Dynamic Programming

- Collection of algorithms used to compute optimal policies given a perfect model of the environment as an MDP
- DP provides an essential foundation for understanding reinforcement learning algorithms

MDP recap

- Assume environment is a finite MDP (states, actions, rewards)
- Dynamics given by set probabilities $p(s', r | s, a)$
- Bellman optimally equations:

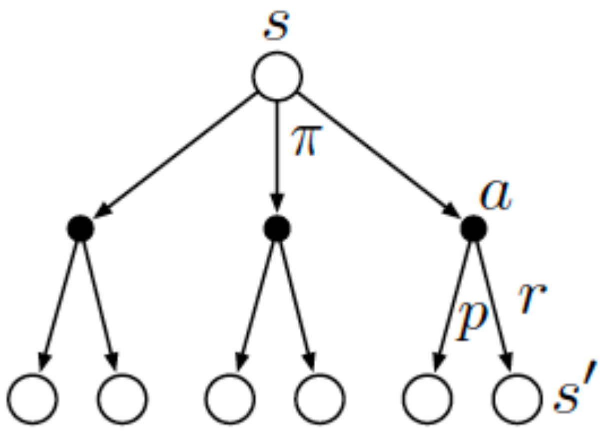
$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a) \Big[r + \gamma v_*(s') \Big], \text{ or}
\end{aligned}
$$

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

Policy Evaluation (Prediction)

- State-value function expected update rule (based on Bellman equation):

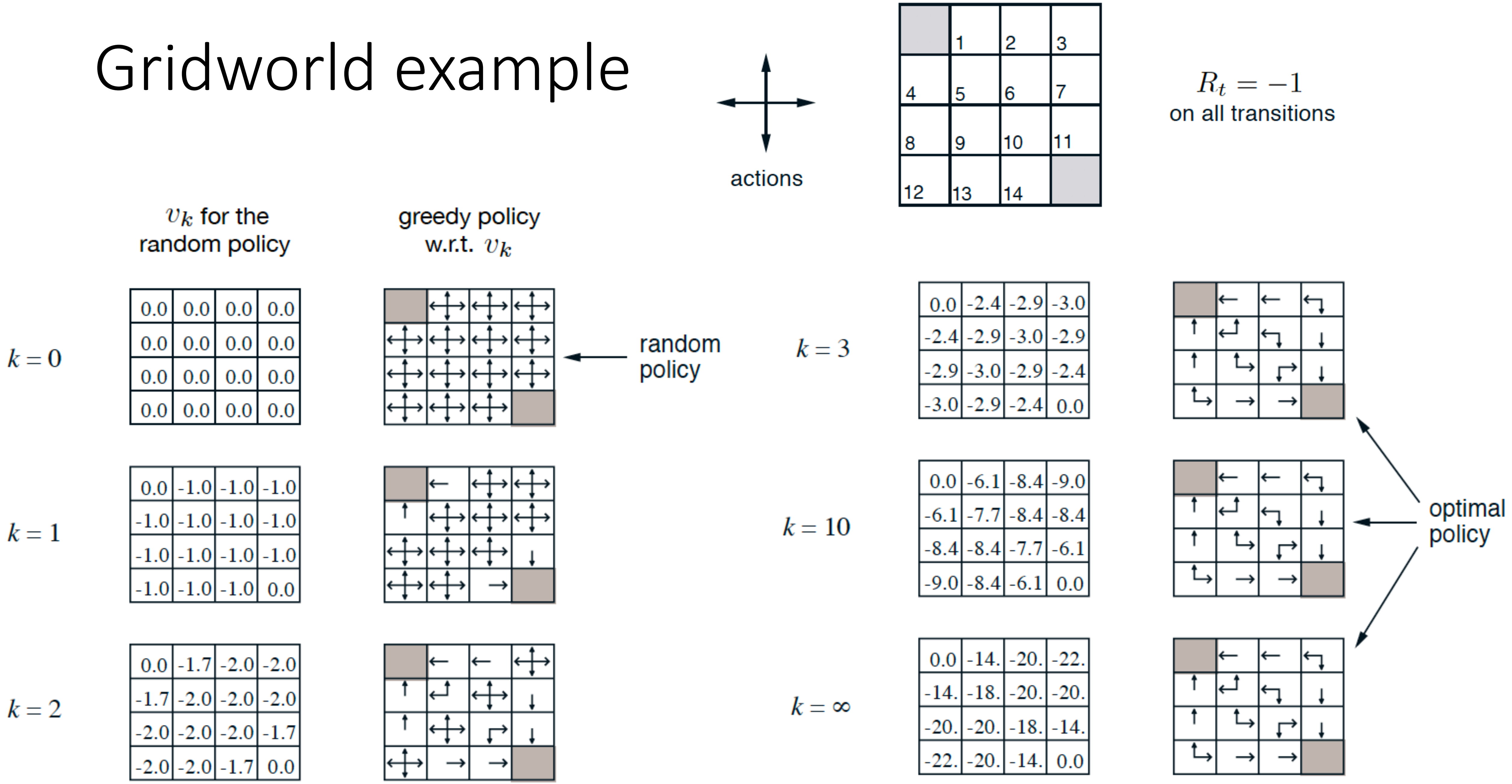
$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$



Backup diagram for v_π

- Bootstrapping: update estimates on the basis of estimates.
- Convergence (update no longer results in value changes) guaranteed as number of samples → ∞

Gridworld example



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 $\Delta < \theta$

Policy Improvement

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

- Policy improvement theorem: policy π' must be as good as, or better than, π.
$$v_{\pi'}(s) \geq v_\pi(s).$$

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

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 $s \in \mathcal{S}$

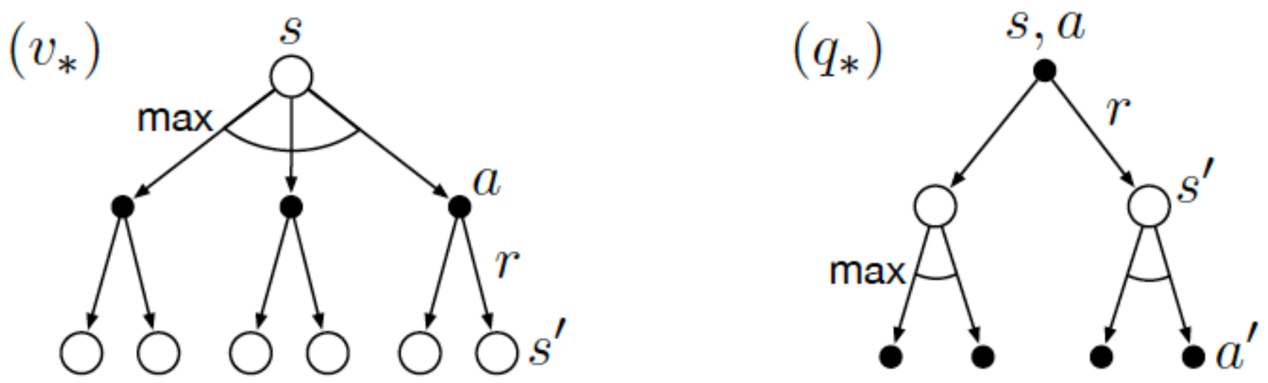
2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s') \big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until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

Value Iteration

- Truncate policy evaluation to a single sweep/update
- Update rule that combines policy improvement and evaluation (Bellman optimally equation):

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



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

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

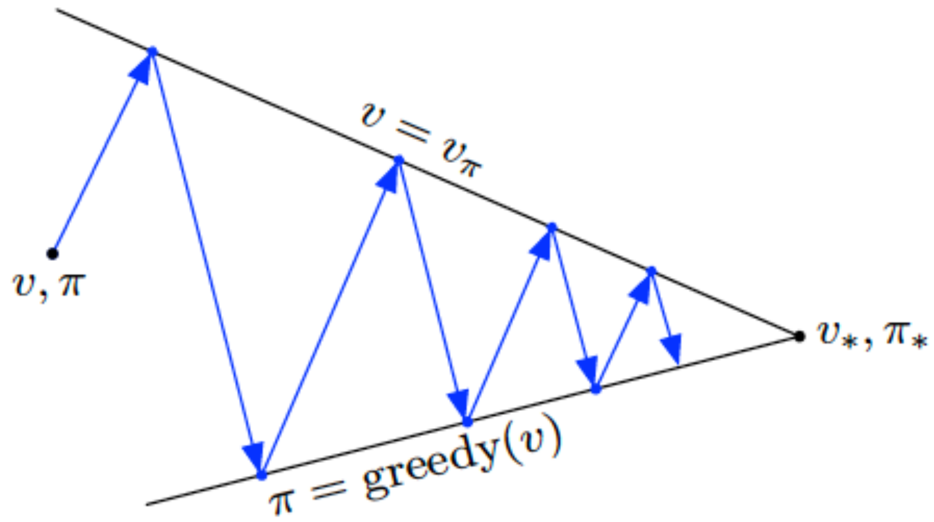
Asynchronous Dynamic Programming

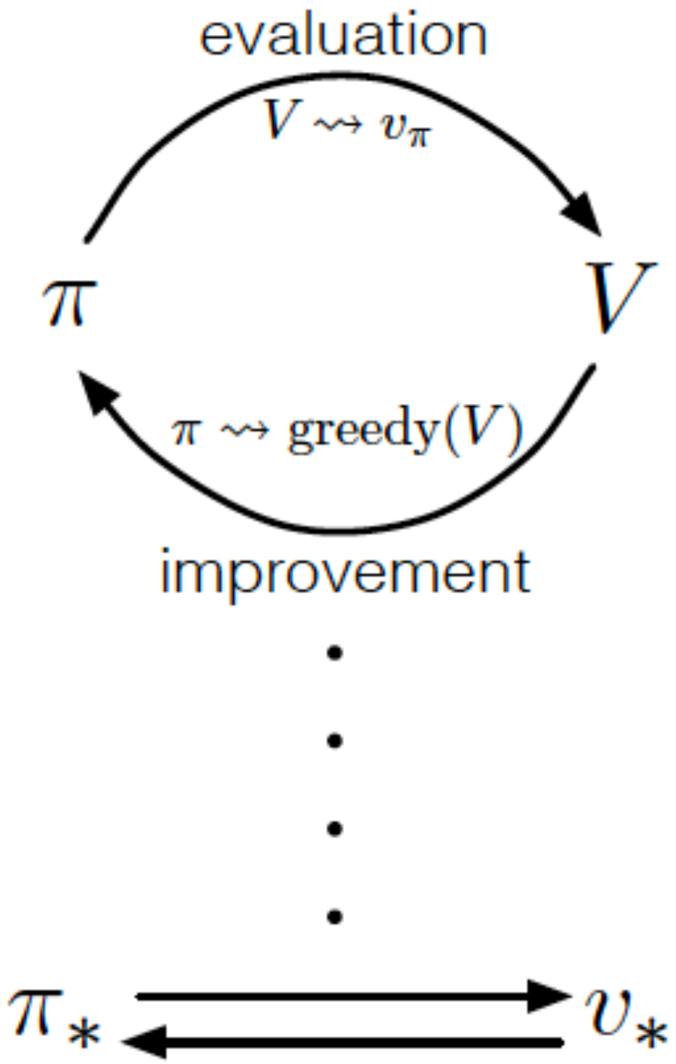
- Synchronous involves a complete sweep of the state space every iteration
- Asynchronous is preferred for problems with large state spaces where the values of states are updated in any order
- Allows flexibility in selecting states to update and the policy can be improved more frequently
- To converge correctly, asynchronous algorithms must continue to update the values of all the states
- Allows real-time interaction

Generalized Policy Iteration

Two interacting processes:

- Policy evaluation - makes the value function consistent with the current policy
- Policy improvement - make the policy greedy with respect to the current value function





Efficiency of Dynamic Programming

- Worst case time DP methods take to find an optimal policy is polynomial in the number of states (n) and actions (k).
- Faster convergence with good initial value functions or policies.
- However, direct search of policy space is exponential in state space
- Classical DP algorithms are of limited use in RL because of known dynamics assumption and great computational expense