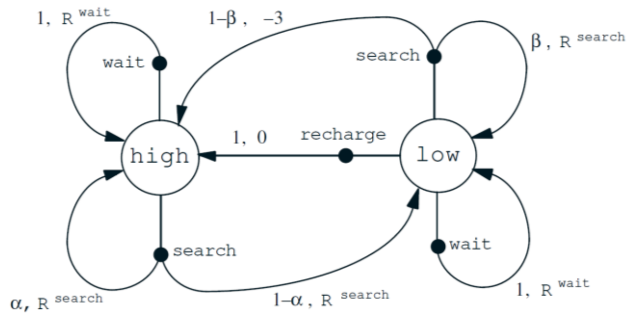


ECE 414/517 Reinforcement Learning

LECTURE 8: DYNAMIC PROGRAMMING

SEP. 20 2022

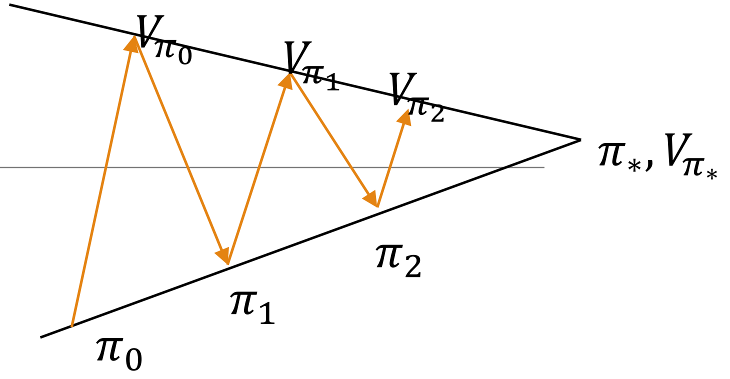
(ADAPTED FROM SLIDES BY DR. ITAMAR AREL)

Dynamic Programming

Dynamic programming is the collection of algorithms that can be used to compute optimal policies given perfect model of the environment.

1. DP constitutes a theoretically optimal methodology
2. In reality it is often limited since DP is computationally expensive.
3. As related to other methods:
   - Hope to do “just as well” as DP
   - Hopefully can do it with less computation
   - Hopefully can do it with less memory
   - Hopefully without model
4. Key Idea (In general): Use value function to derive optimal policy
5. Need to find a way to calculate optimal state value function

Policy Iteration

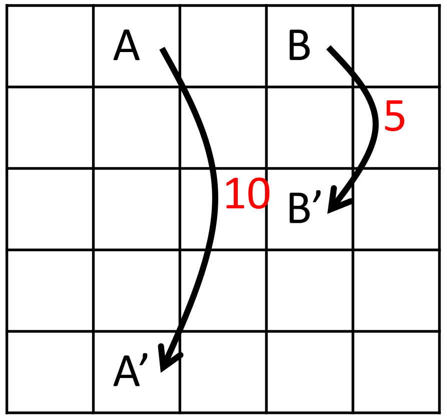


1. Technique for obtaining the optimal policy
2. Comprises of two complementing steps:
   - Policy Evaluation: updating the value function in view of current policy
     - $V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')]$

       ⬇
     - $V^\pi_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi_k(s')]$
   - Policy Improvement: updating the policy given the current value function (which can be sub-optimal)
     - $\pi'(s) = arg\max_a \sum_{s',r} p(s',r\,|s,a)[r + \gamma V^\pi(s')]$
   - It can be shown that $\pi'$is at least as good as π, and if they are equal they are both the optimal policy.

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$$

Example: Grid World – Policy Evaluation



$$\pi(s) = \{0.25, 0.25, 0.25, 0.25\} \forall s \in S$$
$$\gamma = 0.9$$

$$V_{k+1}^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V_k^{\pi}(s')]$$

3.31	8.79	4.43	5.32	1.49
1.52	2.99	2.25	1.91	0.55
0.05	0.74	0.67	0.36	-0.40
-0.97	-0.44	-0.35	-0.59	-1.18
-1.86	-1.34	-1.23	-1.42	-1.97

V_{π_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_0

-0.50	10.0	-0.25	5.00	-0.5
-0.25	0.00	0.00	0.00	-0.25
-0.25	0.00	0.00	0.00	-0.25
-0.25	0.00	0.00	0.00	-0.25
-0.50	-0.25	-0.25	-0.25	-0.5

V_1

1.47	9.78	3.07	5.00	0.34
-0.48	2.19	-0.06	1.07	-0.48
-0.42	-0.06	0.00	-0.06	-0.42
-0.48	-0.11	-0.06	-0.11	-0.48
-0.84	-0.48	-0.42	-0.48	-0.84

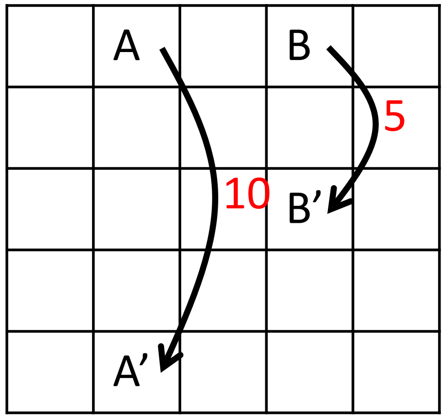
V_2

2.25	9.57	3.75	4.95	0.67
0.37	2.07	1.42	0.99	-0.13
-0.57	0.37	-0.05	0.12	-0.57
-0.66	-0.24	-0.14	-0.24	-0.66
-1.09	-0.66	-0.57	-0.66	-1.09

V_3

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$$

Example: Grid World – Policy Improvement



$$\pi_0(s) = \{0.25, 0.25, 0.25, 0.25\}\ \forall s \in S$$

$$\gamma = 0.9$$

$$\pi'(s) = arg\max_a \sum_{s',r} p(s', r\,|s, a)[r + \gamma V^{\pi}(s')]$$

3.31	8.79	4.43	5.32	1.49
1.52	2.99	2.25	1.91	0.55
0.05	0.74	0.67	0.36	-0.40
-0.97	-0.44	-0.35	-0.59	-1.18
-1.86	-1.34	-1.23	-1.42	-1.97

V_{π_o}

→	↓	←	↓	←
↑	↑	↑	↑	←
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑

π_1

$$\pi(0,0) = \text{argmax} \begin{array}{c|c} \begin{bmatrix} -1 + 0.9 \cdot 3.31 \\ -1 + 0.9 \cdot 3.31 \\ 0 + 0.9 \cdot 8.79 \\ 0 + 0.9 \cdot 1.52 \end{bmatrix} & \begin{matrix} W \\ N \\ E \\ S \end{matrix} \end{array} = E$$

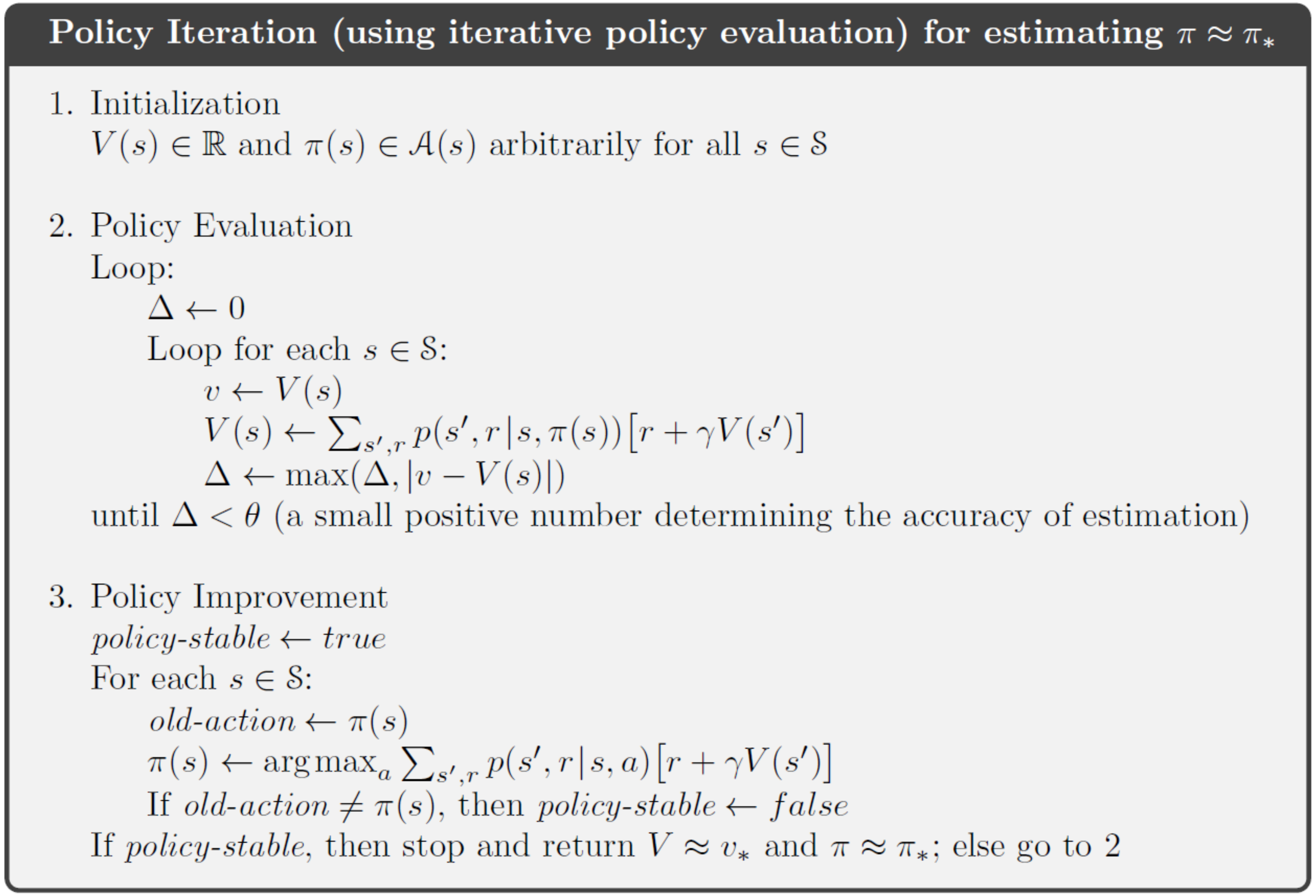
$$= \{0, 0, 1, 0\}$$

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$$

Policy Improvement

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

Policy Iteration



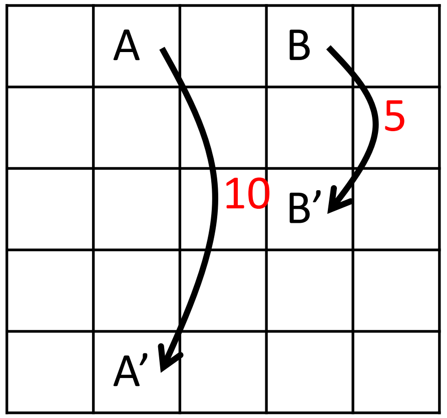
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

```
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* $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
       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: Grid World – Policy Evaluation



$$\pi_0(s) = \{0.25,0.25,0.25,0.25\}\ \forall s \in S$$
$$\gamma = 0.9$$

$$V_{k+1}^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V_k^{\pi}(s')]$$

3.31	8.79	4.43	5.32	1.49
1.52	2.99	2.25	1.91	0.55
0.05	0.74	0.67	0.36	-0.40
-0.97	-0.44	-0.35	-0.59	-1.18
-1.86	-1.34	-1.23	-1.42	-1.97

V_{π_o}

→	↓	←	↓	←
↑	↑	↑	↑	←
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑

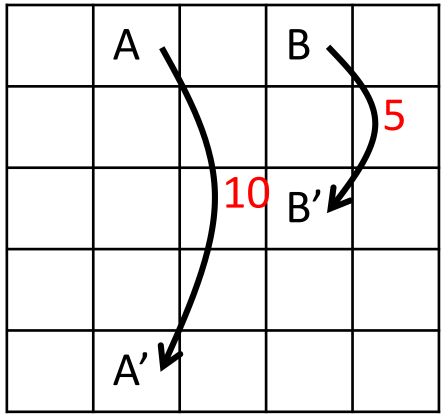
π_1

21.98	24.42	21.98	19.42	17.48
19.78	21.98	19.78	17.80	16.02
17.80	19.78	17.80	16.02	14.42
16.02	17.80	16.02	14.42	12.98
14.42	16.02	14.42	12.98	11.68

V_{π_1}

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$$

Example: Grid World – Policy Improvement



$$\pi_0(s) = \{0.25,0.25,0.25,0.25\}\ \forall s \in S$$
$$\gamma = 0.9$$

$$\pi'(s) = arg\max_a \sum_{s',r} p(s',r\,|s,a)[r + \gamma V^{\pi}(s')]$$

3.31	8.79	4.43	5.32	1.49
1.52	2.99	2.25	1.91	0.55
0.05	0.74	0.67	0.36	-0.40
-0.97	-0.44	-0.35	-0.59	-1.18
-1.86	-1.34	-1.23	-1.42	-1.97

V_{π_o}

→	↓	←	↓	←
↑	↑	↑	↑	←
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑

π_1

21.98	24.42	21.98	19.42	17.48
19.78	21.98	19.78	17.80	16.02
17.80	19.78	17.80	16.02	14.42
16.02	17.80	16.02	14.42	12.98
14.42	16.02	14.42	12.98	11.68

V_{π_1}

→	↓	←	↓	←
↑	↑	↑	↑	←
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑
↑	↑	↑	↑	↑

π_2

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$$

Value Iteration

$$V_{k+1}^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \alpha V_k^{\pi}(s')]$$

$$V_0 \rightarrow V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \cdots \rightarrow V^{\pi}$$

$$\pi'(s) = arg \max_{a} \sum_{s',r} p(s',r\ |s,a)[r + \alpha V^{\pi}(s')]$$

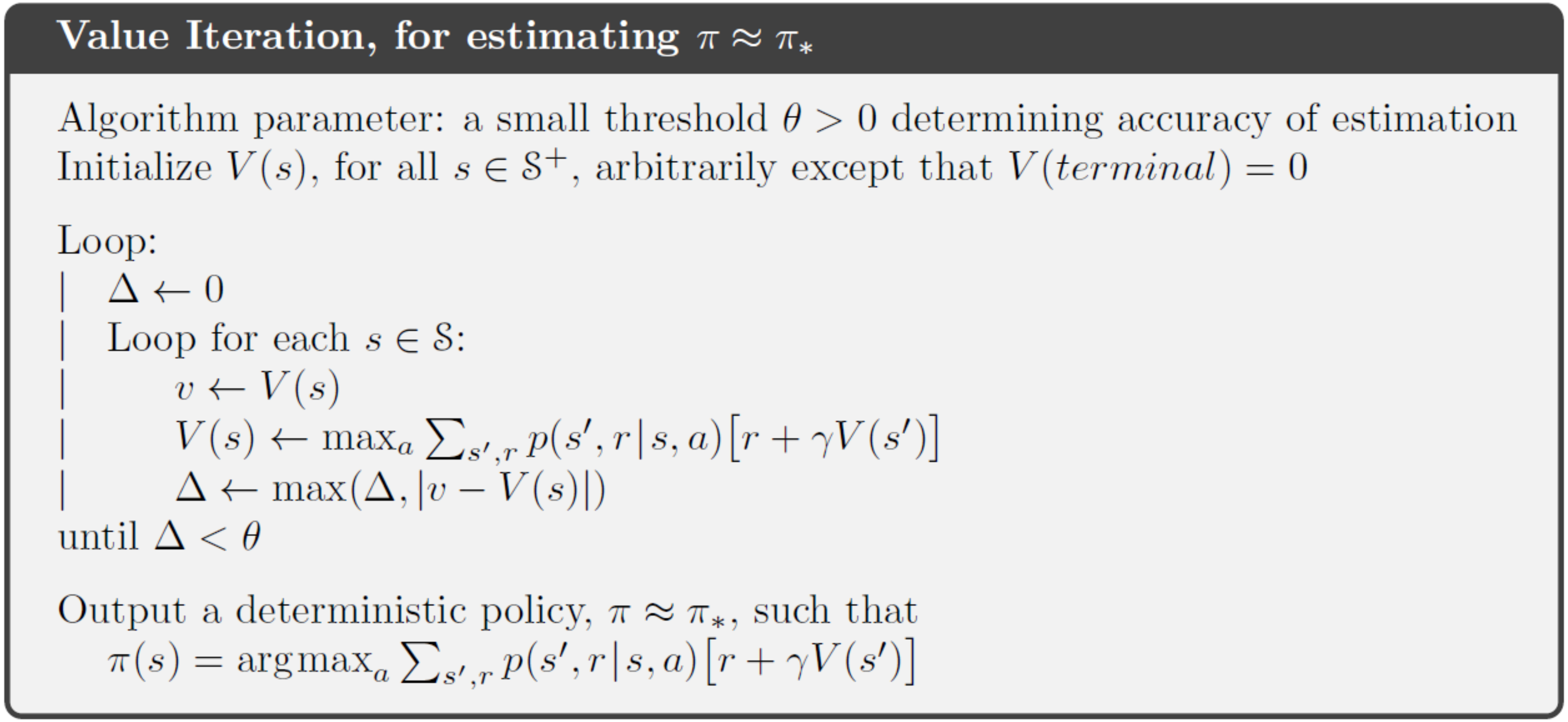
$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$$

$$\Downarrow$$

$$V_{k+1}(s) = \max_{a} \sum_{s',r} p(s',r|s,a)\left[r + \alpha V_k\ (s')\right]$$

Solving the Bellman Optimality Equation Iteratively

Value Iteration



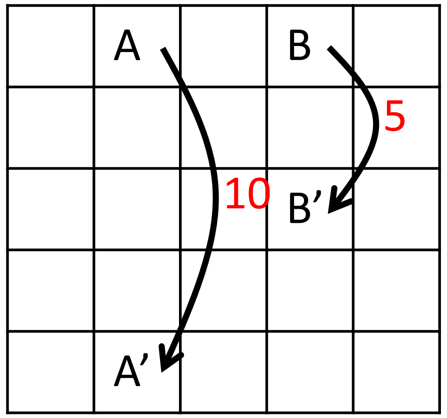
Value Iteration, for estimating $\pi \approx \pi_*$

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
```

Example: Grid World



$$\pi(s) = \{0.25,0.25,0.25,0.25\} \forall s \in S$$
$$\gamma = 0.9$$

V_1

-0.50	10.0	2.00	5.00	0.63
-0.36	2.17	0.94	1.34	0.19
-0.33	0.41	0.30	0.37	-0.12
-0.32	0.02	0.07	0.10	-0.26
-0.57	-0.37	-0.32	-0.30	-0.62

$$V_{k+1}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \alpha V_k^{\pi}(s')]$$

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_0

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\left[r + \alpha V_k\ \ (s')\right]$$

V_1

0.00	10.00	9.00	5.00	4.50
0.00	9.00	8.10	7.29	6.56
0.00	8.10	7.29	6.56	5.90
0.00	7.29	6.56	5.90	5.31
0.00	6.56	5.90	5.31	4.78

$$V_1(0,0) = \max_a \begin{bmatrix} -1 + 0.9 \cdot 0 \\ -1 + 0.9 \cdot 0 \\ 0 + 0.9 \cdot 0 \\ 0 + 0.9 \cdot 0 \end{bmatrix} \begin{array}{c} W \\ N \\ E \\ S \end{array} = 0$$

$$V_1(0,1) = 10$$

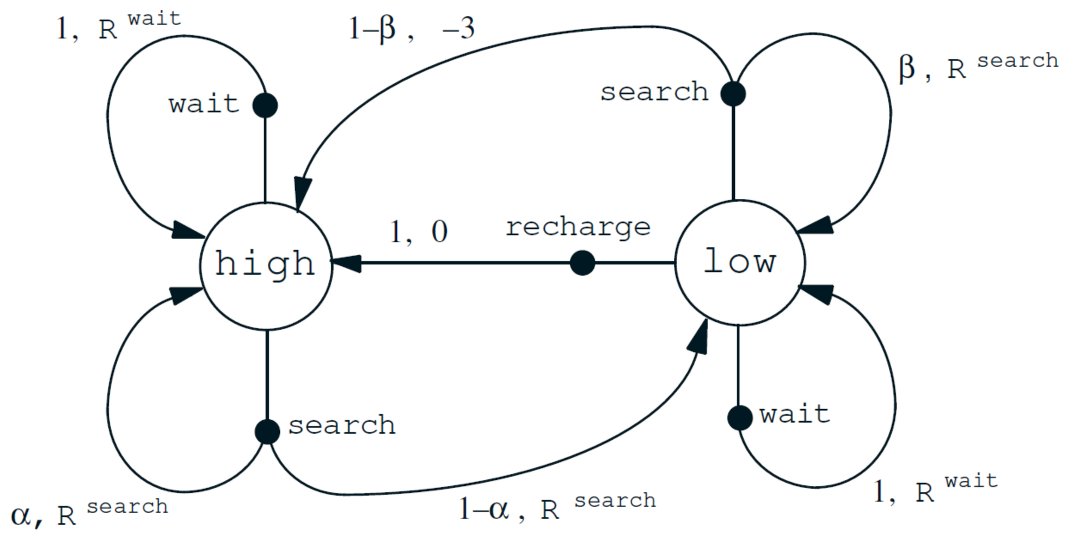
$$V_2(0,2) = \begin{bmatrix} 0 + 0.9 \cdot 10 \\ -1 + 0.9 \cdot 0 \\ 0 + 0.9 \cdot 0 \\ 0 + 0.9 \cdot 0 \end{bmatrix} = 9$$

Recycling Robot

h: high
l: low
s: search
w: wait
re: recharge

$$V^*(h) = \max \begin{Bmatrix} R^{search} + \alpha\gamma V^*(h) + (1-\alpha)\gamma V^*(l)] \\ R^{wait} + \gamma V^*(h) \end{Bmatrix}$$

$$V^*(l) = \max \begin{Bmatrix} \beta[R^{search} + \gamma V^*(l)] + (1-\beta)[-3 + \gamma V^*(h)] \\ R^{wait} + \gamma V^*(l) \\ \gamma V^*(h) \end{Bmatrix}$$



Gambler's Problem

1. Make bets on the outcomes of a sequence of coin flips
2. If the coin comes up heads, win as many dollars as she staked on that flip.
3. If tails, she loses her stake.
4. The game ends by reaching the goal of $100, or losing by running out of money.
5. On each flip the gambler must decide how much of his capital to stake.
6. This problem could be formulated as an undiscounted, finite (non-deterministic MDP).
   - States? $\{0, \dots, 100\}$ (how much \$)
   - Actions? $A(s) = [1, \dots s]$ (how much \$ to bet)
   - Rewards? +1 win \$ 0 lose or other action
7. If probability of heads is known can be solved using dynamic programming

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r | s, a)\left[r + \alpha V_k\ (s')\right]$$

$P_h = 0.4$

$V_1(0) = 0$ (bc terminal state)

$\vdots$

$V_1(1) = [0.6 \cdot (0+0) + 0.4 \cdot (0+0)] = 0$

$$V_1(2) = \max_a \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

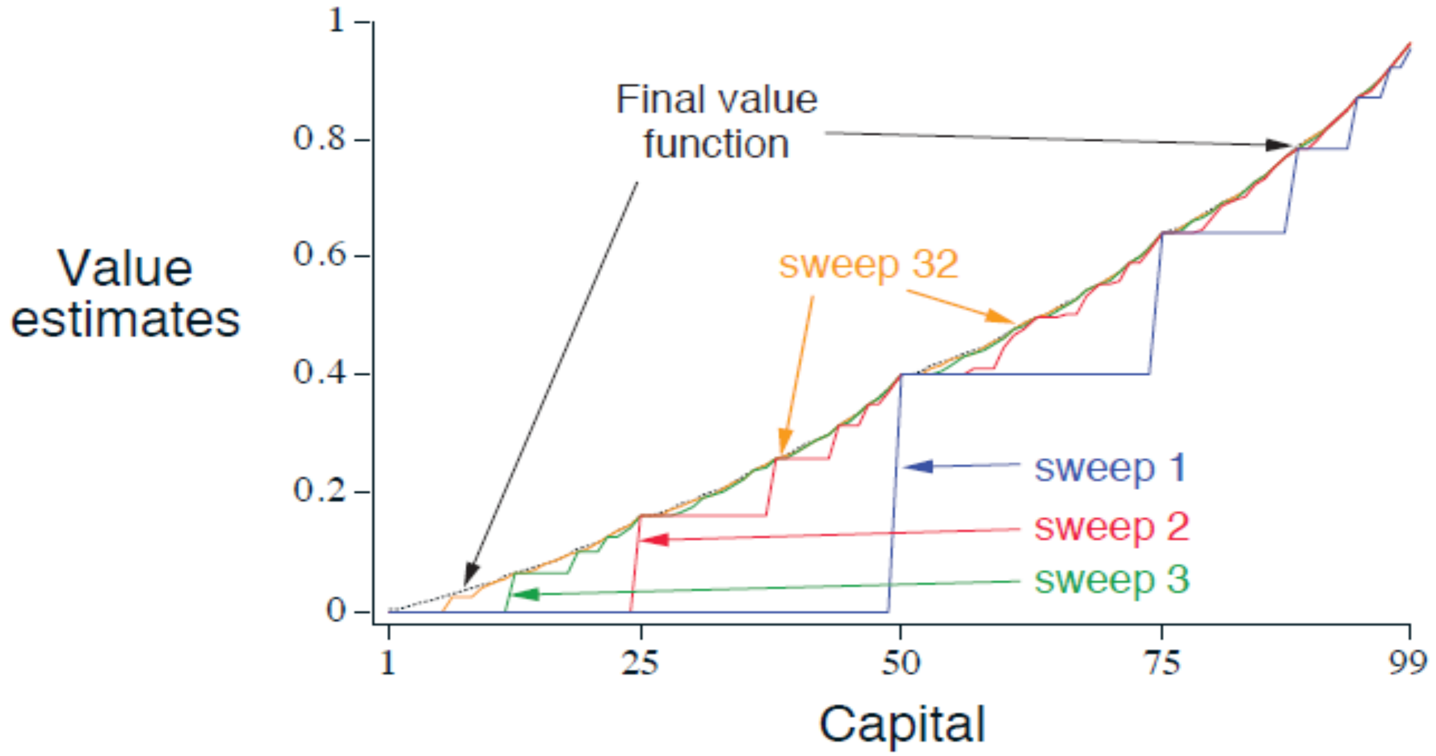
$\vdots$

$$V_1(50) = \begin{bmatrix} \vdots \\ 0.6 \cdot (0+0) + 0.4 \cdot (1+0) = 0.4 \end{bmatrix}_{(50)} = 0.4$$

$$V_1(51) = \begin{bmatrix} 0.6(0+0.4) + 0.4(0.0) \\ 0.6 \cdot (0+0) + 0.4(1+0) \end{bmatrix}^{(1)}_{(49)} = 0.4$$

$V_1(100) = 0$

Gambler’s Problem



$p_h = 0.4$

Efficiency of Dynamic Programming

1. DP may not be practical for large problems (even when environment is known).
2. Still a lot better than exhaustive search.
3. If $|A| = M$ and $|S| = N$ then *DP* is guaranteed to find the optimal policy in less then some polynomial function of M and N
4. Exhaustive Search:
    1. M^N
5. Usually impractical because $|S|$ grows exponentially with the number of state variables (curse of dimensionality)
6. These days DP methods can solve (using PC) MDP's with millions of states.