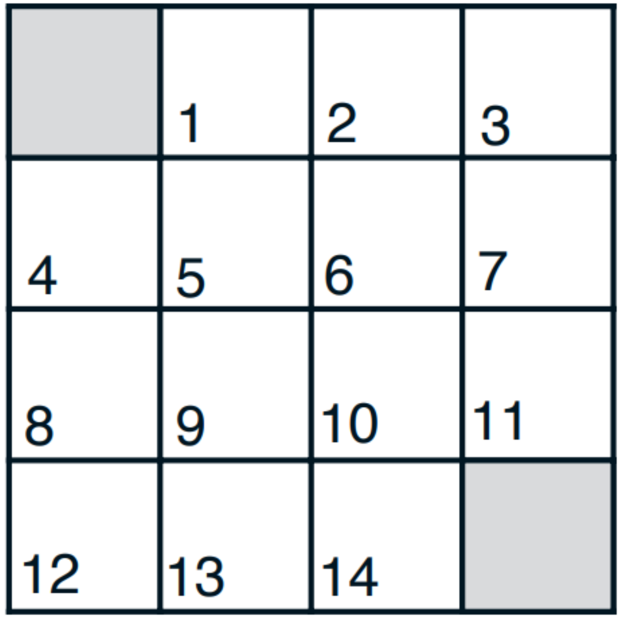
Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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 $\Delta < \theta$

Gridworld



actions



$$R_t = -1$$

on all transitions

Policy π: Random walk

Initial value: $\boldsymbol{V_0 = 0}$

Gridworld

$k = 0$

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

$$V_{k+1}(s) = \sum \frac{1}{4}(-1 + \gamma V_k(s'))$$

$k = 1$

0.0	-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

Gridworld

$k = 0$

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

$k = 1$

0.0	-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

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-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	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Improvement

Theorem

Let π and π' be any pair of deterministic policies such that, for all $s \in S$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

Then the policy π' must be as good as, or better than, π. That is, it must obtain greater or equal expected return from all states $s \in S$:

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

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$ Meaning??

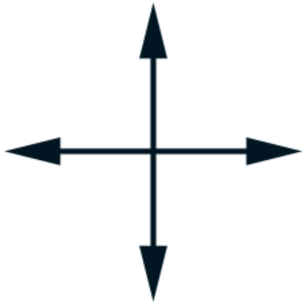
Policy Improvement

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

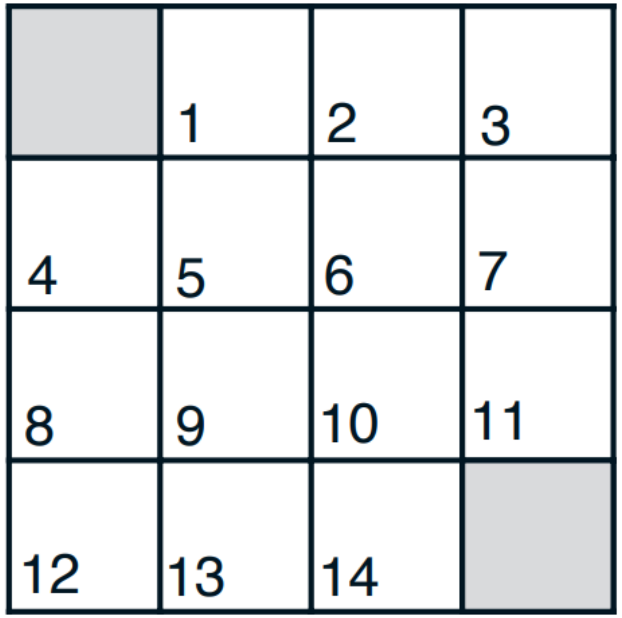
Selecting the Best Policy

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

Gridworld



actions



$$R_t = -1$$

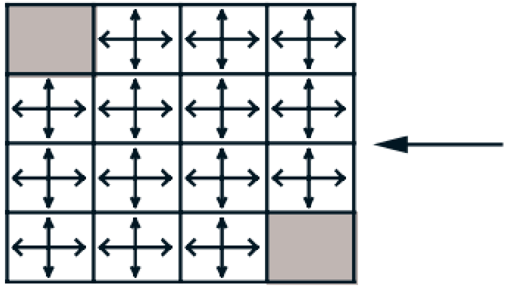
on all transitions

Policy π: Random walk
Initial value: $V_0 = 0$

Gridworld

$k = 0$

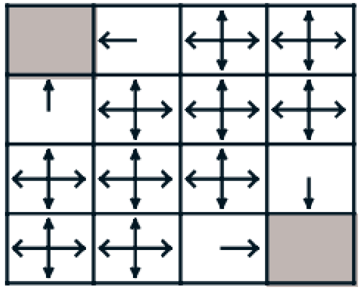
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

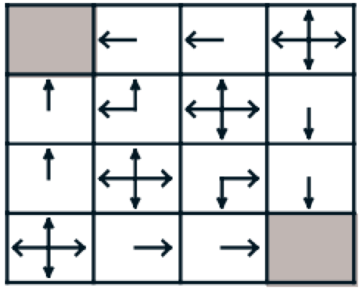
$k = 1$

0.0	-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



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Gridworld

$k = 3$

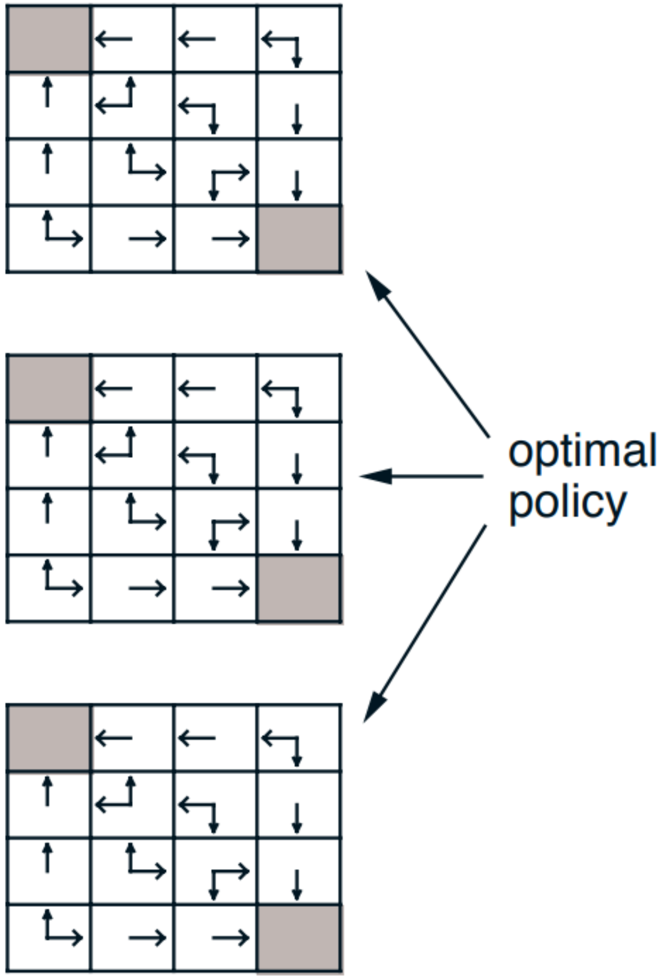
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-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	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Gridworld

$k = 3$

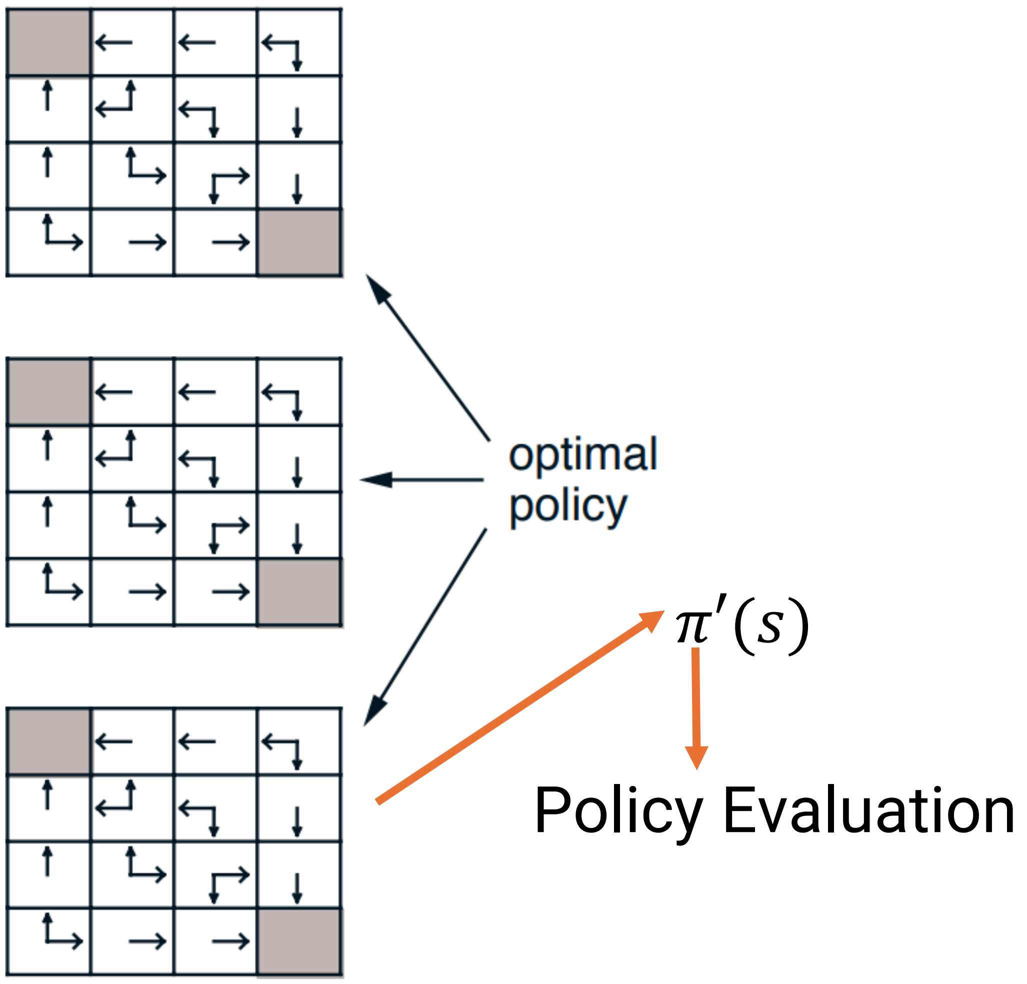
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-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	0.0

$k = \infty$

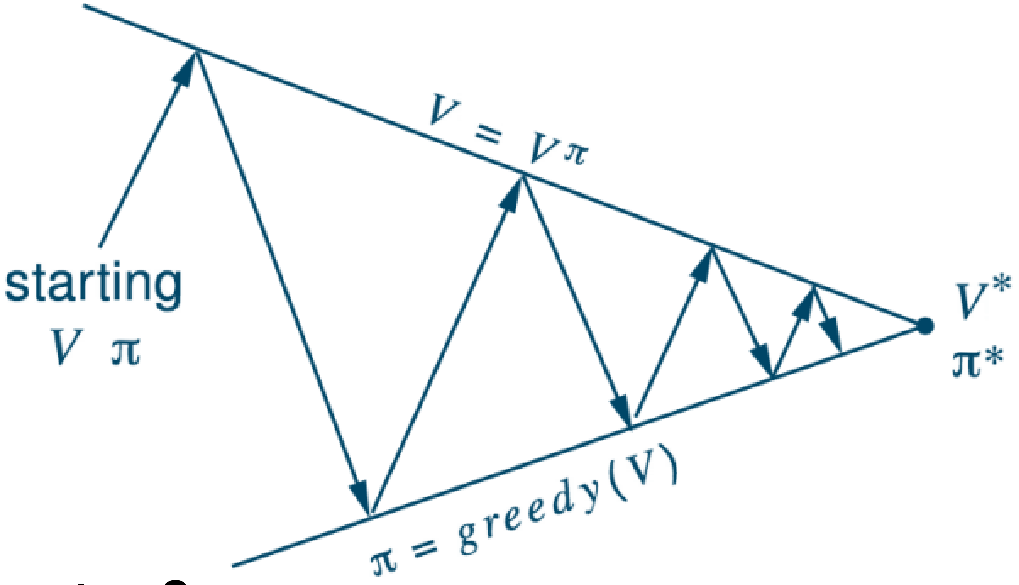
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

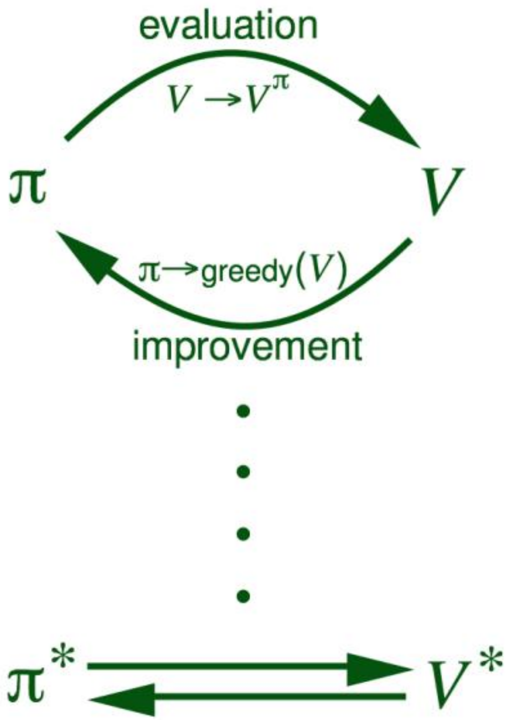


Policy Iteration

PI = Policy Evaluation + Policy Improvement

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





Convergence Guarantee?

Algorithm

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 $s \in \mathcal{S}$

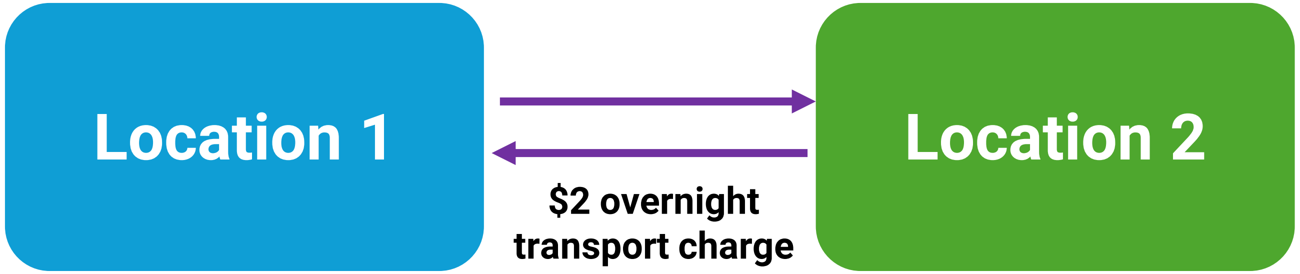
2. Policy Evaluation
   Loop:
   - $\Delta \leftarrow 0$
   - Loop for each $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   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* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   - 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

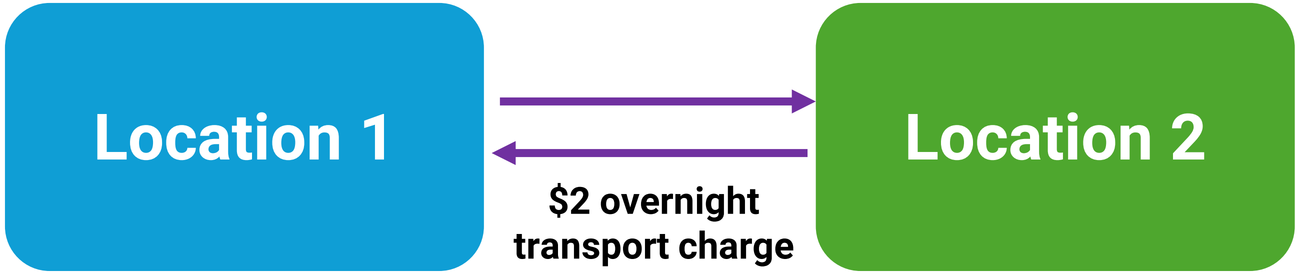
Example: Jack's Car Rental



- $10 flat rental fee
- Random rental rates (Poisson)
- Random return rates (Poisson)
- 20 car capacity per location

- States?
- Actions?

Example: Jack's Car Rental

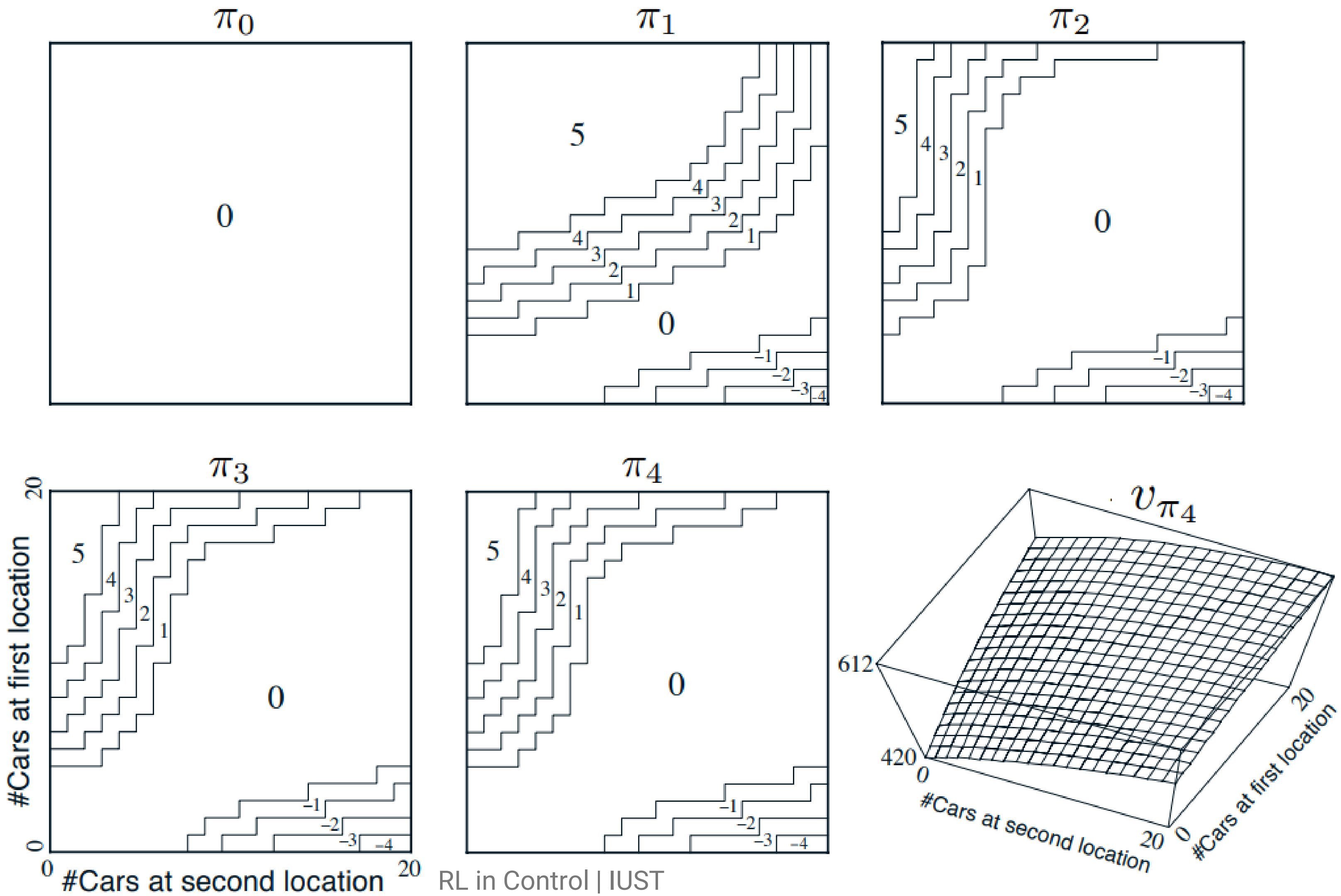


- $10 flat rental fee
- Random rental rates (Poisson)
- Random return rates (Poisson)
- 20 car capacity per location

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars between locations overnight
- Reward: $10 for each car rented (must be available)

Example: Jack’s Car Rental

Policy Iteration



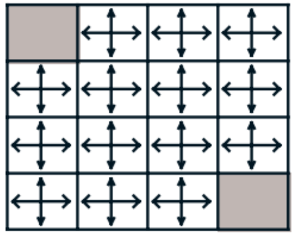
Gridworld

Initial policy: Uniform distribution

Compute π' after each computation of $\boldsymbol{V_\pi}$

$k = 0$

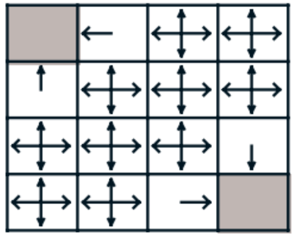
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

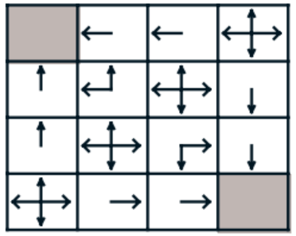
$k = 1$

0.0	-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



$k = 2$

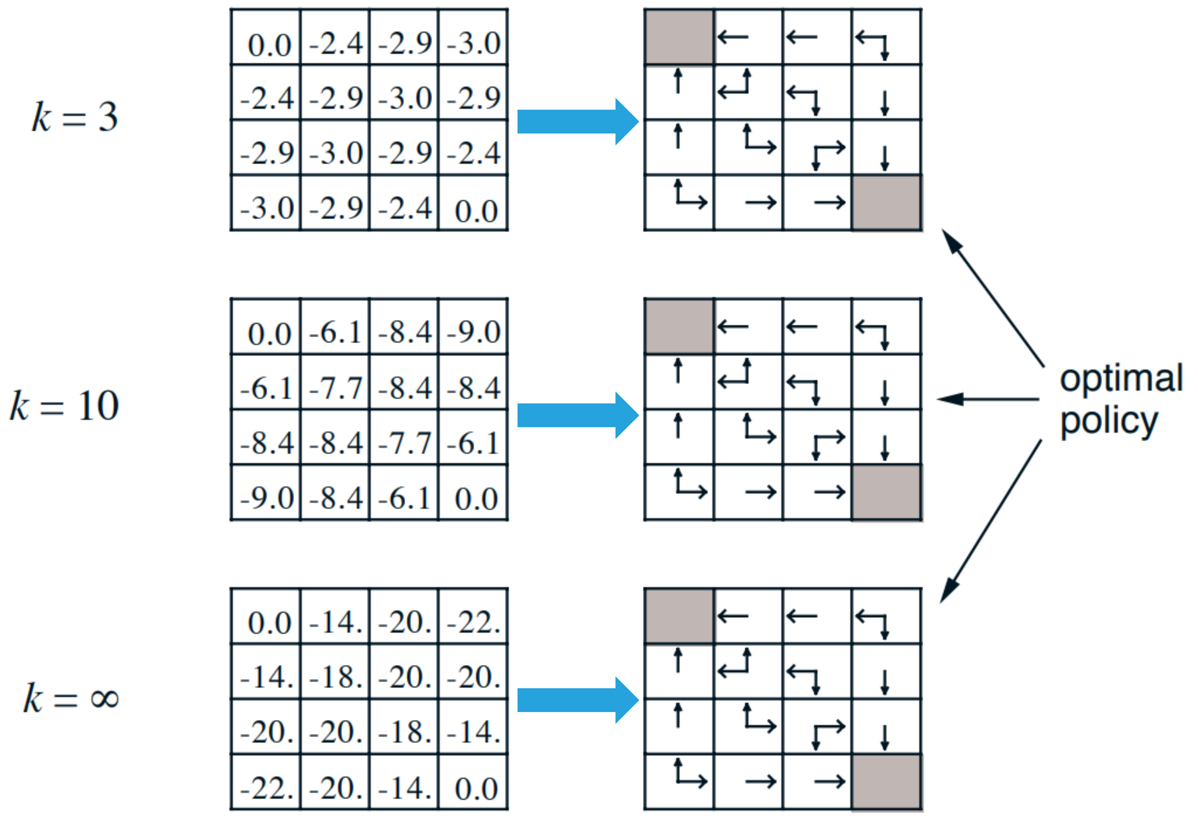
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Gridworld

Initial policy: Uniform distribution

Compute π' after each computation of V_π



Update of V_π approximation at each sample time

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

Compute $\boldsymbol{\pi}'$ for each approximation of $\boldsymbol{V_\pi}$

Algorithm

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

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

Example: Gambler's Problem

- **Total coins:** 100
- **Game:**
  - Probability of heads: 0.4
  - Bet a portion of the 100 coins
  - **Heads:** the bet is doubled
  - **Tails:** the bet is lost
- **End of the game:** when the gambler reaches either 0 or 100 coins

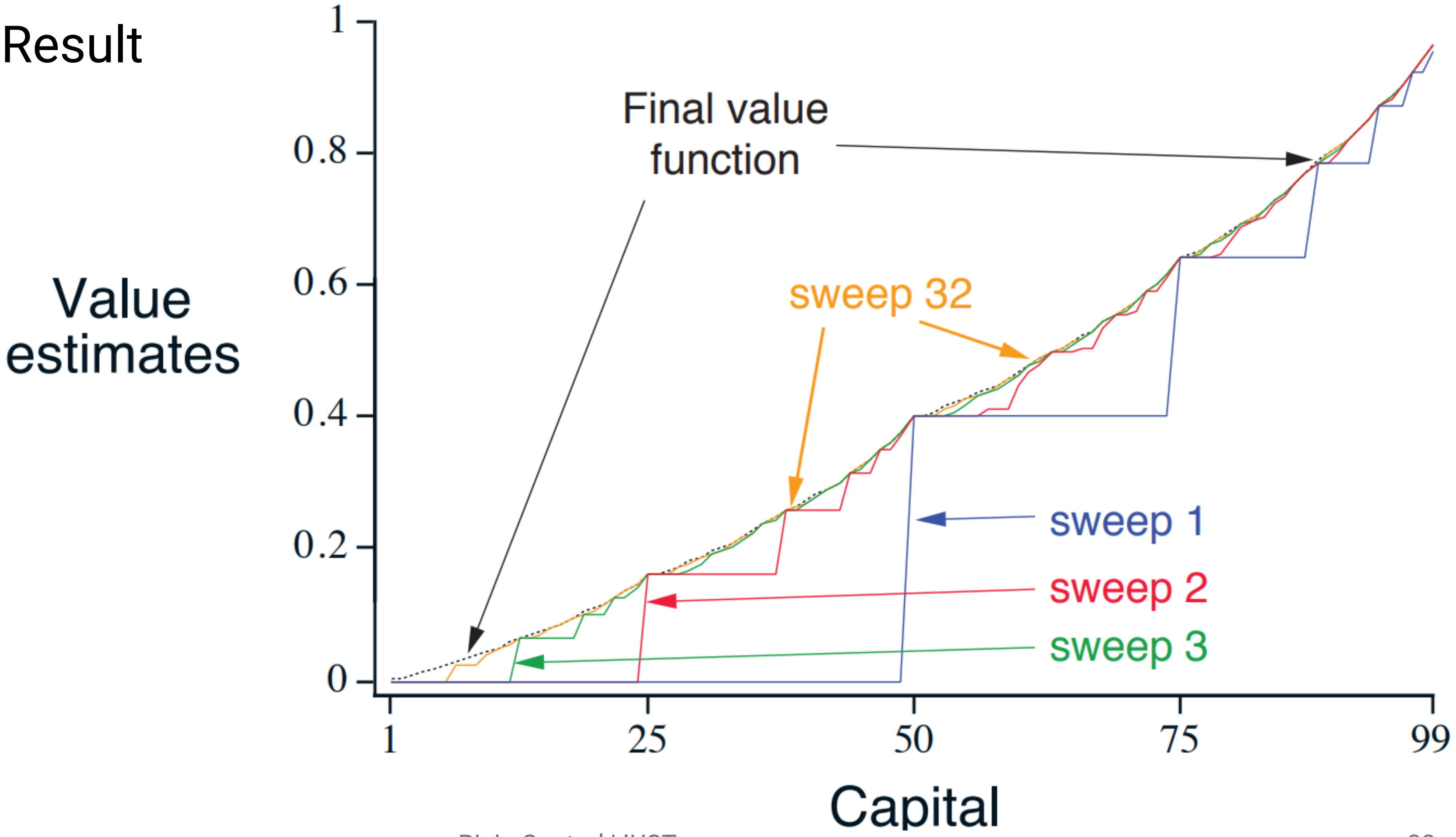
Action Space? $a \in \{0, 1, \ldots, \min(s, 100 - s)\}$

State Space? $s \in \{1, 2, \ldots, 99\}$

Reward? Reward: Goal:+1 any transition:-1

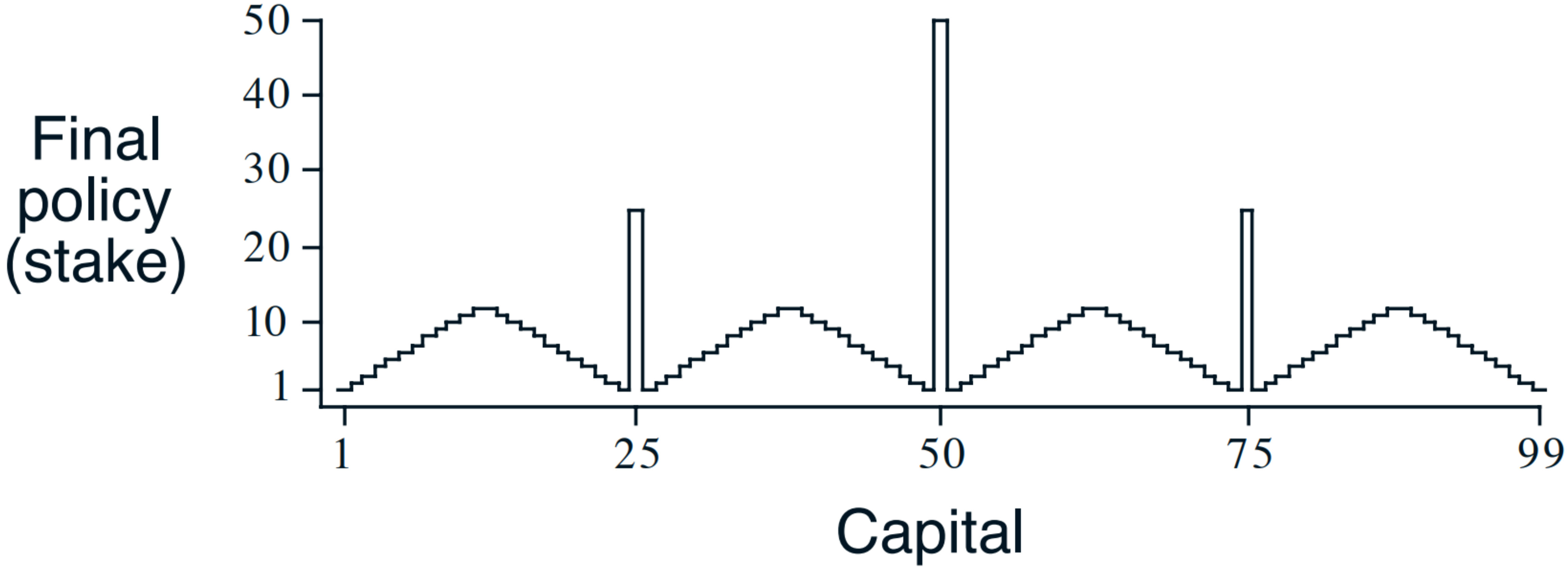
Example: Gambler's Problem

Value Iteration Result



Example: Gambler's Problem

Optimal Policy



Asynchronous Dynamic Programming

Updating all states in a large state space?

$\rightarrow$ Requires a lot of memory!

Solution:

In-Place Iteration Dynamic Programming

New capability:

Update **important** parts of the state space

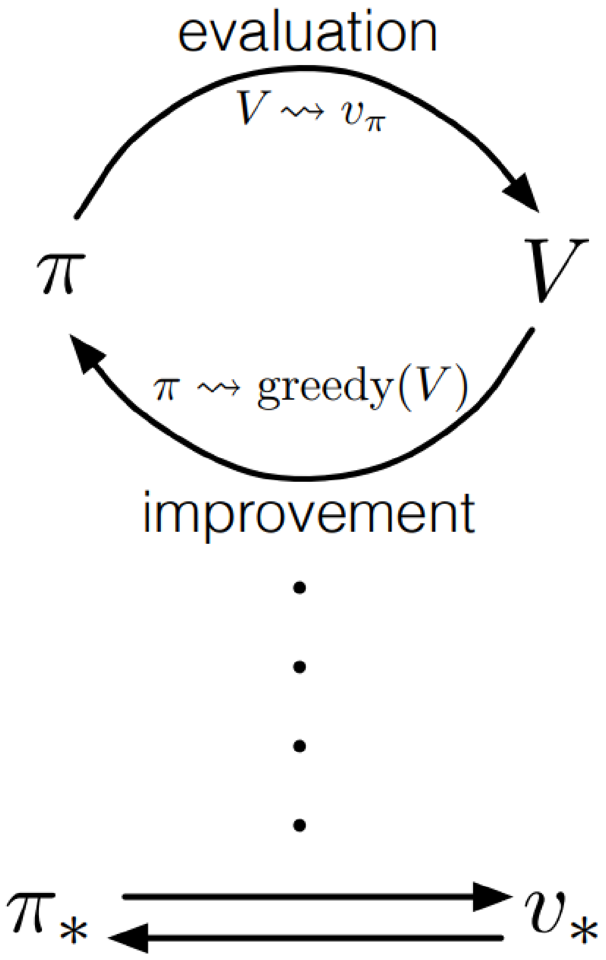
Perform **fewer or no updates** for **less important** parts

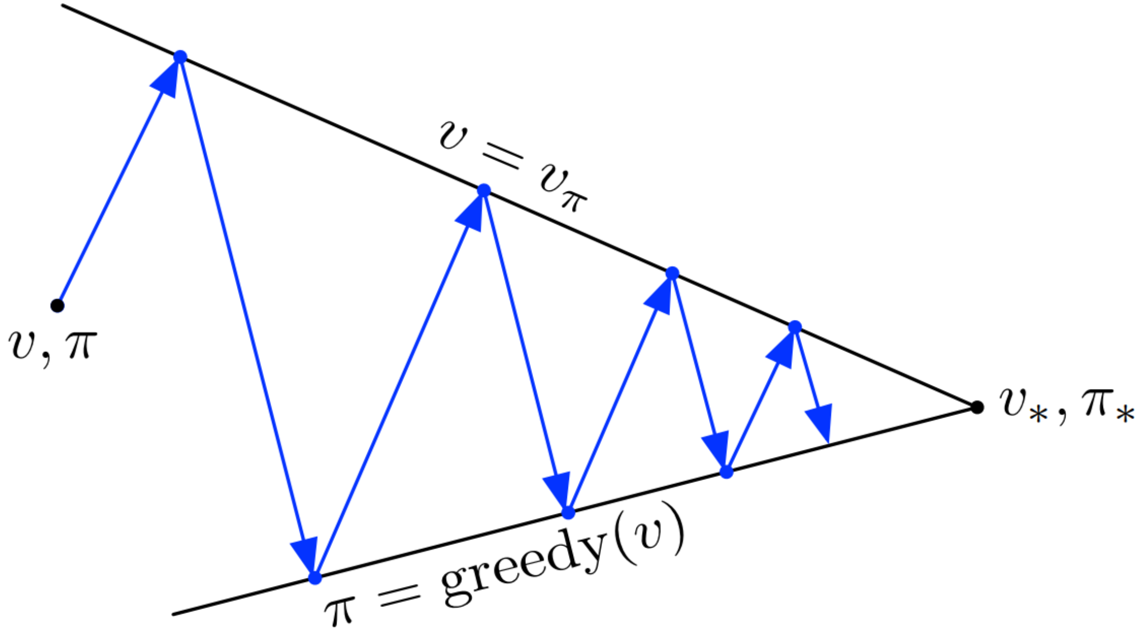
Update the value of **states the agent actually visits**

Challenge: Convergence and Optimality?!

Generalized Policy Iteration

VI or PI
Random Policy
Initial Value
….





Challenge: Convergence and Optimality?!

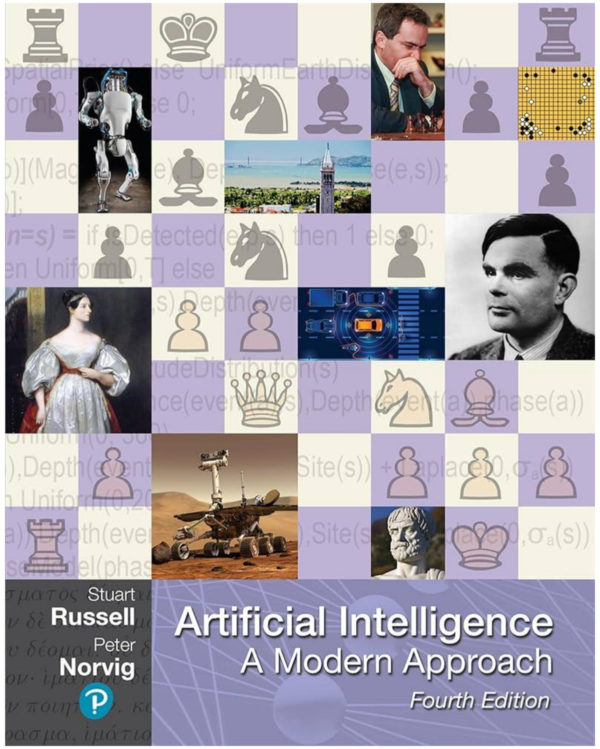
Efficiency of Dynamic Programming

n states
k actions

Policy space?
curse of dimensionality

LP & DP

Convergence of Dynamic Programming



Artificial Intelligence

A Modern Approach

By Stuart **Russell**,
and Peter **Norvig**

Convergence of Dynamic Programming

Return and Bellman Equation

A Utility of a state sequence is:

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

With discounted rewards, the utility of an infinite sequence is finite.

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1-\gamma)$$

Convergence of Dynamic Programming

Return and Bellman Equation

The expected utility obtained by executing π starting in s is given by

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t)\right]$$

Now, out of all the policies the agent could choose to execute starting in s, one (or more) will have higher expected utilities than all the others.

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s)$$

Convergence of Dynamic Programming

Return and Bellman Equation

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action. That is, the utility of a state is given by

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

This is called the Bellman equation, after Richard Bellman (1957).

Convergence of Dynamic Programming

Value Iteration

We start with arbitrary initial values for the utilities, calculate the right-hand side of the equation, and plug it into the left-hand side—thereby updating the utility of each state from the utilities of its neighbors. We repeat this until we reach an equilibrium. Let $U_i(s)$ be the utility value for state s at the ith iteration. The iteration step, called a Bellman update, looks like this:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

Convergence of Dynamic Programming

Convergence Analysis

The basic concept used in showing that value iteration converges is the notion of a contraction. Roughly speaking, a contraction is a function of one argument that, when applied to two different inputs in turn, produces two output values that are “closer together,” by at least some constant factor, than the original inputs.

Convergence of Dynamic Programming

Convergence Analysis

1. A contraction has only one fixed point; if there were two fixed points they would not get closer together when the function was applied, so it would not be a contraction.
2. When the function is applied to any argument, the value must get closer to the fixed point (because the fixed point does not move), so repeated application of a contraction always reaches the fixed point in the limit.

Convergence of Dynamic Programming

Convergence Analysis

View the Bellman update

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

as an operator B that is applied simultaneously to update the utility of every state. Let U_i denote the vector of utilities for all the states at the ith iteration. Then the Bellman update equation can be written as:

$$U_{i+1} \leftarrow B\, U_i$$

Convergence of Dynamic Programming

Convergence Analysis

Next, we need a way to measure distances between utility vectors. We will use the max norm, which measures the "length" of a vector by the absolute value of its biggest component:

$$||U|| = \max_{s} |U(s)|$$

With this definition, the "distance" between two vectors, is the maximum difference between any two corresponding elements.

Convergence of Dynamic Programming

Convergence Analysis

Let U_i and U_i' be any two utility vectors. Then we have

$$||B\,U_i - B\,U_i'|| \leq \gamma\,||U_i - U_i'||$$

That is, the Bellman update is a contraction by a factor of γ on the space of utility vectors.

Hence, from the properties of contractions in general, it follows that value iteration always converges to a unique solution of the Bellman equations whenever $\gamma < 1$.

Convergence of Dynamic Programming

Convergence Analysis

We can also use the contraction property to analyze the rate of convergence to a solution. In particular, we can replace U_i' with the true utilities U, for which $BU = U$. Then we obtain the inequality

$$||B\,U_i - U|| \leq \gamma\,\underbrace{||U_i - U||}_{\text{error}}$$

We see that the error is reduced by a factor of at least γ on each iteration. This means that value iteration converges exponentially fast.

Convergence of Dynamic Programming

Convergence Analysis

We can calculate the number of iterations required to reach a specified error bound ϵ as follows:

- First, recall from slide 38, that the utilities of all states are bounded by

$$\pm R_{\max}/(1-\gamma)$$

This means that the maximum initial error

$$||U_0 - U|| \leq 2R_{\max}/(1-\gamma)$$

Convergence of Dynamic Programming

Convergence Analysis

Suppose we run for N iterations to reach an error of at most ϵ. Then, because the error is reduced by at least γ each time, we require

$$\gamma^N \cdot 2R_{\max}/(1-\gamma) \leq \epsilon$$

Taking logs, we find

$$N = \lceil \log(2R_{\max}/\epsilon(1-\gamma))/\log(1/\gamma) \rceil$$

iterations suffice.

Convergence of Dynamic Programming

Convergence Analysis

The good news is that, because of the exponentially fast convergence, N does not depend much on the ratio ϵ/R_{max}. The bad news is that N grows rapidly as γ becomes close to 1. We can get fast convergence if we make γ small, but this effectively gives the agent a short horizon and could miss the long-term effects of the agent's actions.