Chapter 4
Dynamic Programming

- The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP)

Bellman equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Bellman optimality equation

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

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

4.1 Policy Evaluation (Prediction)

- For a policy π

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

$$
\boxed{\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}}
$$

- Iterative policy evaluation
  - For a policy π, consider a sequence of approximate value functions v_0, v_1, v_2, …

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

  - Stop iteration if

$$\max_s |v_{k+1}(s) - v_k(s)|$$

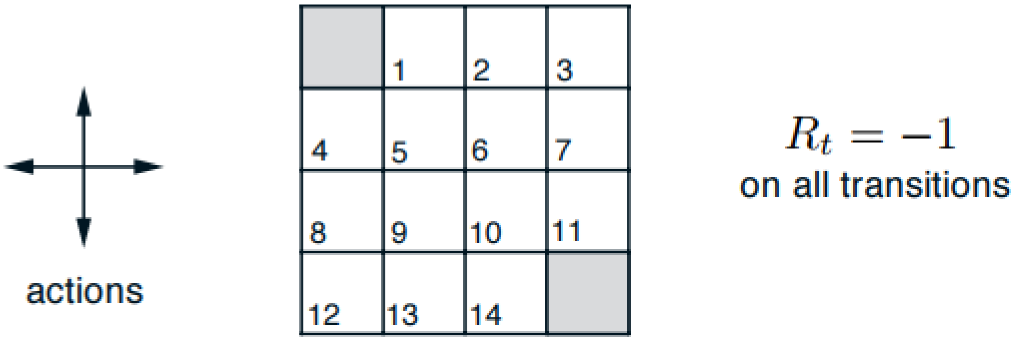
    is sufficiently small

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 $\Delta < \theta$

Example 4.1 Gridworld



$\mathcal{S} = \{1,2,\dots,14\}$
$\mathcal{A} = \{\text{up}, \text{down}, \text{right}, \text{left}\}$
Terminal state = shaded

Each action deterministically cause the corresponding state transitions, except that actions that would take the agent off the grid in fact leave the state unchanged.

$$p(6,-1|5,\text{right}) = 1,\ p(7,-1|7,\text{right}) = 1,$$
$$p(10,r|5,\text{right}) = 0 \text{ for all } r \in \mathcal{R}$$

v_k for the random policy, i.e. $\pi(a|s) = \frac{1}{4}$

$k = 0$

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

$$v_1(14) = \frac{1}{4}\big(-1 + v_0(13)\big) + \frac{1}{4}\big(-1 + v_0(10)\big) + \frac{1}{4}\big(-1 + v_0(15)\big) + \frac{1}{4}\big(-1 + v_0(14)\big)$$
$$= -1$$

$$v_1(14) = \frac{1}{4}\big(-1 + v_1(13)\big) + \frac{1}{4}\big(-1 + v_1(10)\big) + \frac{1}{4}\big(-1 + v_1(15)\big) + \frac{1}{4}\big(-1 + v_1(14)\big)$$
$$= -\frac{7}{4}$$

4.2 Policy Improvement

- Policy improvement
  - The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy

$$\begin{aligned}\pi'(s) &= \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) \\ &= \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)[r + v_\pi(s')]\end{aligned}$$

Example 4.1 Gridworld

v_k for the random policy, i.e. $\pi(a|s) = \frac{1}{4}$

$k = 0$

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

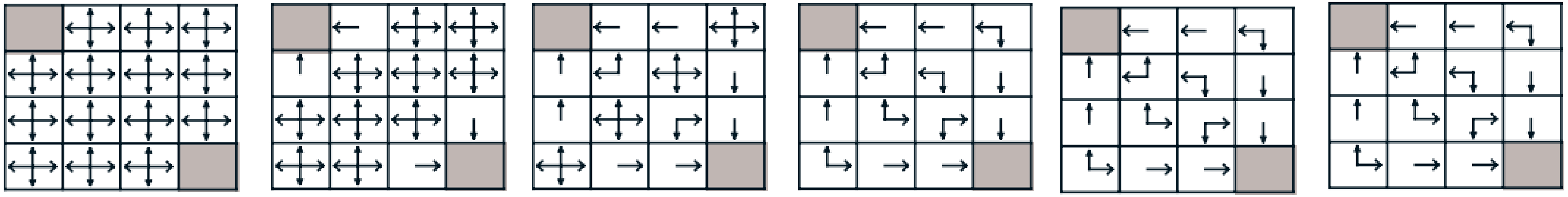
$k = 10$

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

Policies by greedy actions



4.3 Policy Iteration

Repeatation of policy evaluations and policy improvements

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

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

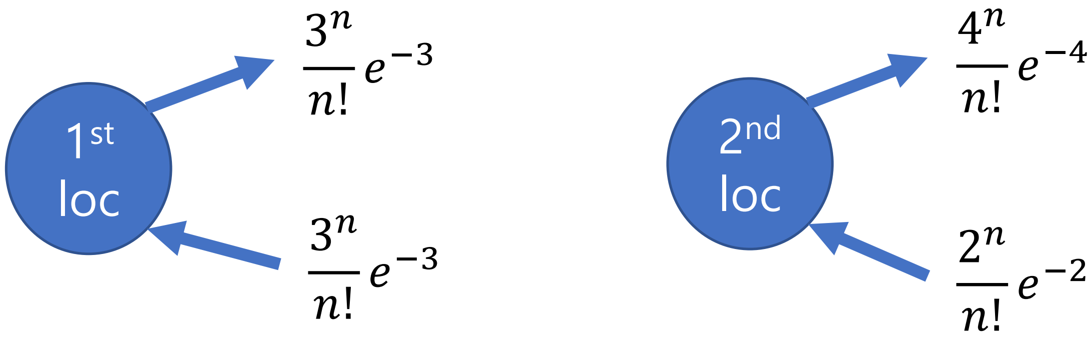
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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)

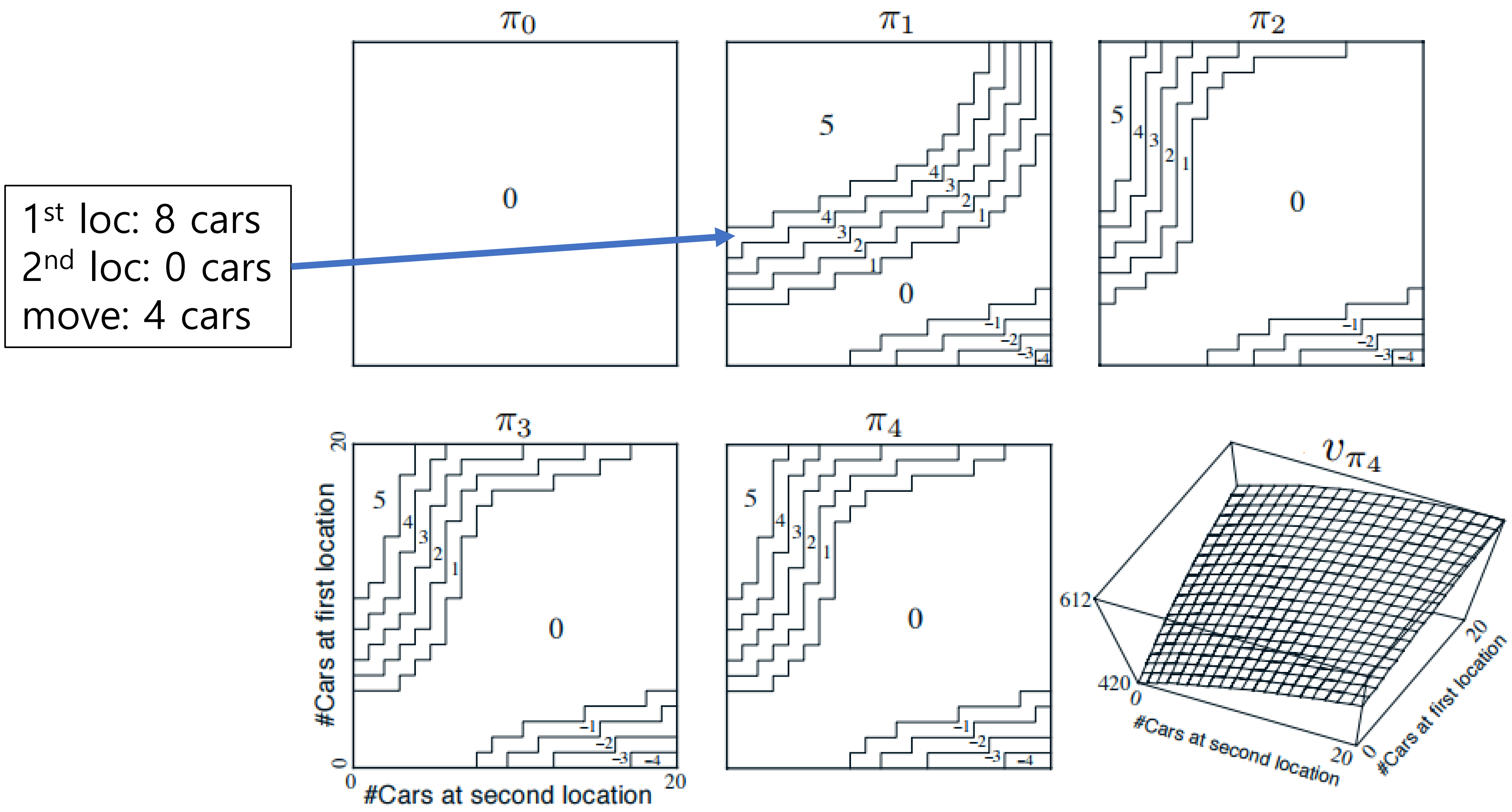
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Example 4.2: Jack's Car Rental

Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved.
We assume that the number of cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is n is $\frac{\lambda^n}{n!}e^{-\lambda}$, where λ is the expected number. Suppose λ is 3 and 4 for rental requests at the first and second locations and 3 and 2 for returns.
To simplify the problem slightly, we assume that there can be no more than 20 cars at each location and a maximum of five cars can be moved from one location to the other in one night.



Discount rate: $\gamma = 0.9$
Time step: day
State: the number of cars at each location at the end of the day
Action: the numbers of cars moved between the two locations overnight



4.4 Value Iteration

- Value iteration
  - Simple version of policy iteration
  - Policy evaluation is stopped after just one sweep

$$v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_k(s')]$$

max means greedy policy

  - Repeat the evaluation

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_*$$

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 $V(\textit{terminal}) = 0$

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

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

Example 4.1 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

$k = 1$

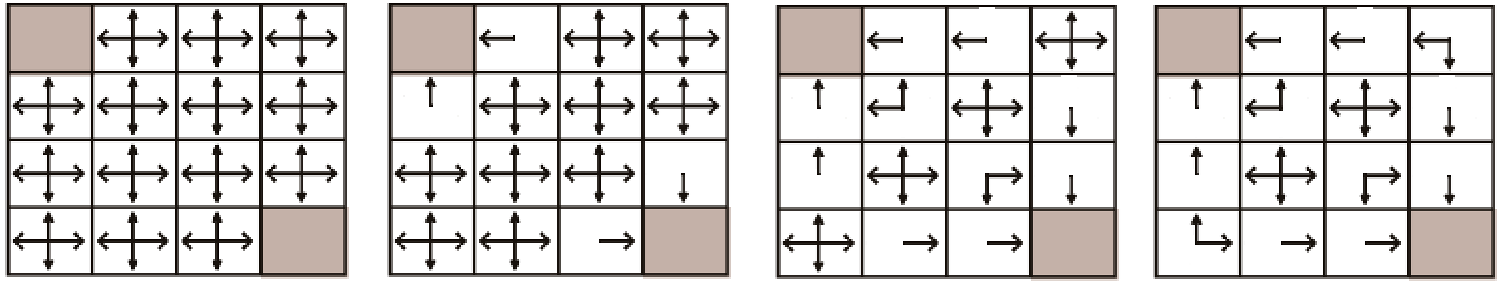
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.0	-2.0	-2.0
-1.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.0
-2.0	-2.0	-1.0	0.0

$k = 3$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0



Example 4.3: Gambler's Problem

A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal of $100, or loses by running out of money.
On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP.

State: the gambler's capital, $s \in \{1,2,\ldots,99\}$
Action: stake, $a \in \{0,1,\ldots,\min(s,100-s)\}$
Reward: +1 if $s = 100$
0 otherwise

State-value function: the probability of winning

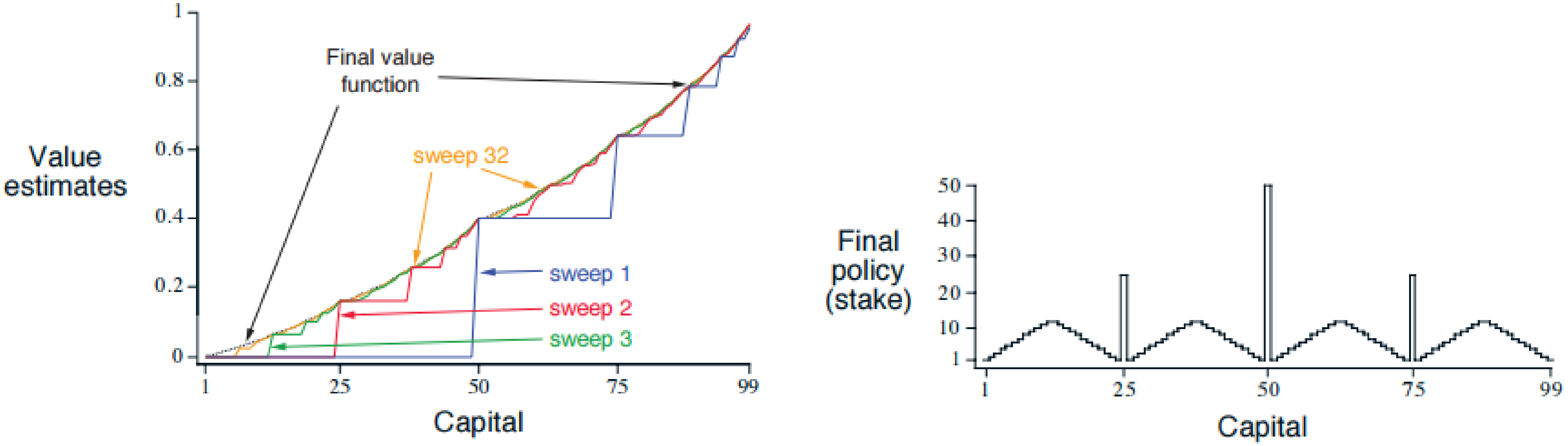


Figure 4.3: The solution to the gambler's problem for $p_h = 0.4$. The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy.

p_h: the probability of the coin coming up heads