# Lecture 3: Model-based RL

Lei Zhang

## Outline

- Introduction to dynamic programming
- Policy Evaluation
- Policy Iteration
- Value Iteration
- Extensions

*Materials are modified from David Silver's RL lecture notes

## Outline

- Introduction to dynamic programming
- Policy Evaluation
- Policy Iteration
- Value Iteration
- Extensions

## What is dynamic programming

- DP refers to a collection of algorithms that simplify a complicated problem by breaking it down into simpler sub-problems in a recursive manner
    - Optimal substructure
    - Overlapping subproblems
- Classical DP algorithms are of limited utility in RL. Why?
    - Need perfect model
    - Large computational expense
- For now, assume finite MDP only. DP ideas can be applied to problems with continuous state and action spaces, exact solutions are possible only in special cases.

## Dynamic Programming

- For prediction:

  Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy $\pi$

  or: MRP $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$

  Output: value function $v_{\pi}$
- Or for control:

  Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

  Output: optimal value function $v_{*}$

  and: optimal policy $\pi_{*}$

## Outline

- Introduction to dynamic programming
- Policy Evaluation
- Policy Iteration
- Value Iteration
- Extensions

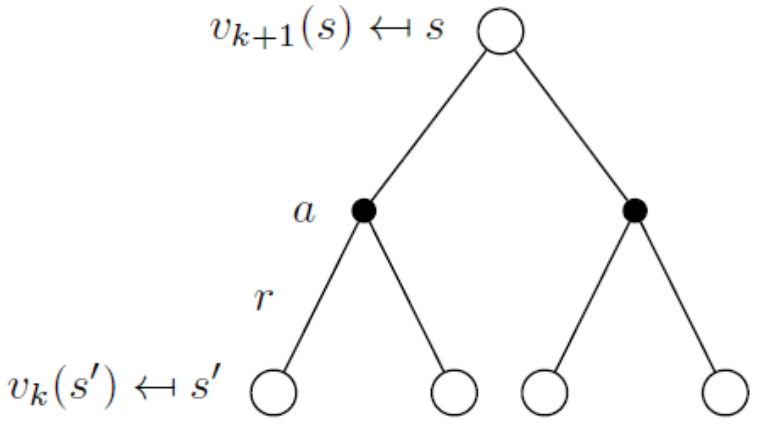
## Policy Evaluation

- Problem: evaluate a given policy $\pi$
- Solution: iterative application of Bellman expectation backup

  $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
- Using *synchronous* backups,

  At each iteration $k+1$
  For all states $s \in \mathcal{S}$
  Update $v_{k+1}(s)$ from $v_k(s')$
  where $s'$ is a successor state of $s$
- We will discuss *asynchronous* backups later

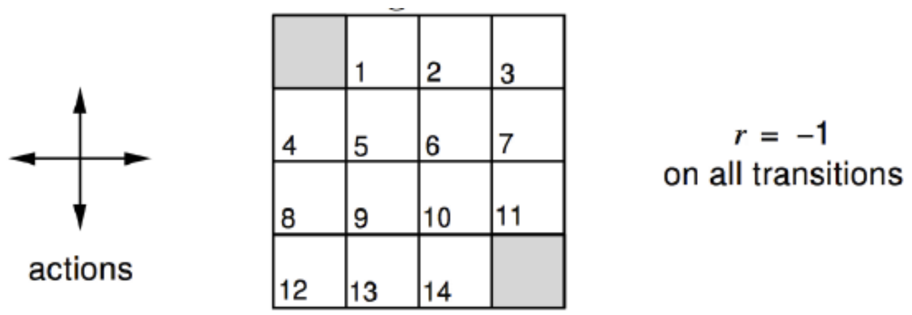
## Policy Evaluation



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

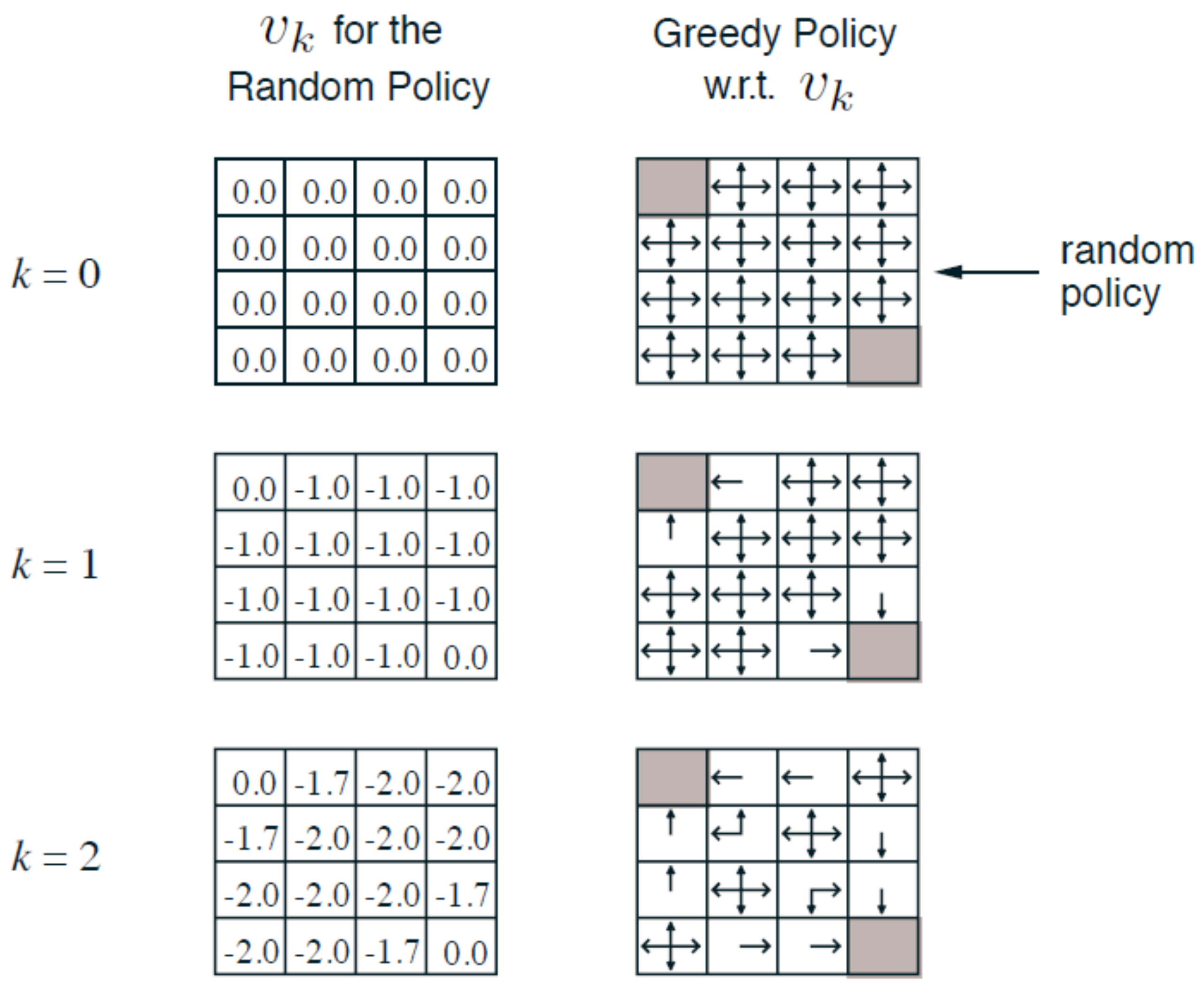
## Example: Small Grid world



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, ..., 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is $-1$ until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

# Example: Small Grid world



$k = 0$

| | | | |
|---|---|---|---|
| 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 |

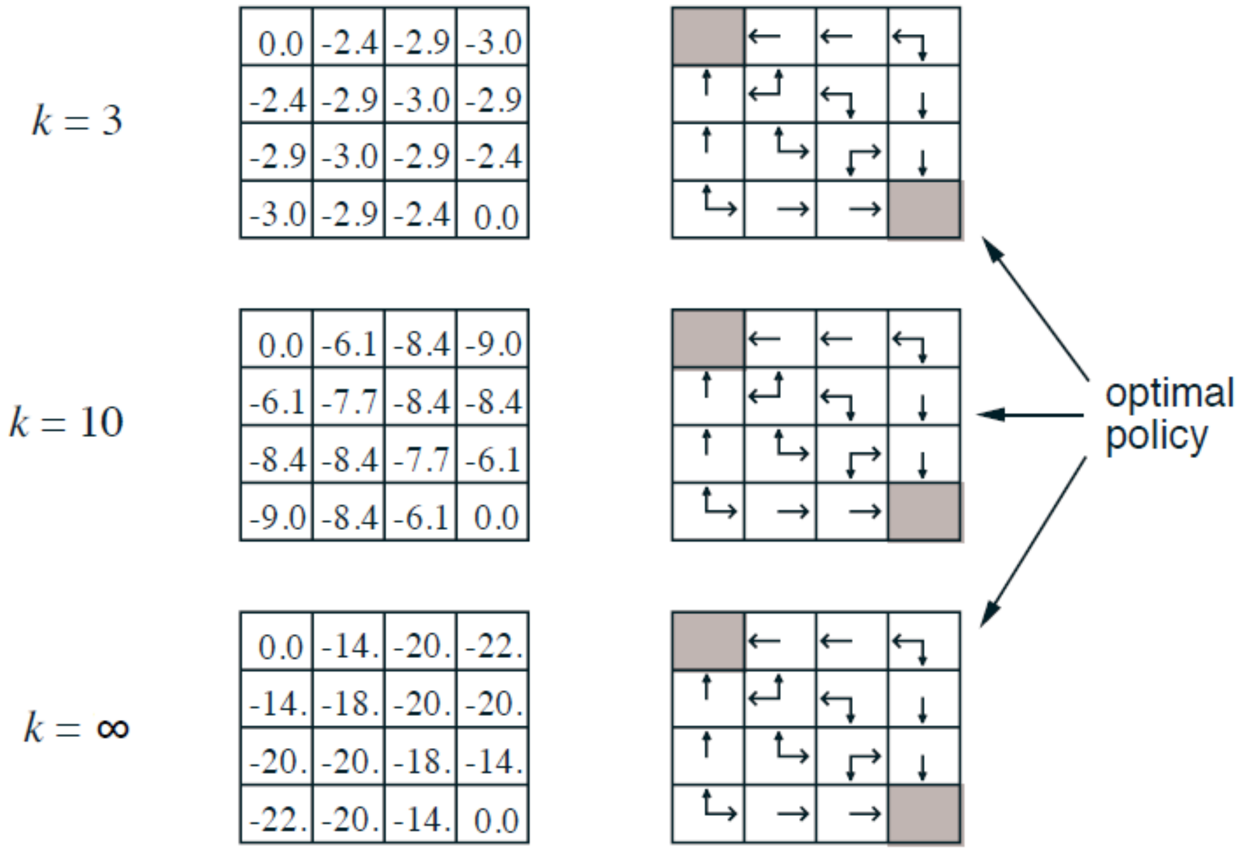
$k = 1$

| | | | |
|---|---|---|---|
| 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 |

# Example: Small Grid world



$k = 3$

| | | | |
|---|---|---|---|
| 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 |

## Outline

- Introduction to dynamic programming
- Policy Evaluation
- Policy Iteration
- Value Iteration
- Extensions

## How to improve a policy?

- Given a policy $\pi$
  - Evaluate the policy $\pi$

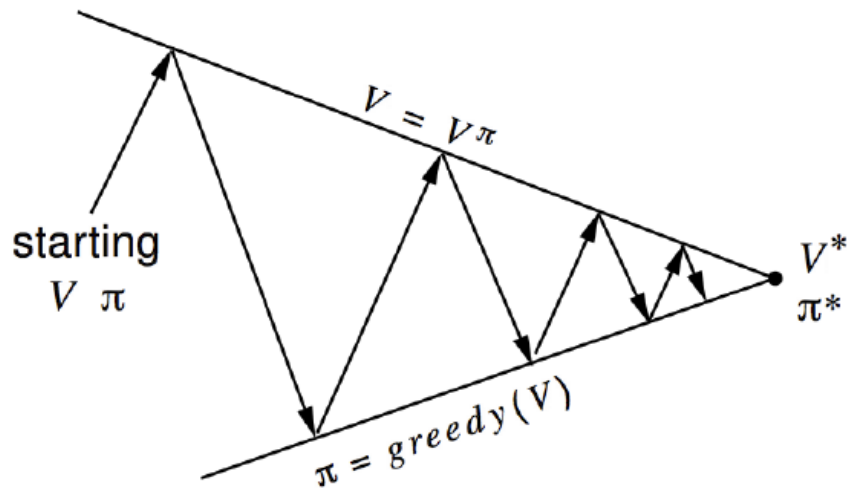
$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ...|S_t = s\right]$$

  - Improve the policy by acting greedily with respect to $v_\pi$

$$\pi' = \text{greedy}(v_\pi)$$

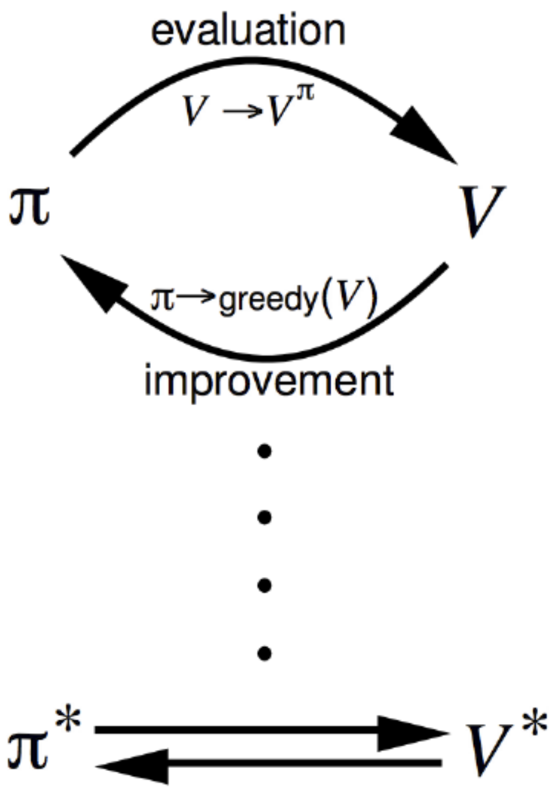
- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi*$

## Policy Iteration



Policy evaluation Estimate $v_{\pi}$
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement



## Policy Iteration - Algorithm

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

2. Policy Evaluation
   Repeat
       $\Delta \leftarrow 0$
       For each $s \in \mathcal{S}$:
           $temp \leftarrow v(s)$
           $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \Big[ r(s, \pi(s), s') + \gamma v(s') \Big]$
           $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $temp \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a) \Big[ r(s, a, s') + \gamma v(s') \Big]$
       If $temp \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $v$ and $\pi$; else go to 2
```

## Proof of Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \operatorname*{argmax}_{a \in \mathcal{A}} q_\pi(s, a)$$

- This improves the value from any state $s$ over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned}
v_\pi(s) \leq q_\pi(s, \pi'(s)) &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s)
\end{aligned}$$

## Proof of Policy Improvement

- If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

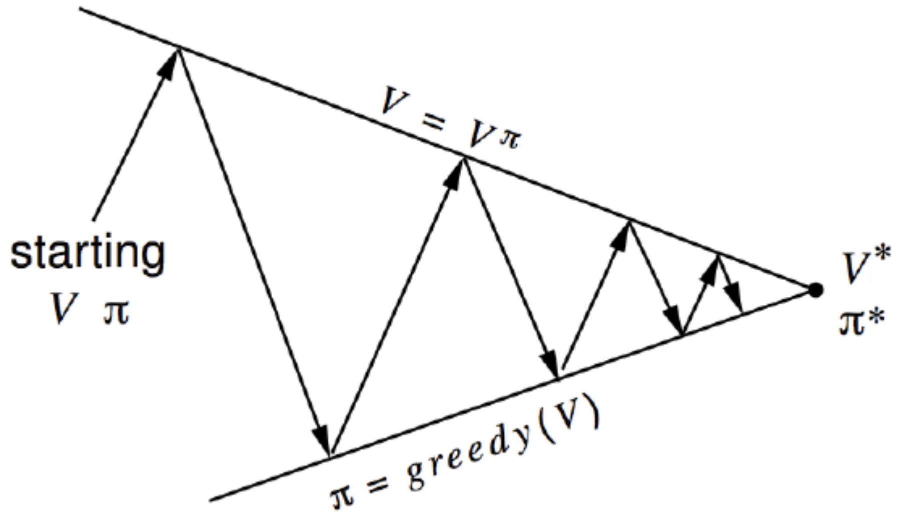
- Then the Bellman optimality equation has been satisfied

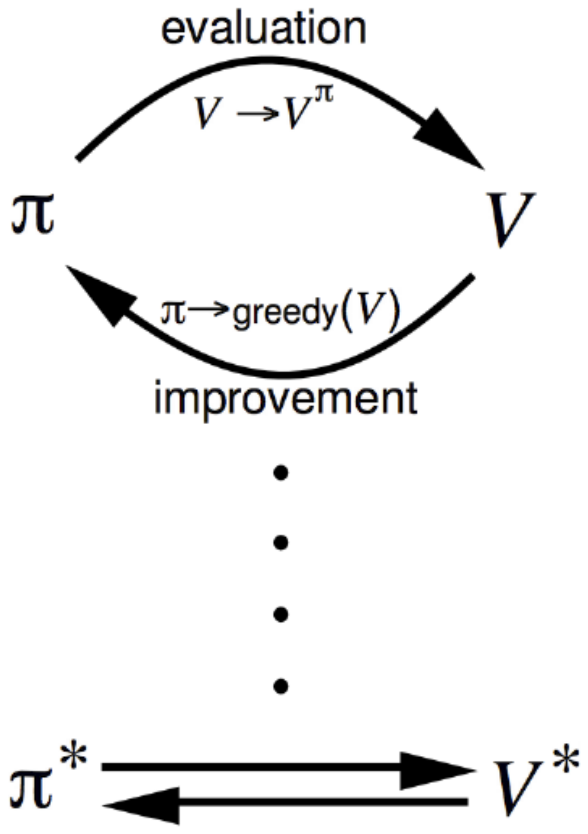
$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$

  so $\pi$ is an optimal policy

## Generalized Policy Iteration





Policy evaluation Estimate $v_\pi$

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

## Outline

- Introduction to dynamic programming
- Policy Evaluation
- Policy Iteration
- Value Iteration
- Extensions

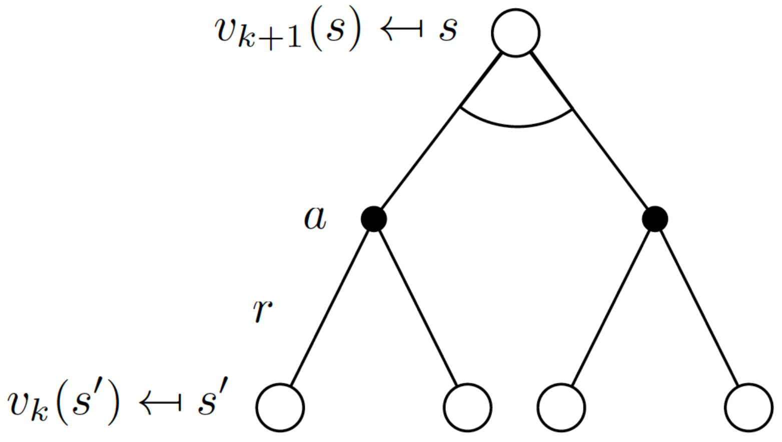
## Value Iteration

- Problem: find optimal policy $\pi$
- Solution: iterative application of Bellman optimality backup

  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Using synchronous backups

  At each iteration $k + 1$
  For all states $s \in \mathcal{S}$
  Update $v_{k+1}(s)$ from $v_k(s')$
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

## Value Iteration



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\mathbf{a}} + \gamma \boldsymbol{\mathcal{P}}^{\mathbf{a}} \mathbf{v}_k$$

## Value Iteration - Algorithm

```
Initialize array $v$ arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $temp \leftarrow v(s)$
        $v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
        $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
    $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$
```

## Example: Shortest Path

| | | | |
|---|---|---|---|
| g | | | |
| | | | |
| | | | |
| | | | |

Problem

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

$V_1$

| | | | |
|---|---|---|---|
| 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |

$V_2$

| | | | |
|---|---|---|---|
| 0 | -1 | -2 | -2 |
| -1 | -2 | -2 | -2 |
| -2 | -2 | -2 | -2 |
| -2 | -2 | -2 | -2 |

$V_3$

| | | | |
|---|---|---|---|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -3 |
| -2 | -3 | -3 | -3 |
| -3 | -3 | -3 | -3 |

$V_4$

| | | | |
|---|---|---|---|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -4 |
| -2 | -3 | -4 | -4 |
| -3 | -4 | -4 | -4 |

$V_5$

| | | | |
|---|---|---|---|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -4 |
| -2 | -3 | -4 | -5 |
| -3 | -4 | -5 | -5 |

$V_6$

| | | | |
|---|---|---|---|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -4 |
| -2 | -3 | -4 | -5 |
| -3 | -4 | -5 | -6 |

$V_7$

## Summary: Synchronous DP

| Problem | Bellman Equation | Algorithm |
|---|---|---|
| Prediction | Bellman Expectation Equation | Iterative Policy Evaluation |
| Control | Bellman Expectation Equation + Greedy Policy Improvement | Policy Iteration |
| Control | Bellman Optimality Equation | Value Iteration |

- Major drawback: involve operations over the entire state set of the MDP
- The game of backgammon has over 10^20 states
- Asynchronous DP can help!

## Outline

## Asynchronous DP

- Three simple ideas for asynchronous DP
    - In-place DP
    - Prioritized sweeping
    - Real-time dynamic programming
- To guarantee convergence, need continue to backup the values of all the states

## In-place DP

- Synchronous value iteration stores two copies of value function

  for all $s$ in $\mathcal{S}$

$$v_{new}(s) \leftarrow \max_{a\in\mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right)$$

  $v_{old} \leftarrow v_{new}$

- In-place value iteration only stores one copy of value function

  for all $s$ in $\mathcal{S}$

$$v(s) \leftarrow \max_{a\in\mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right)$$

## Prioritized Sweeping DP

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) - v(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

## Real-time DP

- Idea: only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step $S_t, A_t, R_{t+1}$
- Backup the state $S_t$

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}^a_{S_t} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{S_t s'} v(s') \right)$$