CS 4649/7649
Robot Intelligence: Planning

MDP solutions

Sungmoon Joo

School of Interactive Computing
College of Computing
Georgia Institute of Technology

*Slides based in part on Dr. Mike Stilman and Dr. Pieter Abbeel's slides

Administrative– Final Project

• CS7649
- project proposal: Due Oct. 30 (proposal outline: proposal_outline.pdf)
- project final report: Due Dec. 4, 23:59pm, conference-style paper
- project presentation: Dec. 11, 11:30am - 2:20pm

• CS4649
- project reviewer assignment: Oct. 28 (2 ~ 3 reviewers/project)
- proposal review report: Due Nov. 6
- project review report(for the assigned project): Due Dec. 11, 11:30am
- project presentation review*(for all presentation): Due Dec. 11, 2:20pm

*presentation review sheets will be provided

Probability

Axioms

(1) $0 \leq P(A) \leq 1$ (2) $P(True) = 1$ (3) $P(False) = 0$

(4) $P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$

(5) $P(A = v_i \wedge A = v_j) = 0$ if $i \neq j$

(6) $P(A = v_1 \vee A = v_2 \vee \ldots \vee A = v_k) = 1$

Theorems

$P(\text{not } A) = P(\neg A) = 1 - P(A)$

$P(A) = P(A \wedge B) + P(A \wedge \neg B)$

$$P(A = v_1 \vee A = v_2 \vee \ldots \vee A = v_i) = \sum_{j=1}^{i} P(A = v_j)$$

$$P(B \wedge (A = v_1 \vee A = v_2 \vee \ldots \vee A = v_i)) = \sum_{j=1}^{i} P(B \wedge A = v_j)$$

Conditional Probability

- Conditional Probability (Definition)

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

- Corollary

$$P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$$

- Bayes Rule

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

Markov Property

- "Markov" generally means that given the present state, the future and the past are independent
- For MP, "Markov" means

$$p(x_{k+1}|x_0, \ldots, x_k) = p(x_{k+1}|x_k), \forall k \in 0, 1, \ldots, N$$

- For MDP, "Markov" means

$$p(x_{k+1}|x_0, a_0 \ldots, x_k, a_k) = p(x_{k+1}|x_k, a_k), \forall k \in 0, 1, \ldots, N$$

Solving MDP

- In deterministic single-agent search problems, we want an optimal plan, or sequence of actions, from start to a goal
- For MDP, we want an optimal policy

$$\pi^* : S \rightarrow A, \ where \ S = \text{set of states}, \ A = \text{set of actions}$$

  - A policy gives an action for each state
  - An optimal policy maximizes expected utility(e.g. sum of rewards), if followed

- Two ways to define stationary utilities
  - Additive utility

$$U([r_0, r_1, \ldots]) = r_0 + r_1 + r_2 + \ldots$$

  - Discounted utility

$$U([r_0, r_1, \ldots]) = r_0 + \lambda r_1 + \lambda^2 r_2 + \ldots$$

Solving MDP

- Problem: infinite state sequences have infinite rewards

- Solutions

  - Finite horizon: Terminate episodes after a fixed horizon of T steps, yielding non-stationary policies (policies depend on time)

  - Discounting: for $0 < \lambda < 1$

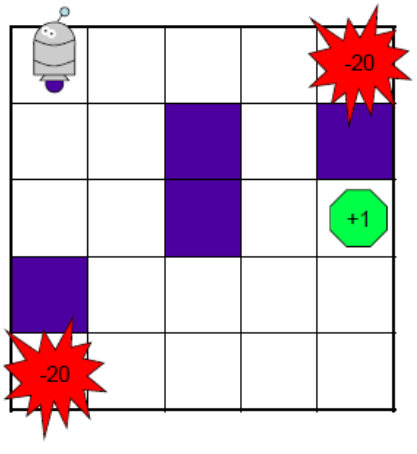
$$U([r_0, r_1, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \lambda^t r_t \leq R_{max}/(1-\lambda)$$

  - …

- We've discussed the case with the infinite horizon & discounting factor

  - the optimal policy is stationary (the same as at all times)

Robots with Uncertain Actions



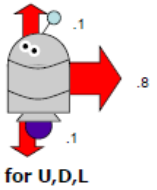
States: $(\{x\}, \{y\})$

Actions: Up, Down, Left, Right

Rewards: 0, -20, +1

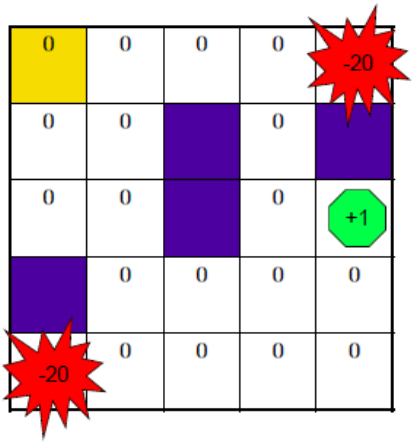
Transition Model:

$$p^R_{(x,y)(x+1,y)} = .8$$
$$p^R_{(x,y)(x,y+1)} = .1$$
$$p^R_{(x,y)(x,y-1)} = .1$$

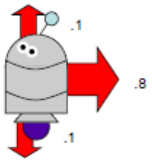


Similarly for U,D,L

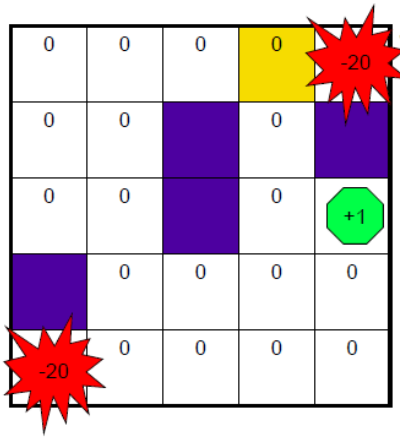
Value Iteration



Any action gets 0 reward



Value Iteration



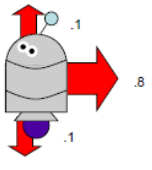
$J_s^R = 0 + .9(.8 * (-20) + .1 * 0) = -14.4$

$J_s^D = 0 + .9(.8 * 0 + .1 * (-20)) = -1.8$

$J_s^L = 0 + .9(.8 * 0 + .1 * 0) = 0$

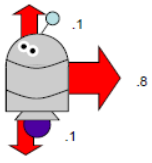
$J_s^U = 0 + .9(.8 * 0 + .1 * (-20)) = -1.8$

$\pi_s = \text{argmax}_l \left[\sum_{j=1}^{n} p_{sj}^l J_j^*\right]$

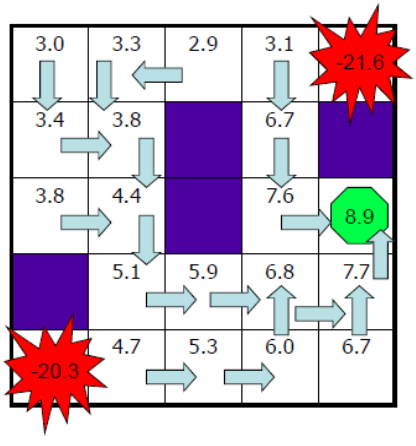


Value Iteration

Iteration 100

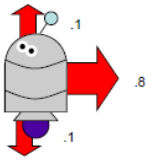


Value Iteration



Optimal Policy

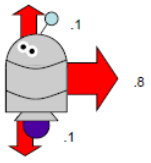
$$\pi_s = \text{argmax}_l \left[\sum_{j=1}^{n} p_{sj}^{l} J_j^*\right]$$



Value Iteration

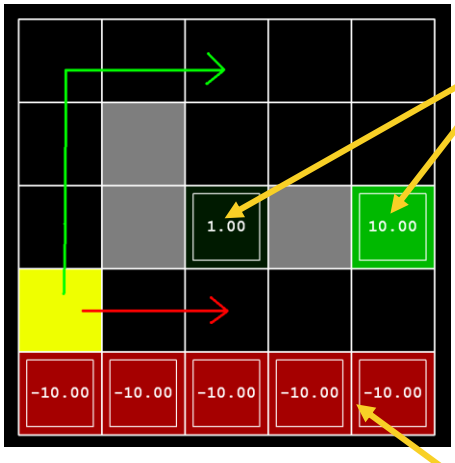
Optimal Value Function

$$J_s^*$$



Effect of Discount & Uncertainty in MDP

Parameters:
- **Discount**
- **Uncertainty/Noise**

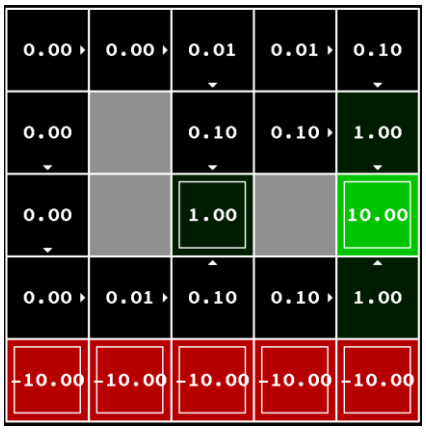


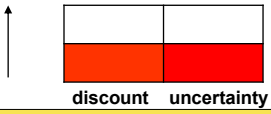
+ Rewards

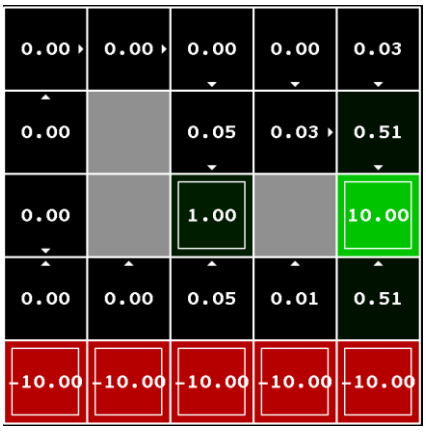
- Rewards (e.g. cliff)

Example from: http://www.cs.berkeley.edu/~pabbeel/cs287-fa13/

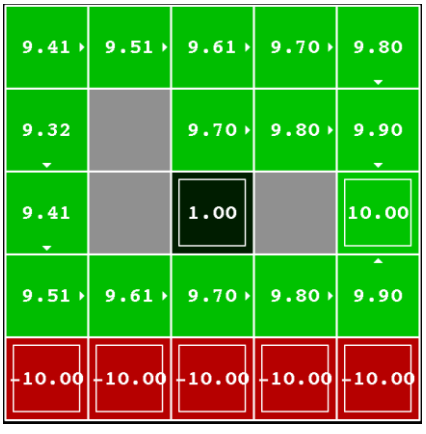
Effect of Discount & Uncertainty in MDP

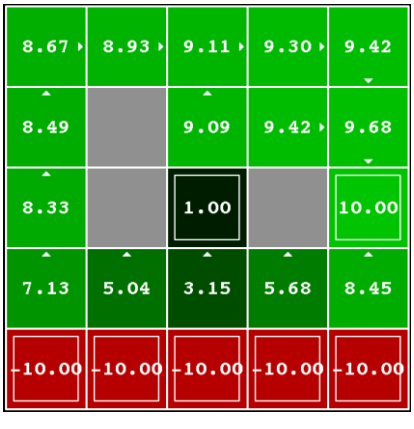


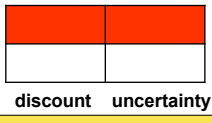




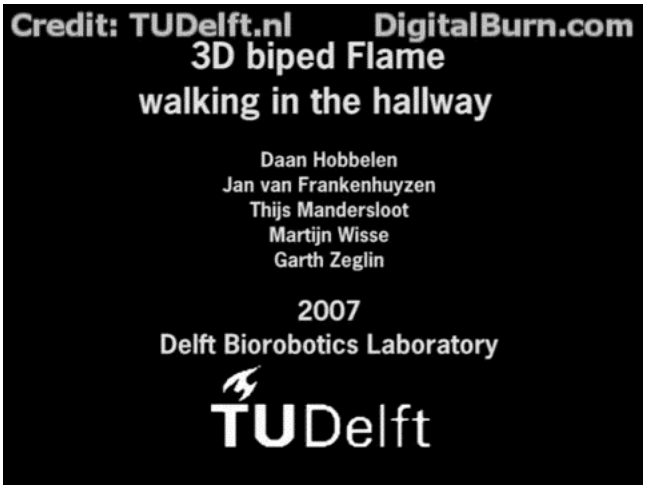
Effect of Discount & Uncertainty in MDP



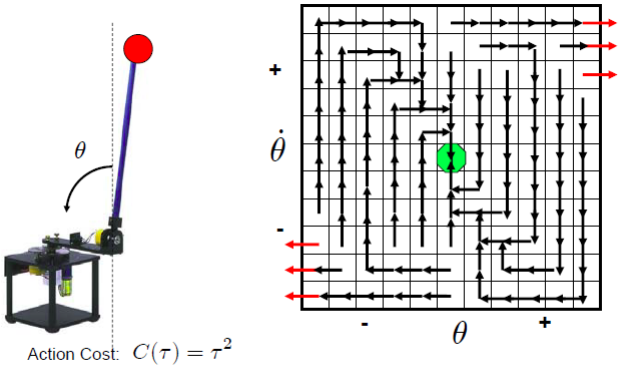




MDP Applications: Passive Dynamic Walking



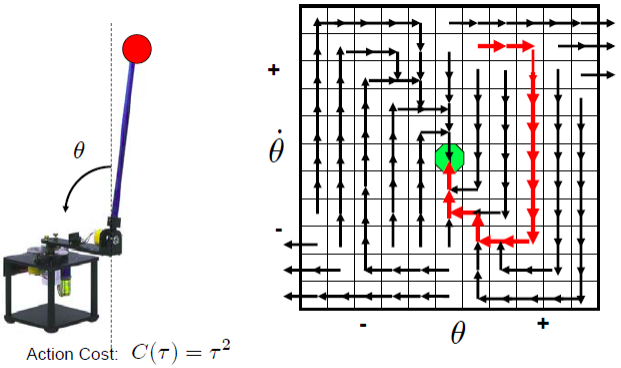
MDP Applications: Inverted Pendulum



Action Cost: $C(\tau) = \tau^2$

Actions: τ^a System Dynamics: $\tau^s = mgl \sin\theta$

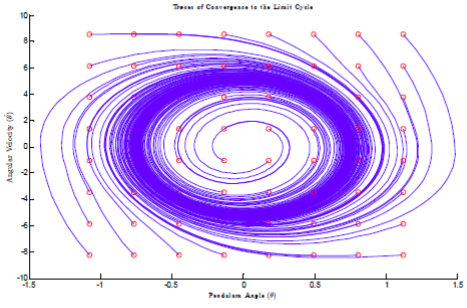
MDP Applications: Inverted Pendulum



Action Cost: $C(\tau) = \tau^2$

Actions: τ^a System Dynamics: $\tau^s = mgl \sin\theta$

MDP Applications: Inverted Pendulum



Improving Efficiency

Value Iteration

- Sparse Matrices
- No Initial Policy
- Few Actions

Policy Iteration

- Given a policy guess
- Many Actions

Policy Evaluation

- Our goal is to find a policy, not just computing values.
- In value iteration, we compute values then find a policy
- How do we calculate the V's for a fixed policy?
  - use Bellman's equation

$$V_0(s) = 0$$

$$V_{k+1}(s) \leftarrow \sum_{s'} P(s'|s, \pi(s))[r_{s'} + \lambda V_k(s)]$$

  - ...

Policy Iteration

- Policy Iteration
  - Step 1. Policy evaluation
    : calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2. Policy improvement
    : update policy using one step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- Properties
  - It's still optimal!
  - Can converge faster under some conditions

Policy Iteration

- Policy Evaluation
  - with a fixed current policy π_i, find values with Bellman updates
  - iterate until values converges

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s, \pi_i(s))[r_{s'} + \lambda V_k^{\pi_i}(s')]$$

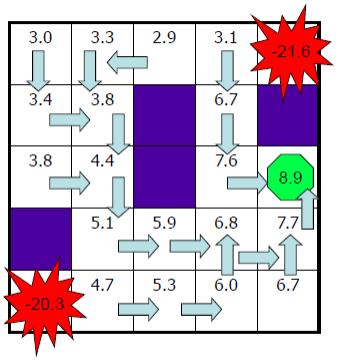
- Policy Improvement
  - with fixed(converged) values, find the best action

$$\pi_{i+1}(s) = \operatorname*{argmin}_a \sum_{s'} P(s'|s, a)[r_{s'} + \lambda V_k^{\pi_i}(s')]$$

- Theorem

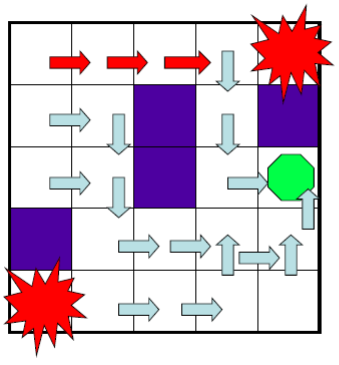
Policy iteration is guaranteed to converge and at convergence, the current policy and its value function are the optimal policy and the optimal value function.

Policy Iteration



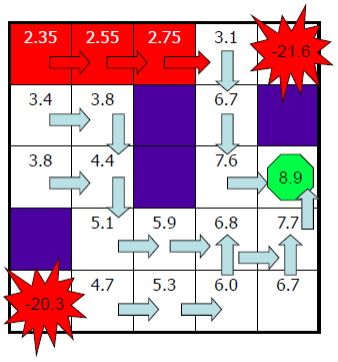
Optimal Policy

Policy Iteration



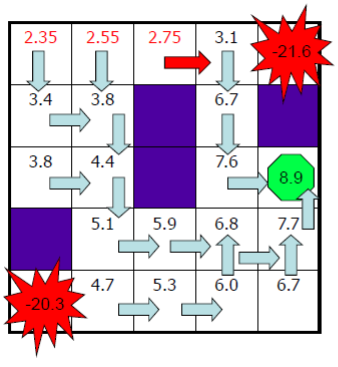
- Modify three actions (new policy)

Policy Iteration



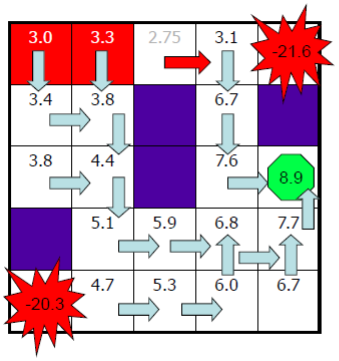
- Modify three actions (new policy)
- Compute Value Function

Policy Iteration



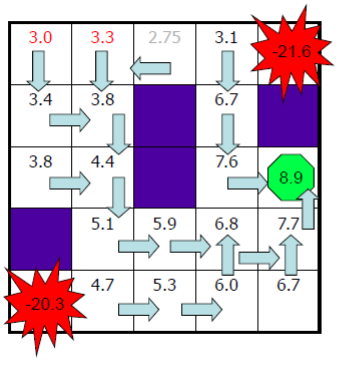
- Modify three actions (Policy 1)
- Compute Value Function
- Compute Policy 2

Policy Iteration



- Modify three actions (Policy 1)
- Compute Value Function
- Compute Policy 2
- Compute Value Function

Policy Iteration



- Modify three actions (Policy 1)
- Compute Value Function
- Compute Policy 2
- Compute Value Function
- Compute Policy 3
- **Optimal Policy**

Comparison

- In value iteration:

- Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)

Let J_s^k be the highest expected sum of discounted rewards from **s** in k steps

- Highest expected value in 1 step: $J_s^1 = r_s$
- Highest expected value in 2 steps: $J_s^2 = \max_l \left[r_s + \lambda \sum_{j=1}^n p_{sj}^l J_j^1 \right]$
- Highest expected value in t steps: $J_s^k = \max_l \left[r_s + \lambda \sum_{j=1}^n p_{sj}^l J_j^{k-1} \right]$

- Bellman's Equation

- Bellman Update

Comparison

- In policy iteration:

- Several passes to update utilities with frozen policy

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s, \pi_i(s))[r_{s'} + \lambda V_k^{\pi_i}(s')]$$

- Occasional passes to update policies

$$\pi_{i+1}(s) = \operatorname*{argmin}_a \sum_{s'} P(s'|s, a)[r_{s'} + \lambda V_k^{\pi_i}(s')]$$

Asynchronous Iterations

- Asynchronous policy iteration(we saw this)

  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often

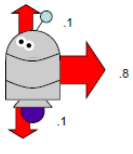
- Asynchronous value iteration

  - In value iteration, we update every state in each iteration
  - Actually, any sequences of Bellman updates will converge if every state is visited infinitely often
  - In fact, we can update the policy as seldom or often as we like, and we will still converge
  - Idea : Update states whose value we expect to change
    If $|V_{k+1}(s) - V_k(s)|$ is large then update predecessors of s

Model availability: P(j|i,a)

What do we do if such a model does not exist?

- **Make one**
- **Q-Learning**



$J^*(s)$ – Value of a state

$\pi^*(s)$ – Optimal policy in the state

$Q^*(S_i,a)$ - Value of taking an action in a state

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

Compute Q iteratively:

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$

α = .7

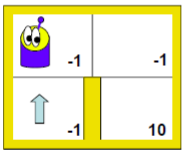
	⇧	⇩	⇦	⇨
$\mathbf{S_1}$	0	0	0	0
$\mathbf{S_2}$	0	0	0	0
$\mathbf{S_3}$	0	0	0	0
$\mathbf{S_4}$	0	0	0	0

Q-Table

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$



$Q^{est}(S_1, ⇧) =$
$.7(-1 + .9 \max(0, 0, 0, 0)) + .3 \times 0$

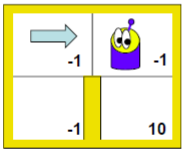
	⇧	⇩	⇦	⇨
$\mathbf{S_1}$	-.7	0	0	0
$\mathbf{S_2}$	0	0	0	0
$\mathbf{S_3}$	0	0	0	0
$\mathbf{S_4}$	0	0	0	0

Q-Table

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$



$Q^{est}(S_2, \Rightarrow) = .7(-1 + .9 \max(0, 0, 0, 0)) + .3 \times 0$

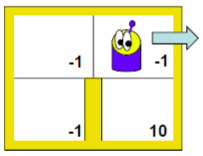
	⇑	⇓	⇐	⇒
$\mathbf{S_1}$	-.7	0	0	0
$\mathbf{S_2}$	0	0	0	-.7
$\mathbf{S_3}$	0	0	0	0
$\mathbf{S_4}$	0	0	0	0

Q-Table

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$



$Q^{est}(S_3, \Rightarrow) = .7(-1 + .9 \max(0, 0, 0, 0)) + .3 \times 0$

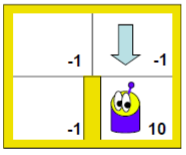
	⇑	⇓	⇐	⇒
$\mathbf{S_1}$	-.7	0	0	0
$\mathbf{S_2}$	0	0	0	-.7
$\mathbf{S_3}$	0	0	0	-.7
$\mathbf{S_4}$	0	0	0	0

Q-Table

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$



$Q^{est}(S_3, \Downarrow) =$
$.7(-1 + .9 \max(0, 0, 0, 0)) + .3 \times 0$

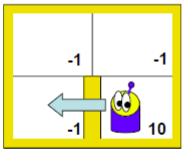
	⇑	⇓	⇐	⇒
$\mathbf{S_1}$	-.7	0	0	0
$\mathbf{S_2}$	0	0	0	-.7
$\mathbf{S_3}$	0	-.7	0	-.7
$\mathbf{S_4}$	0	0	0	0

Q-Table

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$



$Q^{est}(S_4, \Leftarrow) =$
$.7(10 + .9 \max(0, 0, 0, 0)) + .3 \times 0$

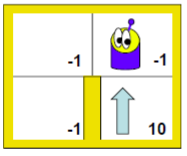
	⇑	⇓	⇐	⇒
$\mathbf{S_1}$	-.7	0	0	0
$\mathbf{S_2}$	0	0	0	-.7
$\mathbf{S_3}$	0	-.7	0	-.7
$\mathbf{S_4}$	0	0	7	0

Q-Table

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$



$Q^{est}(S_4, \uparrow) =$
$.7(10 + .9 \max(0, -.7, 0, -.7)) + .3 \times 0$

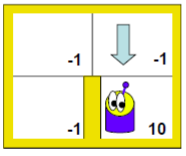
	↑	↓	←	→
$\mathbf{S_1}$	-.7	0	0	0
$\mathbf{S_2}$	0	0	0	-.7
$\mathbf{S_3}$	0	-.7	0	-.7
$\mathbf{S_4}$	7	0	7	0

Q-Table

Q-Learning

$Q(S_i, a)$ = Value of Taking action **a** in state $\mathbf{S_i}$

$$Q^{t+1}(S_i, a) \leftarrow \alpha(r_i + \gamma \max_b Q^t(S_j, b)) + (1 - \alpha)Q^t(S_i, a)$$



$Q^{est}(S_3, \downarrow) =$
$.7(-1 + .9 \max(7,0,7,0)) + .3 \times -.7$

	↑	↓	←	→
$\mathbf{S_1}$	-.7	0	0	0
$\mathbf{S_2}$	0	0	0	-.7
$\mathbf{S_3}$	0	3.5	0	-.7
$\mathbf{S_4}$	7	0	7	0

Q-Table

Summary

MDPs = Useful Representation of Uncertainty in Action

Matrix Inversion
- Few States

Value Iteration
- Sparse Matrices
- No Initial Policy
- Few Actions

Policy Iteration
- Given a policy guess
- Many Actions

RL

Q-Learning
- No model required
- Somewhat inefficient
- Not parameterized

MDP(VI, PI, RL) → Control, Estimation → POMDP