L8a Markov Decision Processes:
Reactive Planning to Maximize Reward

Brian C. Williams

16.410 / 13

October 5th, 2015

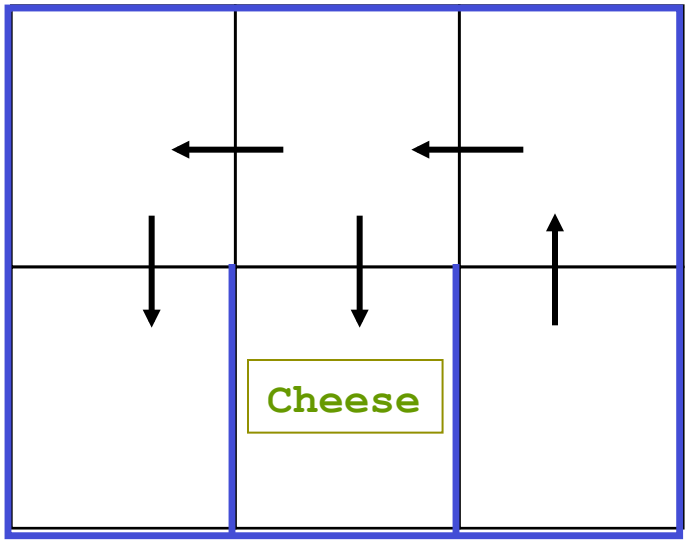
Slides adapted from:
Manuela Veloso,
Reid Simmons, &
Tom Mitchell, CMU

Assignments

- Reading:
  - Today: Markov Decision Processes: AIMA 17.1-3.
  - Wednesday: Hidden Markov Models: AIMA 15.1-3 and 20.1; Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," on Stellar.
- Remember:
  - Problem Set #4, due this Wednesday
- This lecture based on development in:

  "Machine Learning" by Tom Mitchell
  Chapter 13: Reinforcement Learning

How Might a Mouse Search a Maze for Cheese?



- State Space Search?
- Goal-directed Planning?
- As a Constraint Satisfaction Problem?

What is missing?

Ideas in this lecture

- Problem is to accumulate rewards, rather than to achieve goal states.
- Approach is to generate reactive policies for how to act in all situations, rather than plans for a single starting situation.
- Policies fall out of value functions, which describe the greatest lifetime reward achievable at every state.
- Value functions are iteratively approximated.

MDP Examples: TD-Gammon [Tesauro, 1995]
Learning Through Reinforcement

Learns to play Backgammon

States:

- Board configurations (10^{20})

Actions:

- Moves

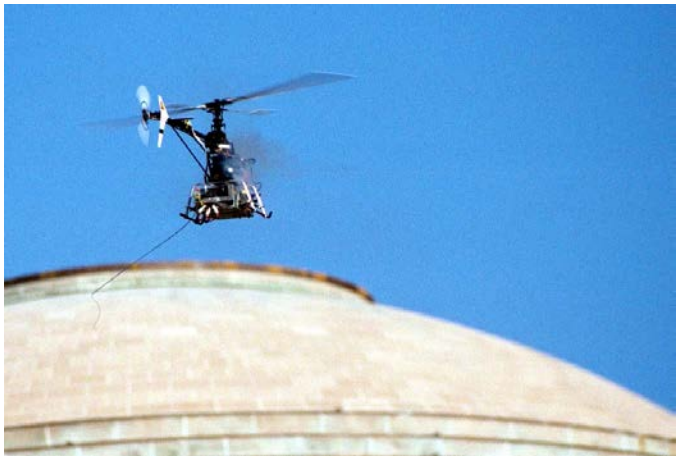
Rewards:

- +100 if win
- - 100 if lose
- 0 for all other states

- Trained by playing 1.5 million games against self.

➔ Became roughly equal to best human player.

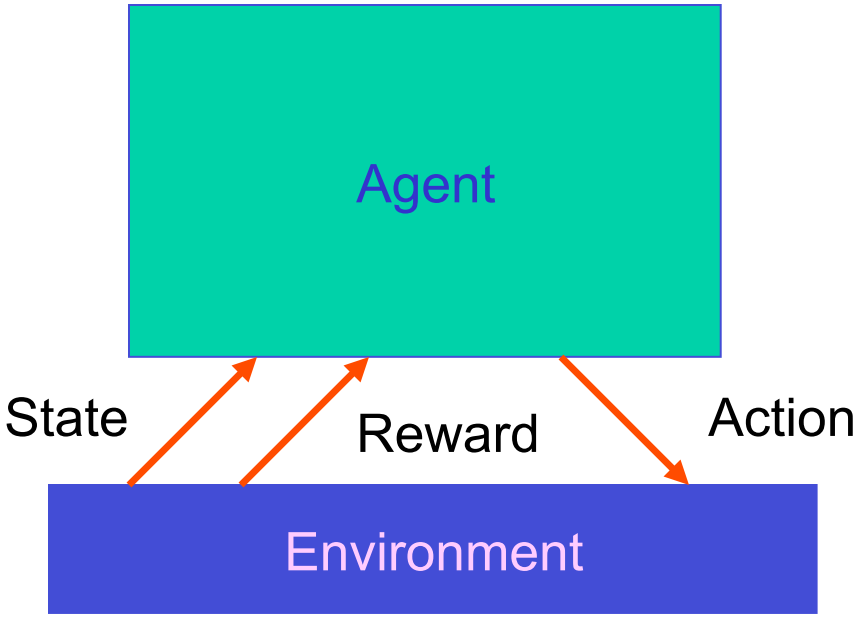
MDP Examples: Aerial Robotics [Frazzoli & Feron]
Computing a Solution from a Continuous Model

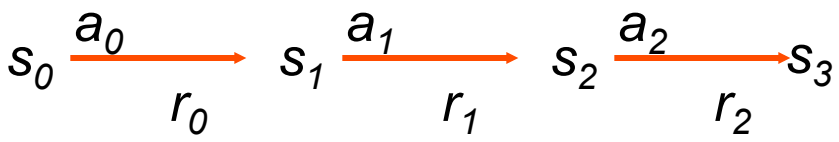


Markov Decision Processes

- Motivation
- What are Markov Decision Processes (MDPs)?
  - Models
  - Lifetime Reward
  - Policies
- Computing Policies From a Model
- Summary

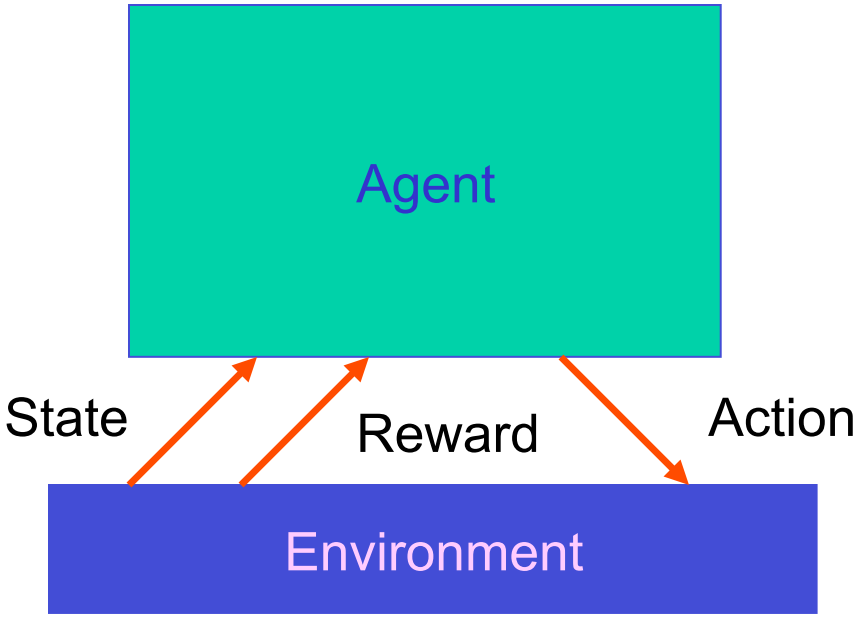
MDP Problem





Given an environment model as a MDP, create a policy for acting that maximizes lifetime reward.

MDP Problem: Model



$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3$

Given an environment model as a MDP, create a policy for acting that maximizes lifetime reward.

Markov Decision Processes (MDPs)

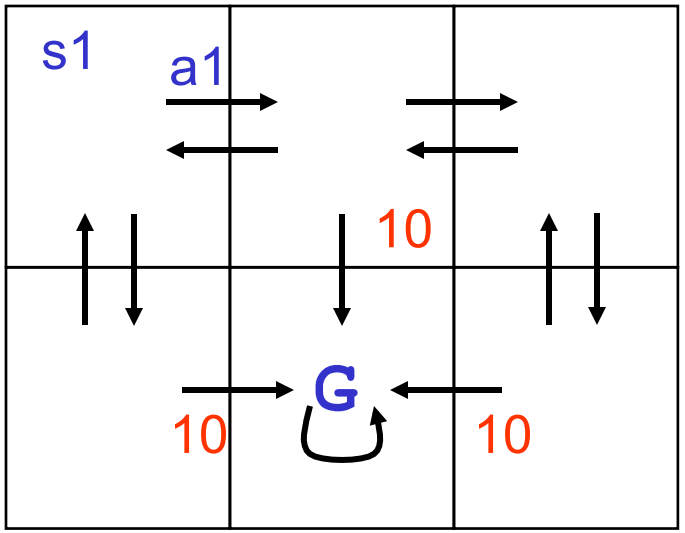
Model:

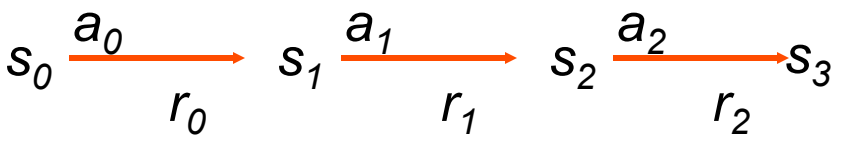
- Finite set of states, S
- Finite set of actions, A
- (Probabilistic) state transitions, $\delta(s,a)$
- Reward for each state and action, $R(s,a)$

Process:

1. Observe state s_t in S.
2. Choose action a_t in A.
3. Receive immediate reward r_t.
4. State changes to s_{t+1}.

Example:





- Legal transitions shown.
- Reward on unlabeled transition is 0.

MDP Environment Assumptions

- Markov Assumption:
  Next state and reward is a function only of the current state and action:
  - $s_{t+1} = \delta(s_t, a_t)$
  - $r_t = r(s_t, a_t)$

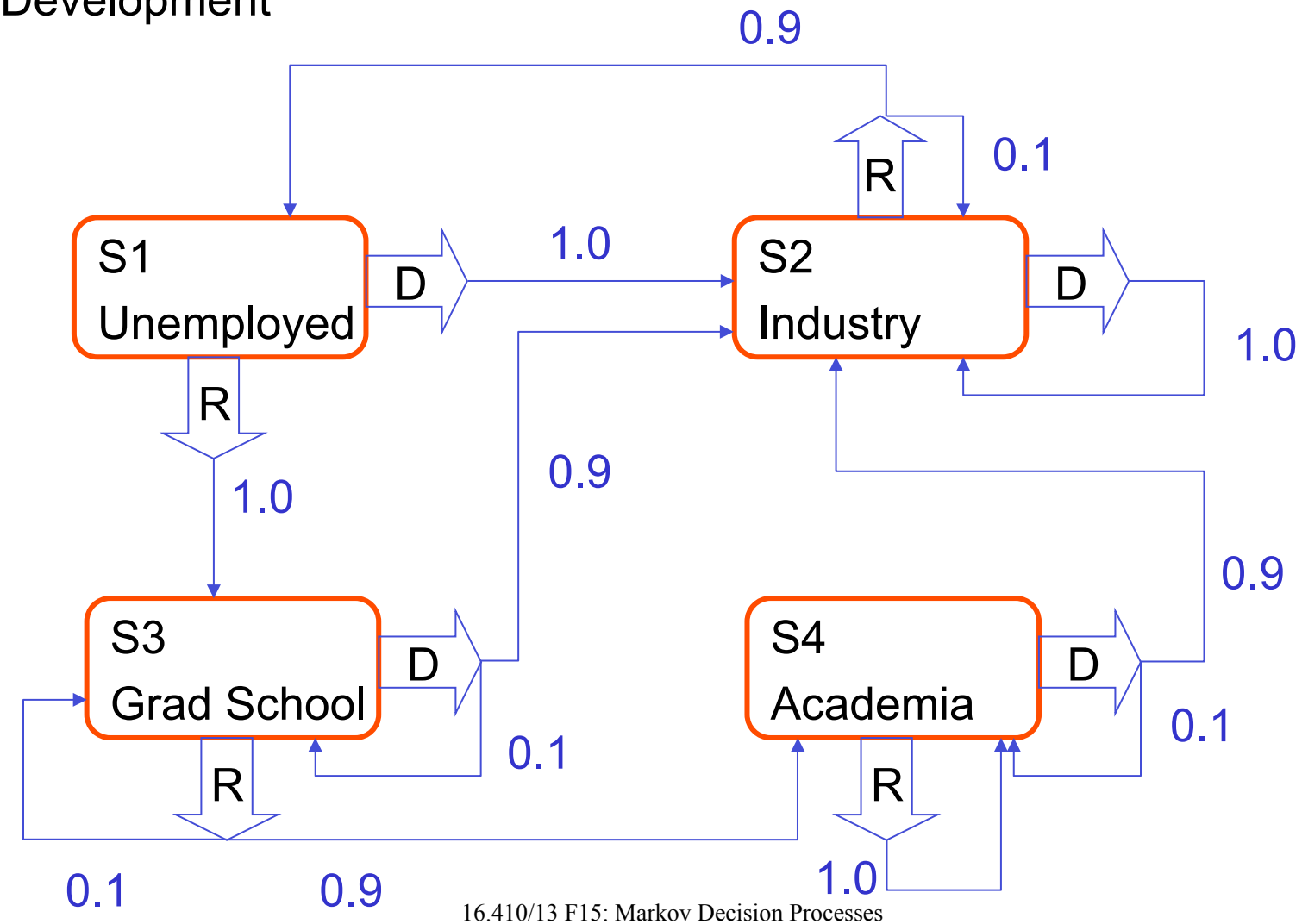
- Uncertain and Unknown Environment:
  δ and r may be nondeterministic and unknown.

MDP Nondeterministic Example

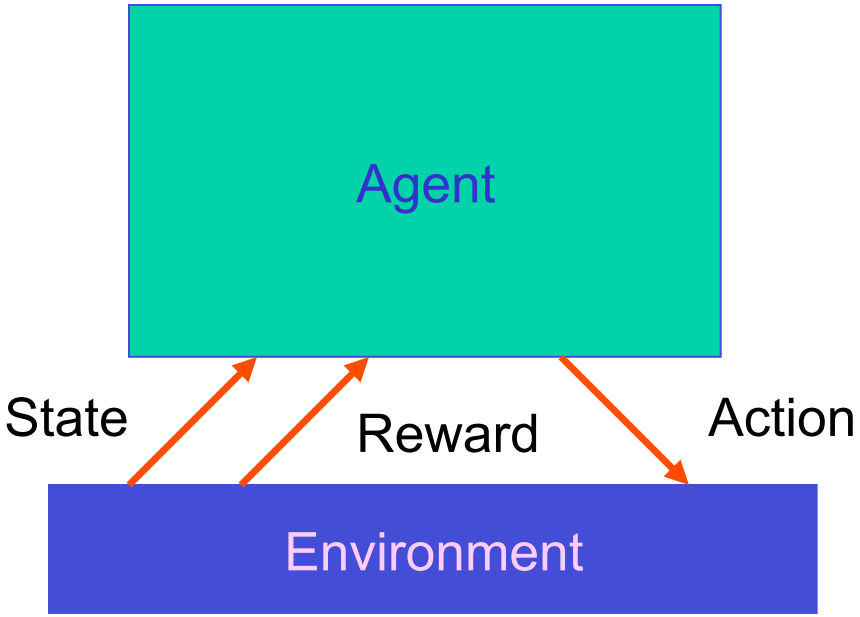
R – Research

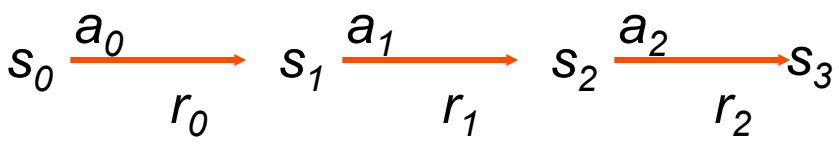
D – Development

Today we only consider the deterministic case



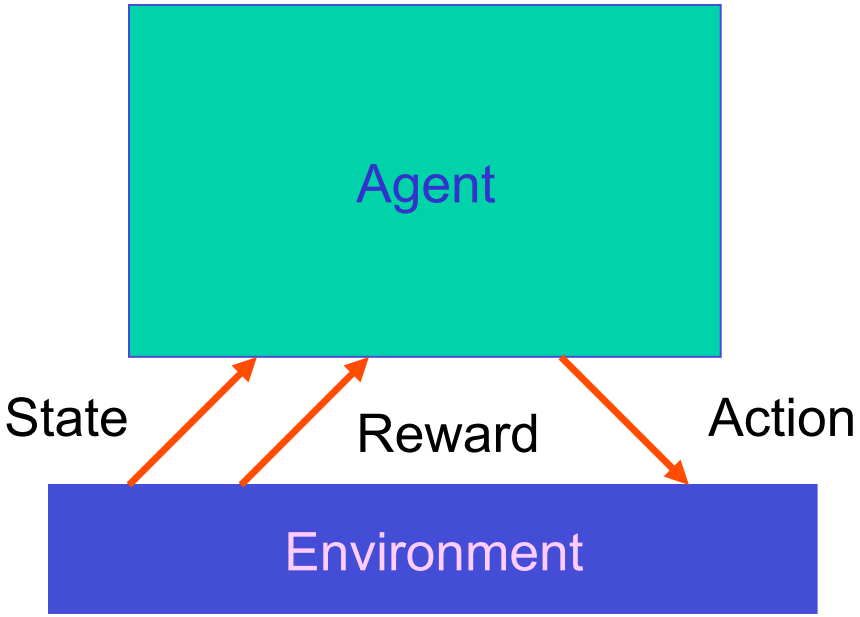
MDP Problem: Model





Given an environment model as a MDP, create a policy for acting that maximizes lifetime reward.

MDP Problem: Lifetime Reward



$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3$

Given an environment model as a MDP, create a policy for acting that maximizes lifetime reward.

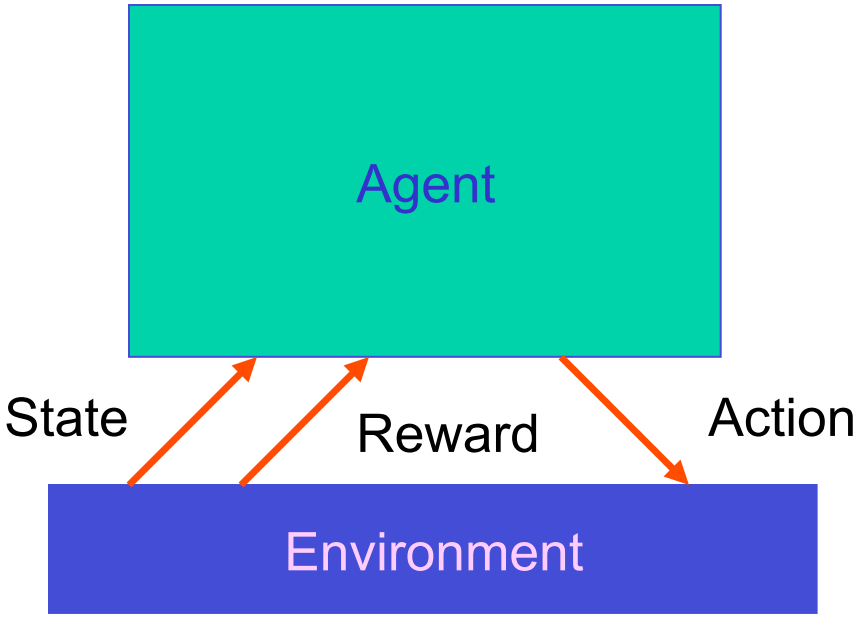
Lifetime Reward

- Finite horizon:
  - Rewards accumulate for a fixed period.
  - $100K + $100K + $100K = $300K

- Infinite horizon:
  - Assume reward accumulates for ever.
  - $100K + $100K + . . . = infinity

- Discounting:
  - Future rewards are not worth as much (a bird in hand …).
  - Introduce discount factor γ
    $100K + γ $100K + γ^2 $100K. . . Converges.
  - Will make the math work.

MDP Problem: Lifetime Reward

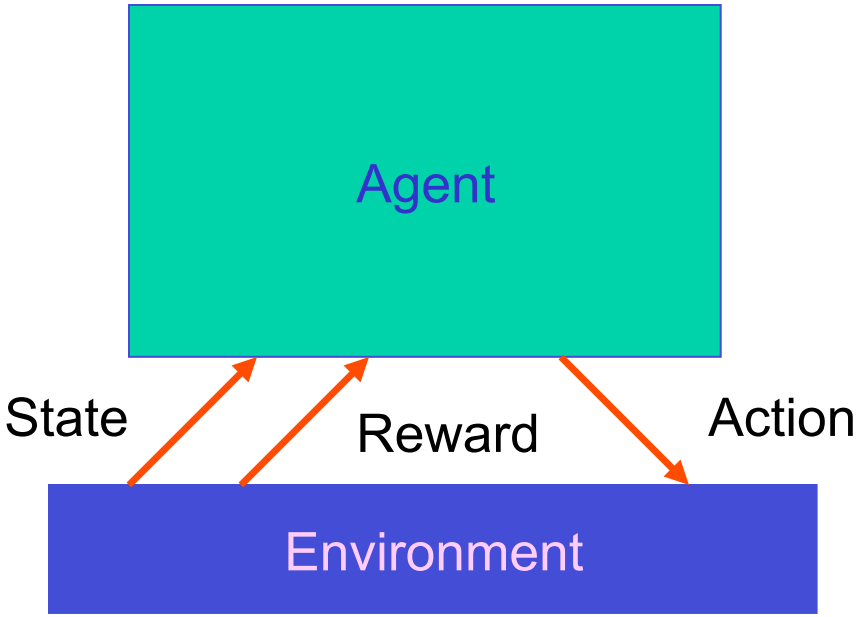


$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3$$

Given an environment model as a MDP, create a policy for acting that maximizes lifetime reward.

$$V = r_0 + \gamma\, r_1 + \gamma^2\, r_2 \ldots$$

MDP Problem: Policy



$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3$

Given an environment model as a MDP, create a policy for acting that maximizes lifetime reward.

$$V = r_0 + \gamma\, r_1 + \gamma^2\, r_2 \ldots$$

Assume deterministic world

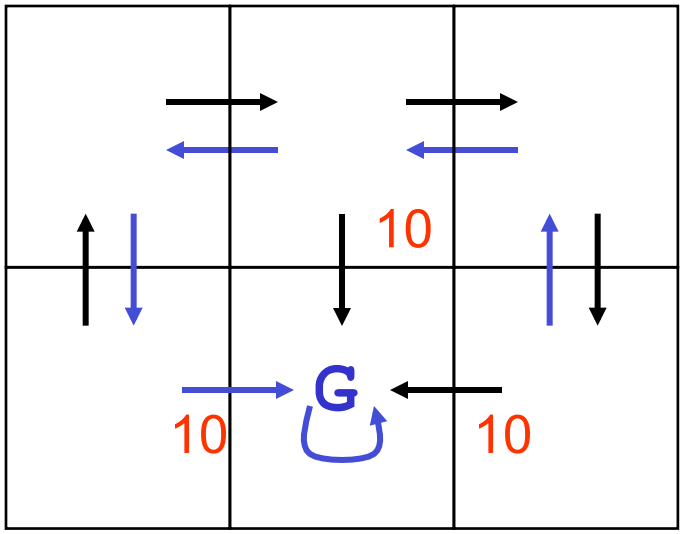
Policy π : $S \rightarrow A$

- Selects an action for each state.

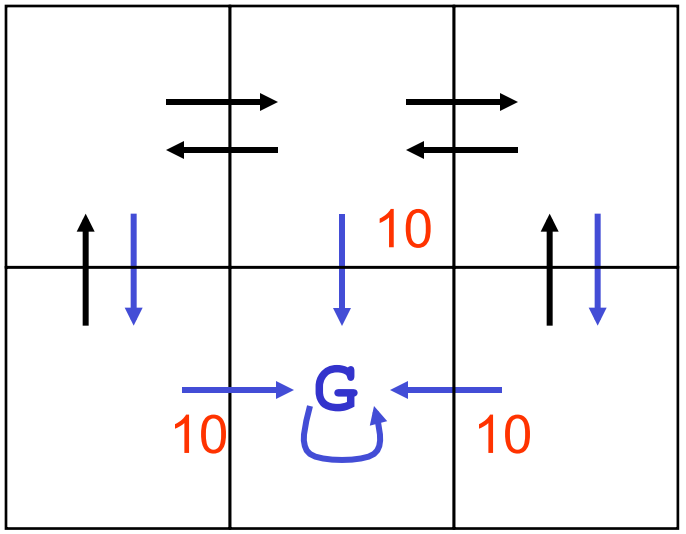
Optimal policy π^*: $S \rightarrow A$

- Selects action for each state that maximizes lifetime reward.

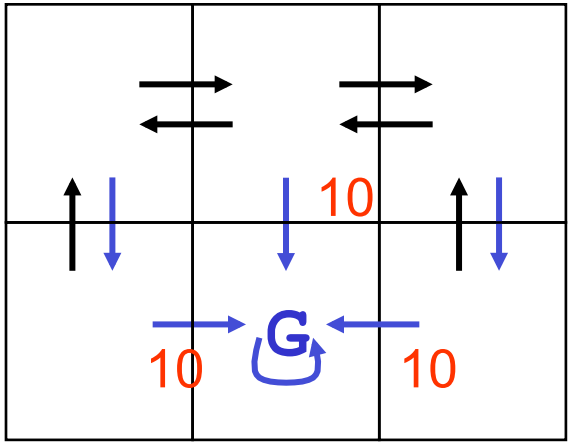
π

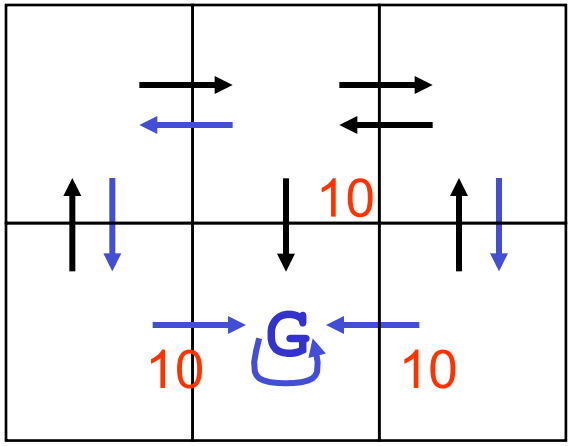


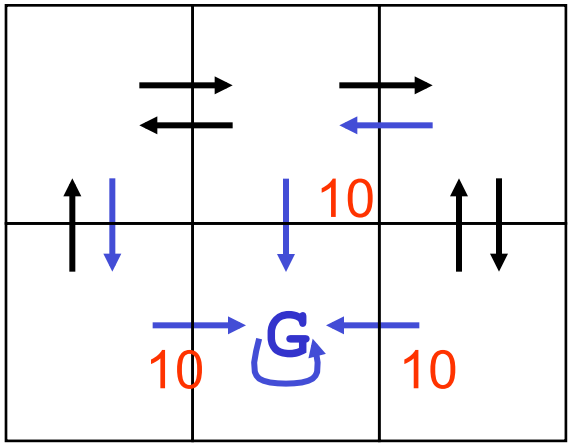
π^*



- There are many policies, not all are necessarily optimal.
- There may be several optimal policies.







Markov Decision Processes

- Motivation
- What are Markov Decision Processes (MDPs)?
  - Models
  - Lifetime Reward
  - Policies
- Computing Policies From a Model
- Summary

Markov Decision Processes

- Motivation
- Markov Decision Processes
- Computing Policies From a Model
  - Value Functions
  - Mapping Value Functions to Policies
  - Computing Value Functions through Value Iteration
  - An Alternative: Policy Iteration (appendix)
- Summary

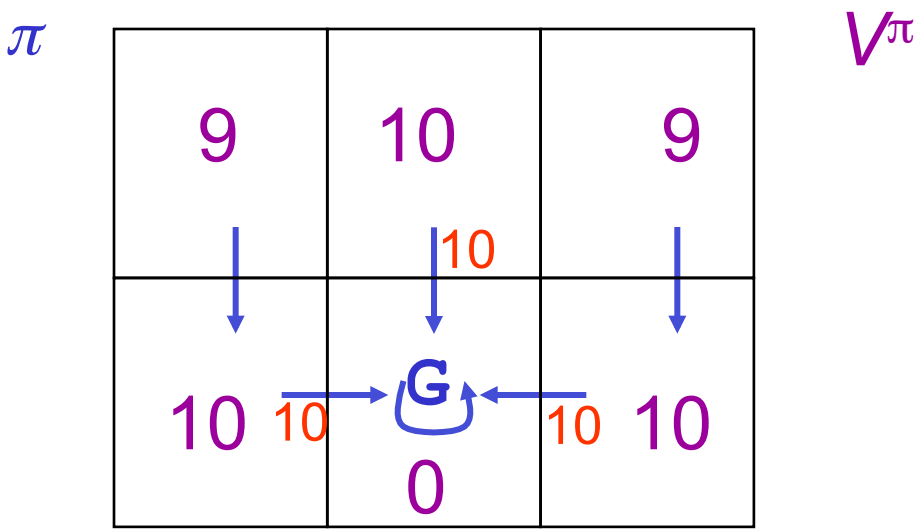
Value Function V^{π} for a Given Policy π

- $V^{\pi}(s_t)$ is the accumulated lifetime reward resulting from starting in state s_t and repeatedly executing policy π:

$$V^{\pi}(s_t) = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} \ldots$$
$$V^{\pi}(s_t) = \sum_i \gamma^i\, r_{t+i}$$

where r_t, r_{t+1}, r_{t+2} . . . are generated by following π, starting at s_t.

Assume $\gamma = .9$



An Optimal Policy π^* Given Value Function V^*

Idea: Given state s

1. Examine **all** possible actions a_i in state s.
2. Select action a_i with greatest lifetime reward.

Lifetime reward Q(s, a_i) is:

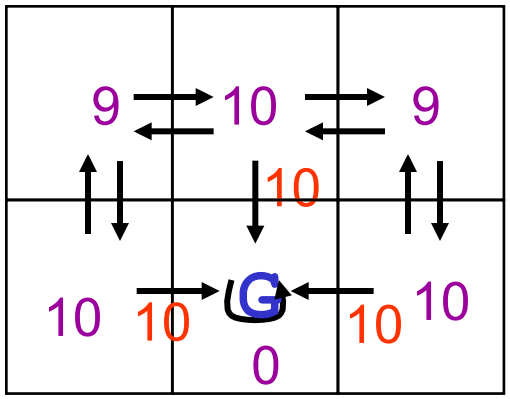
- the immediate reward for taking action $r(s,a)$ …
- plus life time reward starting in target state $V(\delta(s, a))$ …
- discounted by γ.

$$\pi^*(s) = \text{argmax}_a [r(s,a) + \gamma V^*(\delta(s, a))]$$

Must Know:

- Value function
- Environment model.
  - $\delta : S \times A \rightarrow S$
  - $r : S \times A \rightarrow \Re$

π

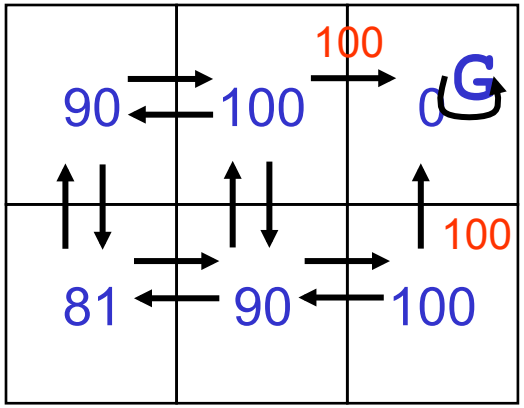


Example: Mapping Value Function to Policy

- Agent selects optimal action from *V*:

$$\pi(s) = \text{argmax}_a \left[r(s,a) + \gamma V(\delta(s, a)) \right]$$

Model + V:



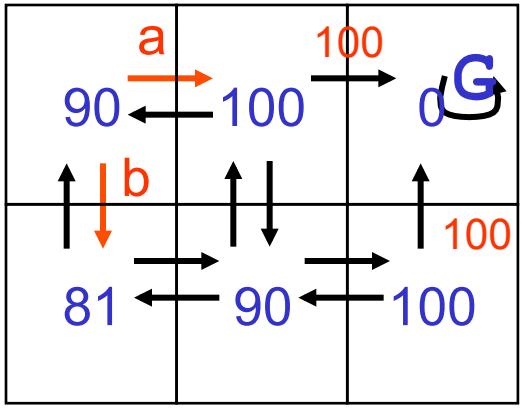
$\gamma = 0.9$

Example: Mapping Value Function to Policy

- Agent selects optimal action from *V*:

$$\pi(s) = \text{argmax}_a [r(s,a) + \gamma V(\delta(s, a)]$$

Model + V:

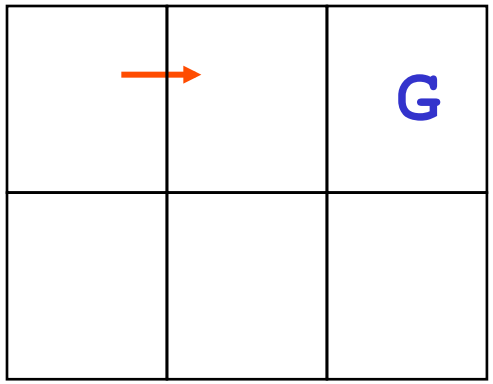


$\gamma = 0.9$

- a: 0 + 0.9 x 100 = 90
- b: 0 + 0.9 x 81 = 72.9

➢ select a

π:

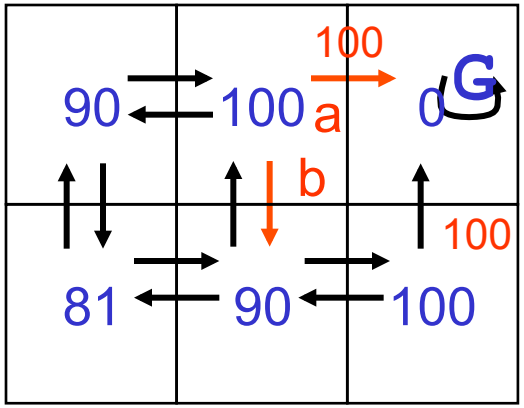


Example: Mapping Value Function to Policy

- Agent selects optimal action from *V*:

$$\pi(s) = \text{argmax}_a [r(s,a) + \gamma V(\delta(s, a)]$$

Model + V:

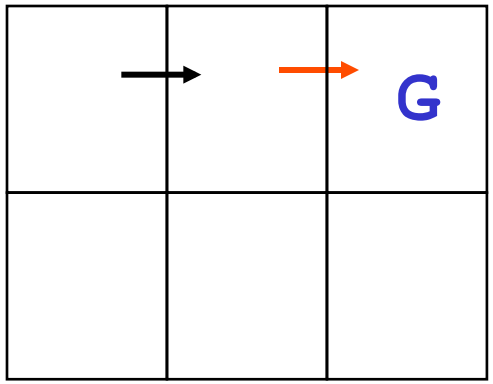


$\gamma = 0.9$

- a: 100 + 0.9 x 0 = 100
- b: 0 + 0.9 x 90 = 81

➢ select a

π:

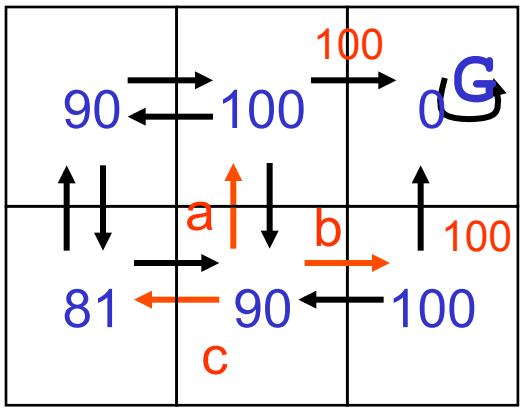


Example: Mapping Value Function to Policy

- Agent selects optimal action from *V*:

$$\pi(s) = \text{argmax}_a \left[r(s,a) + \gamma V(\delta(s, a)) \right]$$

Model + V:

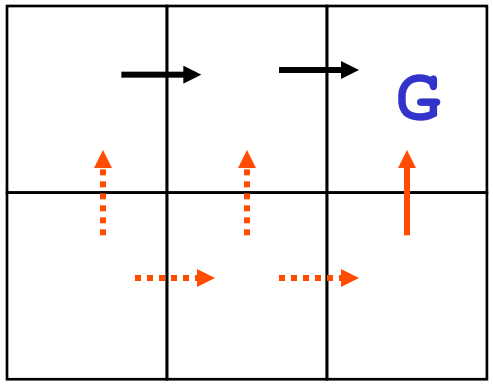


$\gamma = 0.9$

- a: ?
- b: ?
- c: ?

➢ select ?

π:

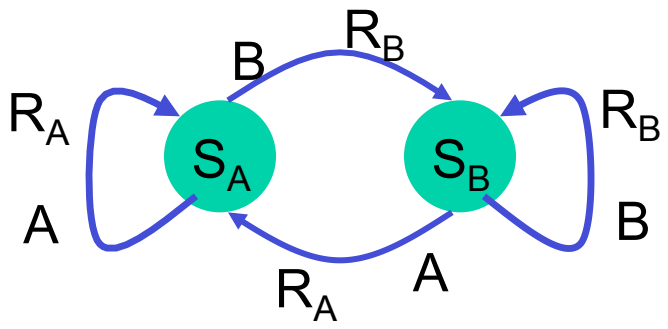


Markov Decision Processes

- Motivation
- Markov Decision Processes
- Computing Policies From a Model
  - Value Functions
  - Mapping Value Functions to Policies
  - Computing Value Functions through Value Iteration
  - An Alternative: Policy Iteration
- Summary

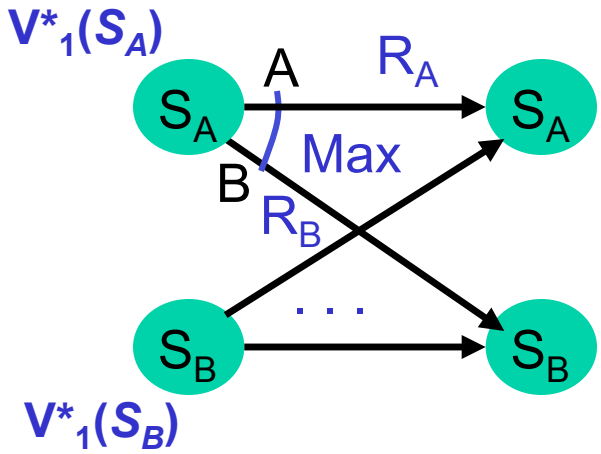
Value Function V^* for an optimal policy π^*

Example



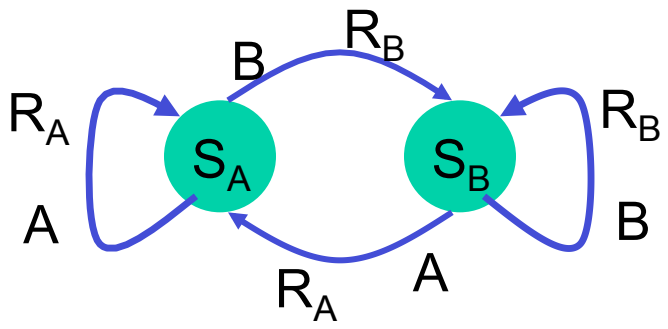
- Optimal value function for a one step horizon:

$$V^*_1(s) = \max_{a_i} [r(s,a_i)]$$



Value Function V^* for an optimal policy π^*

Example



- Optimal value function for a one step horizon:

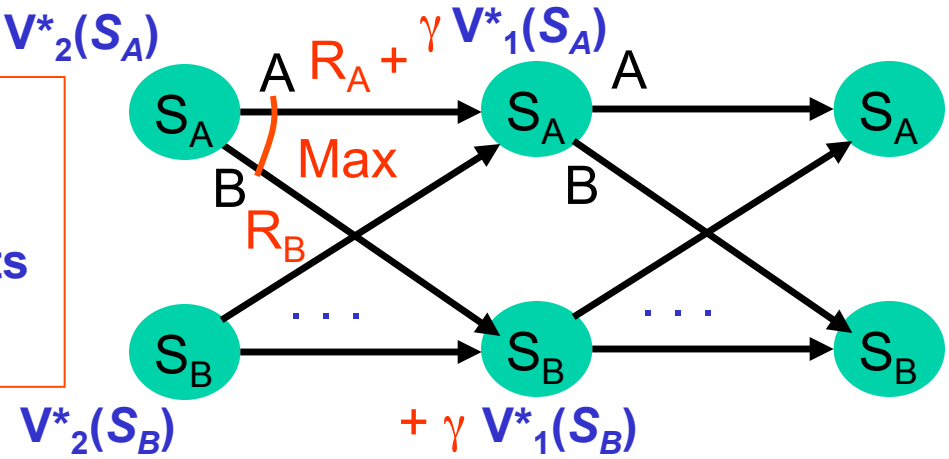
$$V^*_1(s) = \max_{a_i} [r(s,a_i)]$$

- Optimal value function for a two step horizon:

$$V^*_2(s) = \max_{a_i} [r(s,a_i) + \gamma V^*_1(\delta(s, a_i))]$$

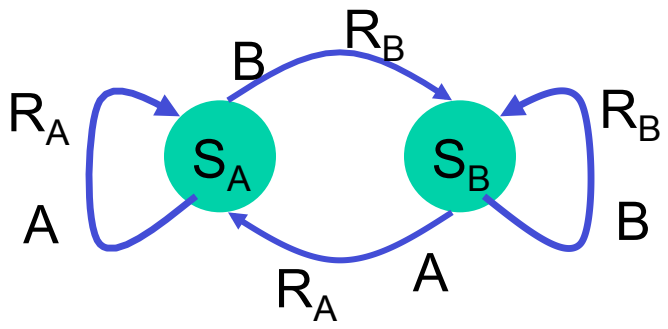
Instance of the Dynamic Programming Principle:

- **Reuse shared sub-results**
- **Exponential saving**



Value Function V^* for an optimal policy π^*

Example



- Optimal value function for a one step horizon:

$$V^*_1(s) = \max_{a_i} [r(s,a_i)]$$

- Optimal value function for a two step horizon:

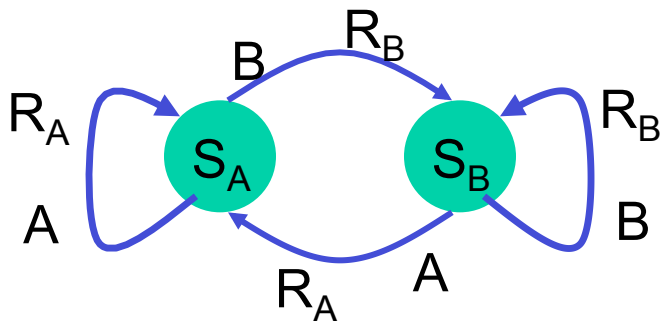
$$V^*_2(s) = \max_{a_i} [r(s,a_i) + \gamma V^*_1(\delta(s, a_i))]$$

- Optimal value function for an n step horizon:

$$V^*_n(s) = \max_{a_i} [r(s,a_i) + \gamma V^*_{n-1}(\delta(s, a_i))]$$

Value Function V^* for an optimal policy π^*

Example



- Optimal value function for a one step horizon:

$$V^*_1(s) = \max_{a_i} [r(s,a_i)]$$

- Optimal value function for a two step horizon:

$$V^*_2(s) = \max_{a_i} [r(s,a_i) + \gamma V^*_1(\delta(s, a_i))]$$

- Optimal value function for an n step horizon:

$$V^*_n(s) = \max_{a_i} [r(s,a_i) + \gamma V^*_{n-1}(\delta(s, a_i))]$$

➢ Optimal value function for an infinite horizon:

$$V^*(s) = \max_{a_i} [r(s,a_i) + \gamma V^*(\delta(s, a_i))]$$

Solving MDPs by Value Iteration

Insight: Calculate optimal values iteratively using Dyn Prog

Algorithm:

- Iteratively calculate value using Bellman's Equation:

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

- Terminate when values are "close enough"

$$|V^*_{t+1}(s) - V^*_t(s)| < \varepsilon$$

- Agent selects optimal action by one step lookahead on V^*:

$$\pi^*(s) = \text{argmax}_a [r(s,a) + \gamma V^*(\delta(s, a)]$$

Convergence of Value Iteration

- If terminate when values are "close enough"

$$|V_{t+1}(s) - V_t(s)| < \varepsilon$$

  Then:

$$\text{Max}_{s \text{ in } S} |V_{t+1}(s) - V^*(s)| < 2\varepsilon\gamma/(1 - \gamma)$$

- Converges in polynomial time.
- Convergence guaranteed even if updates are performed infinitely often, but asynchronously and in any order.

Convergence of Value iteration in Detail:

Given:

- Initialization

$$V_0(s) = 0$$

- Recursion

$$V_t(s) = \max_a \sum_{s' \in S} T(s,a,s') \left(R(s,a,s') + \gamma V_{t-1}(s') \right)$$

until .

$$\left\| V_t - V_{t-1} \right\|_\infty < \varepsilon$$

Contraction

Lemma. The discounted Bellman backup operator T is a max-norm contraction:

$$\|TV \quad T\overline{V}\|_\infty \leq \gamma \|V \quad \overline{V}\|_\infty$$

where .

$$\left\|V\right\|_\infty = \max_s V(s)$$

Contraction

Proof.

$$
\begin{aligned}
& |(TV)(s) \quad (T\overline{V})(s)| \\
= \quad & \max_a \bullet R(s,) + \gamma \sum_{s'} \Big(P(s'|s,a)V(s') \Big) \quad \max_a \bullet R(s,) + \gamma \sum_{s'} \Big(P(s'|s,a)\overline{V}(s') \Big) \Big(\\
\leq & \max_a \quad \bullet R(s,) + \gamma \sum_{s'} \Big(P(s'|s,a)V(s') \Big) \quad \bullet R(s,) + \gamma \sum_{s'} \Big(P(s'|s,a)\overline{V}(s') \Big) \Big(\\
= & \max_a \gamma \sum_{s'} \Big(P(s'|s,a)|V(s') \quad \overline{V}(s')| \\
\leq & \max_a \gamma \sum_{s'} \Big(P(s'|s,a)\|V \quad \overline{V}\|_\infty \\
= & \gamma\|V \quad \overline{V}\|_\infty \max_a \sum_{s'} \Big(P(s'|s,a) \\
= & \gamma\|V \quad \overline{V}\|_\infty
\end{aligned}
$$

Performance bound

Theorem. If $\|V_{t+1} - V_t\| < \varepsilon$, then $\|V_t - V^*\| < \frac{\varepsilon}{1-\gamma}$.

Performance bound

Theorem. If $\|V_{t+1} - V_t\| < \varepsilon$, then $\|V_t - V^*\| < \frac{\varepsilon}{1-\gamma}$

Proof.

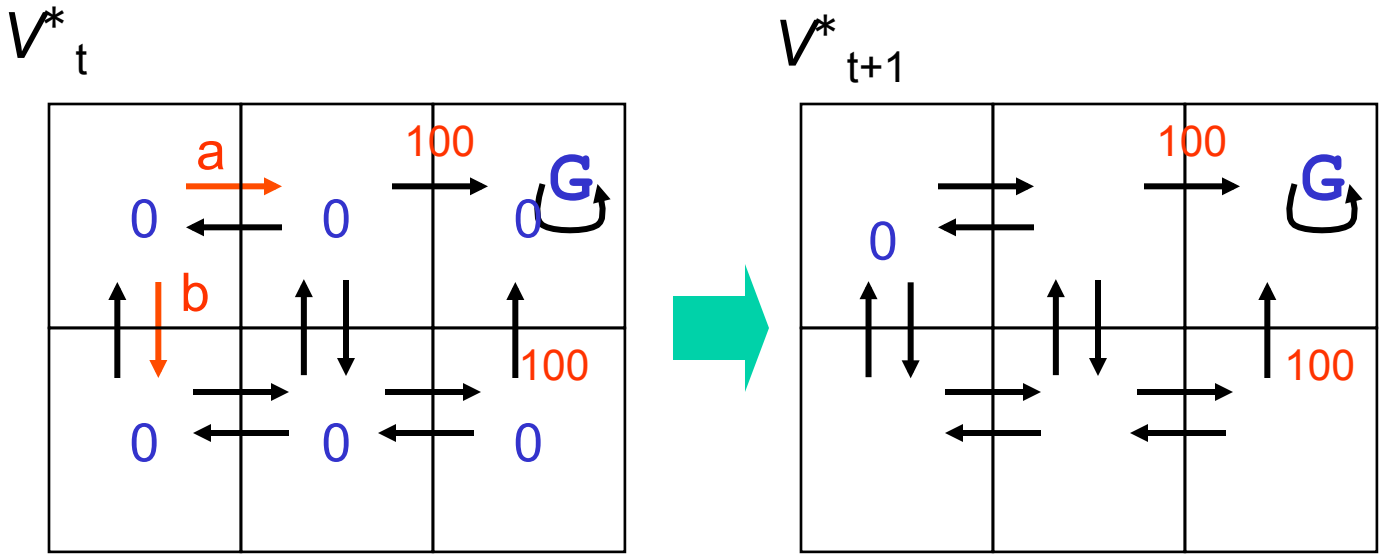
$$
\begin{aligned}
&\|V_t - V^*\| \\
=&\|V_t - V_{t+1} + V_{t+1} - V^*\| \\
\leq&\|V_{t+1} - V_t\| + \|V_{t+1} - V^*\| \\
=&\epsilon + \|TV_t - TV^*\| \\
=&\epsilon + \gamma\|V_t - V^*\|
\end{aligned}
$$

Rearranging the inequality gives the final result.

Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

$\gamma = 0.9$



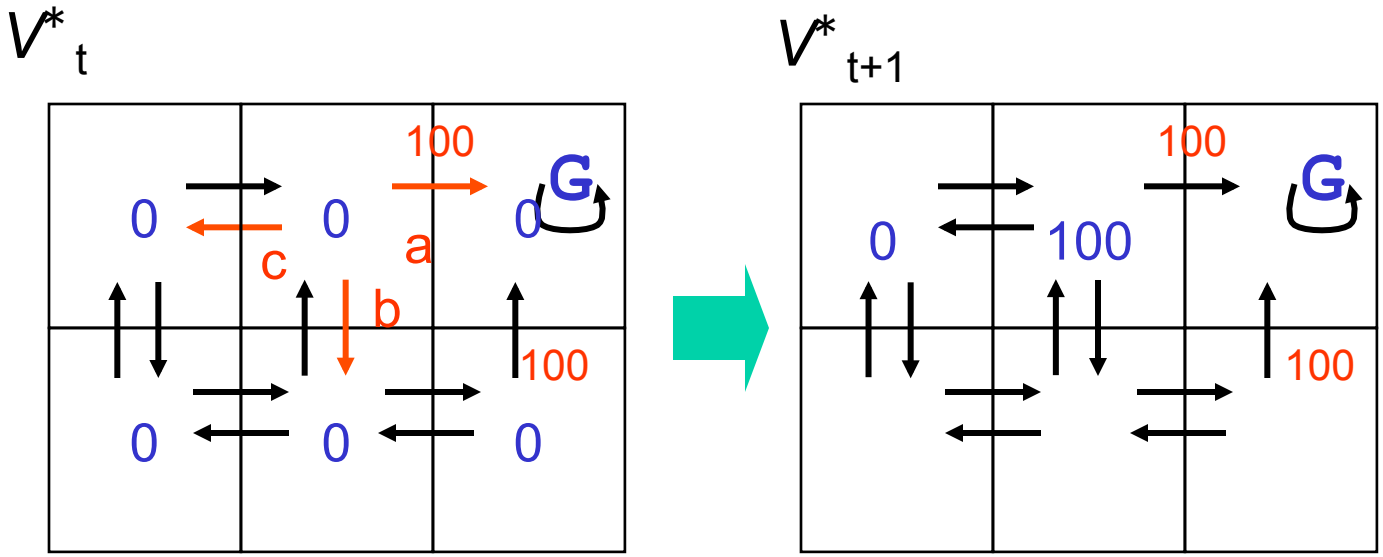
- a: 0 + 0.9 x 0 = 0
- b: 0 + 0.9 x 0 = 0

➢ Max = 0

Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

$\gamma = 0.9$



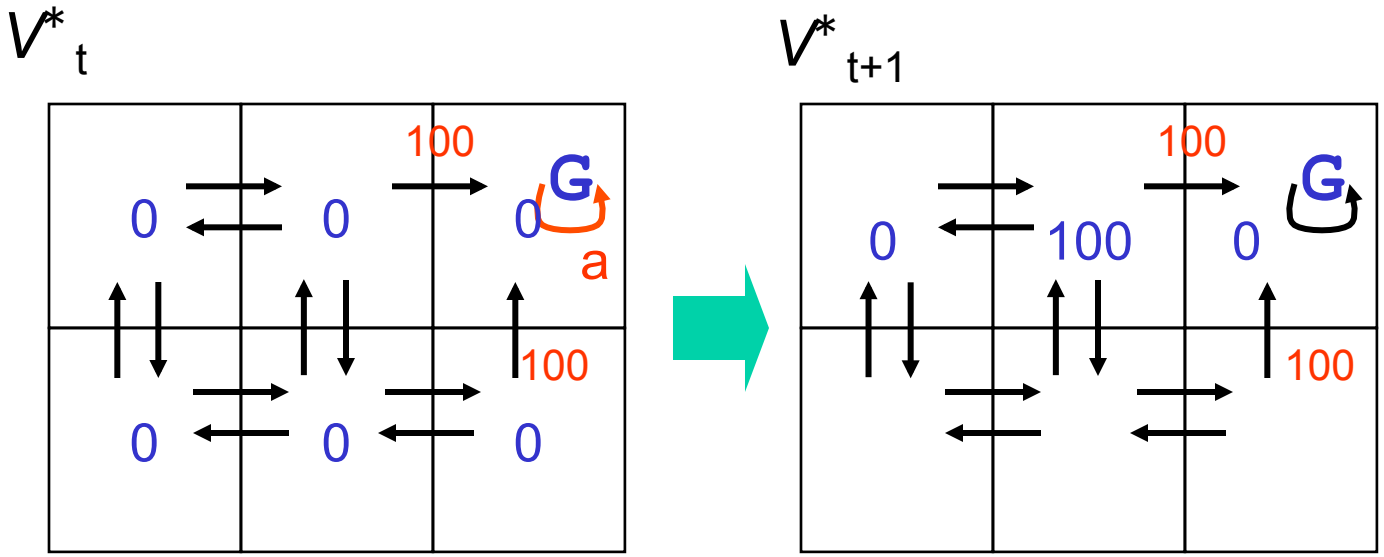
- a: 100 + 0.9 x 0 = 100
- b: 0 + 0.9 x 0 = 0
- c: 0 + 0.9 x 0 = 0

➢ Max = 100

Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

$\gamma = 0.9$

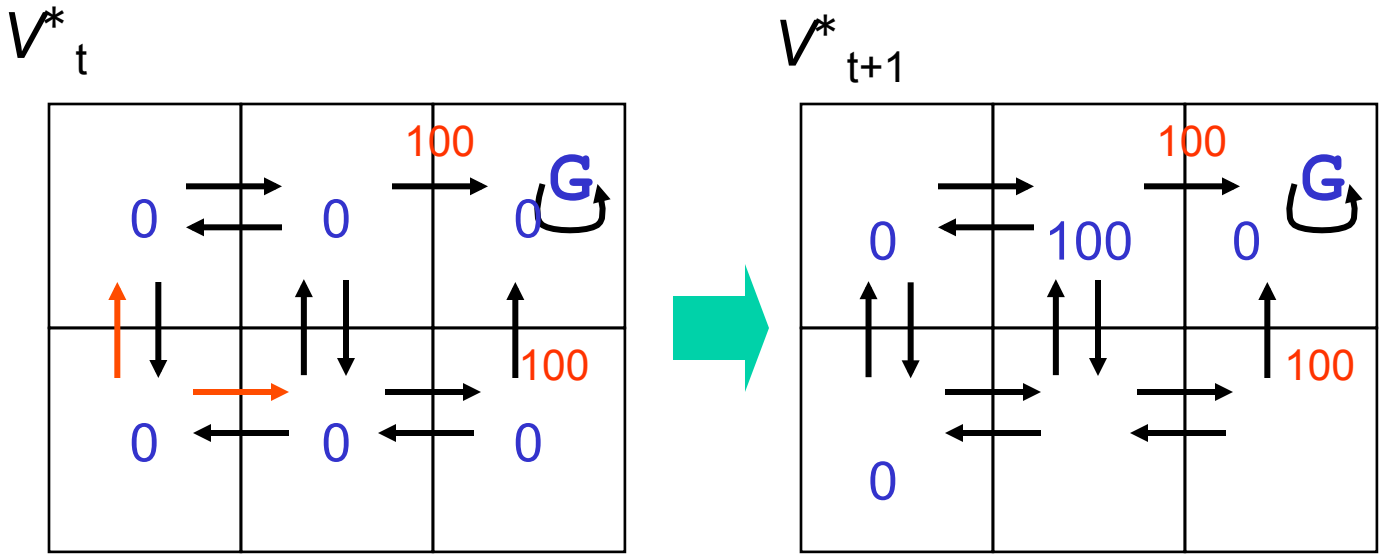


- a: 0 + 0.9 x 0 = 0
  - ➢ Max = 0

Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

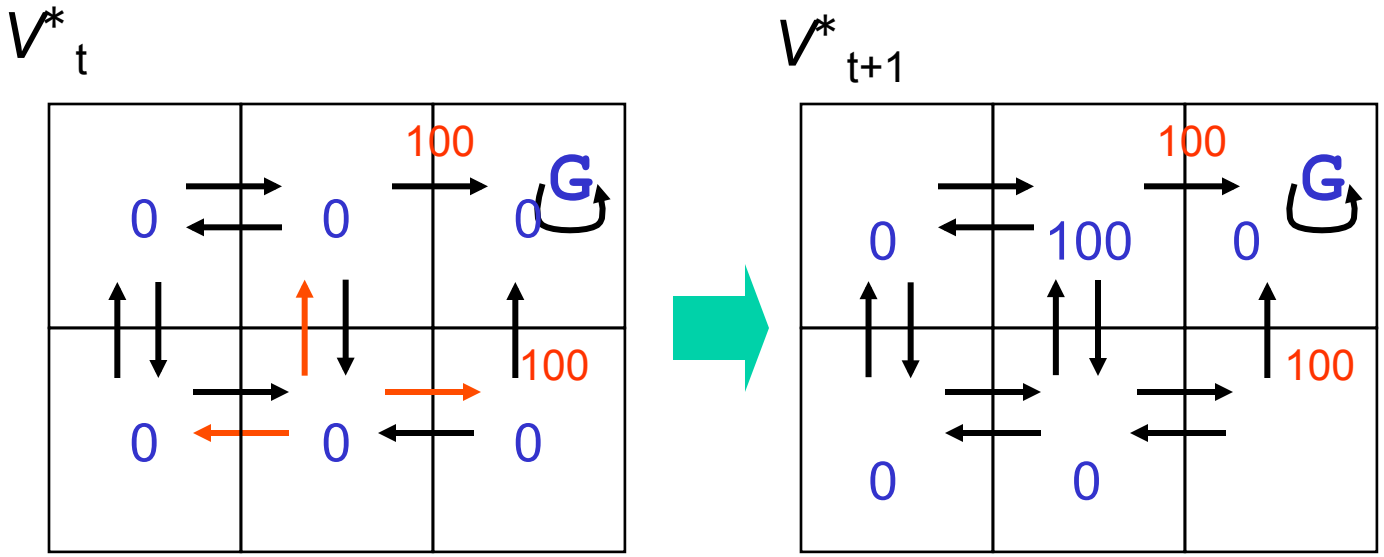
$$\gamma = 0.9$$



Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

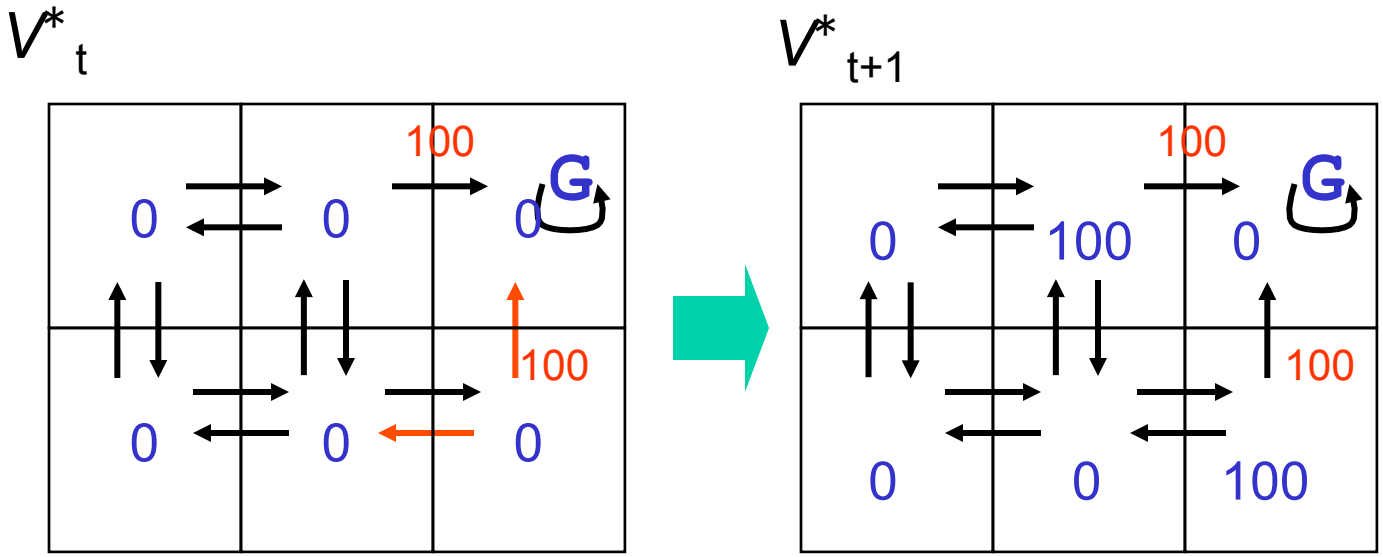
$\gamma = 0.9$



Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

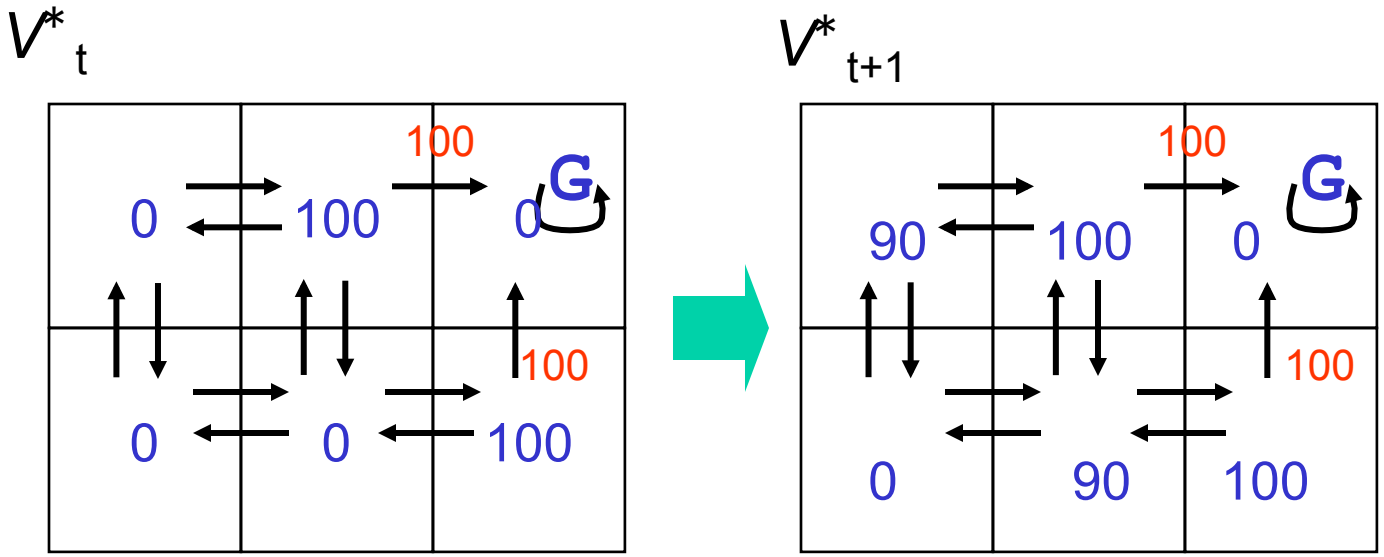
$\gamma = 0.9$



Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

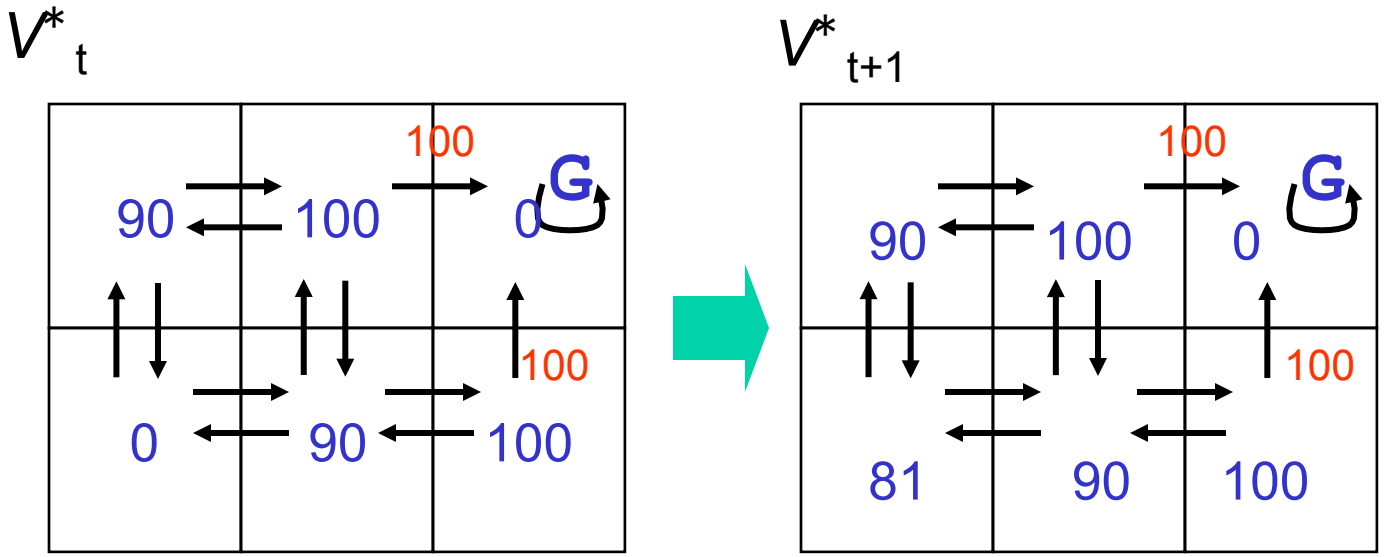
$$\gamma = 0.9$$



Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

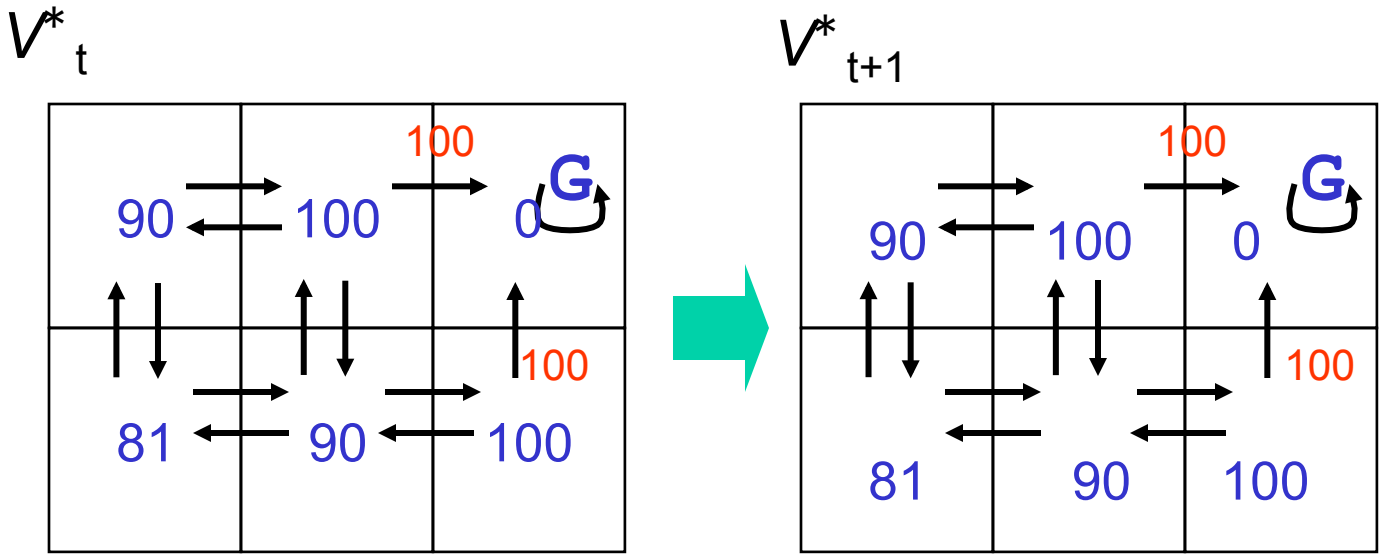
$$\gamma = 0.9$$



Example of Value Iteration

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

$$\gamma = 0.9$$



Markov Decision Processes

- Motivation
- Markov Decision Processes
- Computing policies from a modelValue Functions
  - Mapping Value Functions to Policies
  - Computing Value Functions through Value Iteration
  - An Alternative: Policy Iteration (appendix)
- Summary

Appendix: Policy Iteration

Idea: Iteratively improve policy.

1. Policy Evaluation: Given policy π_i calculate $V_i = V^{\pi i}$, the utility of each state if π_i were executed.
2. Policy Improvement: Calculate new maximum utility policy π_{i+1} using one-step look ahead based on V_i.

- π_i improves at every step, converging if $\pi_i = \pi_{i+1}$.
- Computing V_i is simpler than for Value Iteration (no max):

$$V^*_{t+1}(s) \leftarrow r(s, \pi_i(s)) + \gamma V^*_t(\delta(s, \pi_i(s)))]$$

  - Solve linear equations in $O(N^3)$
  - Solve iteratively, similar to value iteration.

Markov Decision Processes

- Motivation
- Markov Decision Processes
- Computing policies from a model
  - Value Iteration
  - Policy Iteration
- Summary

Markov Decision Processes (MDPs)

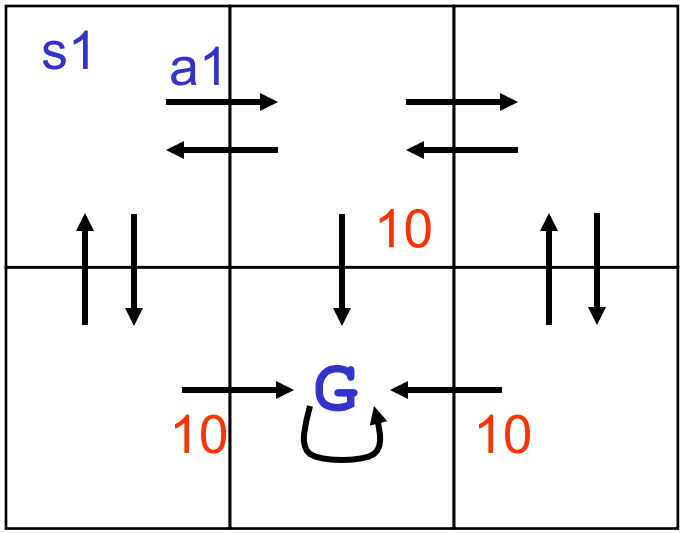
Model:

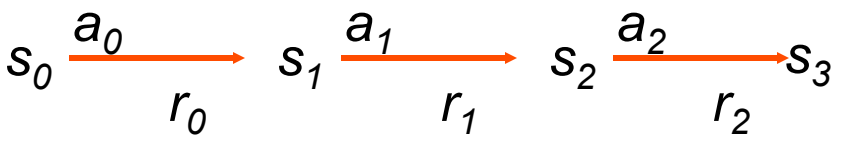
- Finite set of states, *S*
- Finite set of actions, *A*
- Probabilistic state transitions, $\delta(s,a)$
- Reward for each state and action, $R(s,a)$

Process:

- Observe state s_t in S
- Choose action a_t in A
- Receive immediate reward r_t
- State changes to s_{t+1}

Deterministic Example:





Crib Sheet: MDPs by Value Iteration

Insight: Calculate optimal values iteratively using Dynamic Programming.

Algorithm:

- Iteratively calculate value using Bellman's Equation:

$$V^*_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma V^*_t(\delta(s, a))]$$

- Terminate when values are "close enough"

$$|V^*_{t+1}(s) - V^*_t(s)| < \varepsilon$$

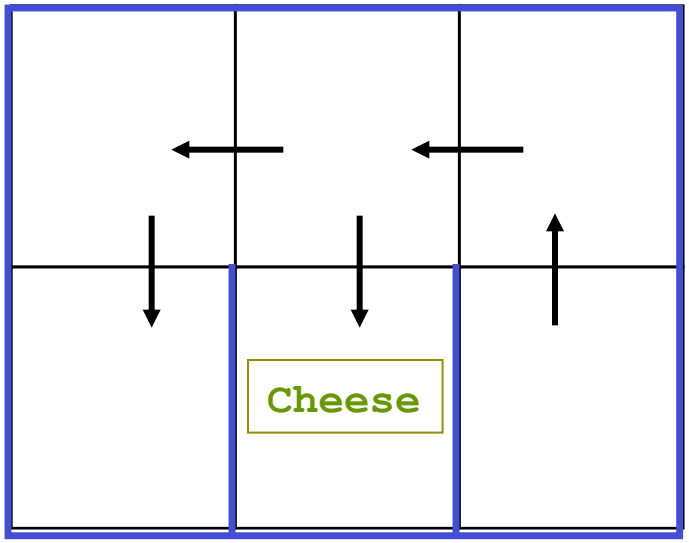
- Agent selects optimal action by one step look ahead on V^*:

$$\pi^*(s) = \text{argmax}_a [r(s,a) + \gamma V^*(\delta(s, a)]$$

Ideas in this lecture

- Objective is to accumulate rewards,
  rather than goal states.
- Objectives are achieved along the way,
  rather than at the end.
- Task is to generate policies for how to act in all situations,
  rather than a plan for a single starting situation.
- Policies fall out of value functions, which describe the
  greatest lifetime reward achievable at every state.
- Value functions are iteratively approximated.

How Might a Mouse Search a Maze for Cheese?



- By Value Iteration?
- What is missing?

16.412J / 6.834J Cognitive Robotics
Spring 2016