CSCI 561
Foundation for Artificial Intelligence

Probabilistic Decision Making

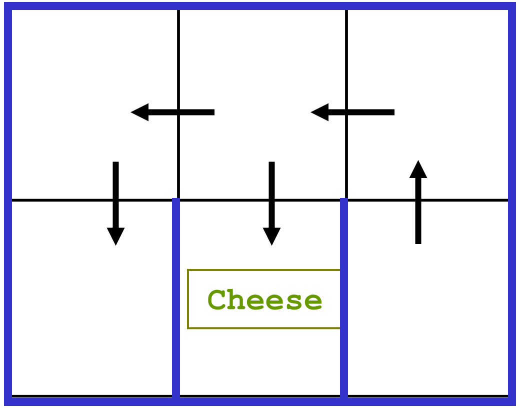
Professor Wei-Min Shen
University of Southern California

Outline

- Probabilistic decision making
  - Motivation
  - Decision Problems
  - Markov Decision Process (MDP)
  - Value Iteration
  - Policy Iteration

Slides contributions from: Brian C. Williams
(MIT 16.410), Manuela Veloso,
Reid Simmons, & Tom Mitchell, CMU

How Might a Mouse Search a Maze for Cheese?

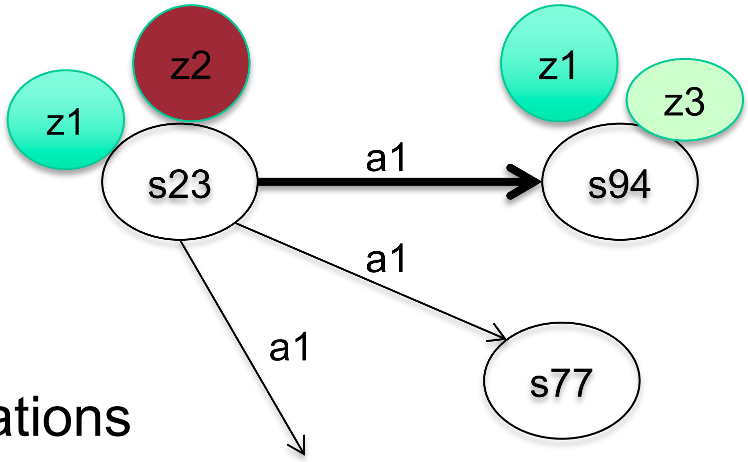


- State Space Search?
- As a Constraint Satisfaction Problem?
- Goal-directed Planning?
- Linear Programming?

What is missing?

Action and Sensor Models (review)

1. Actions
2. Percepts (observations)
3. States
4. Appearance: states → observations
5. Transitions: (states, actions) → states
6. Current State

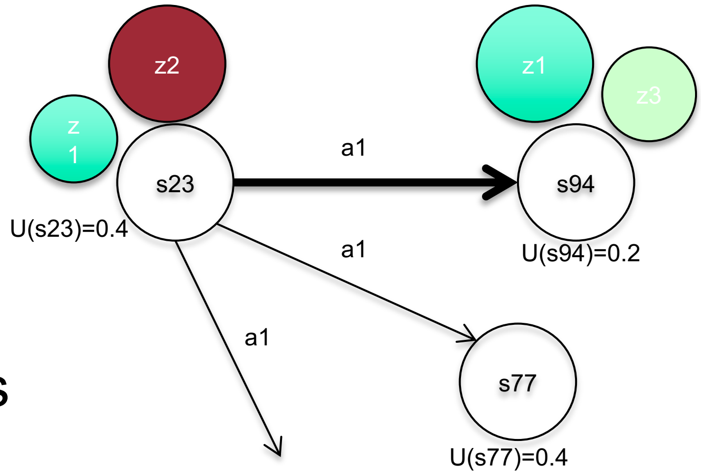


•What about the goals?

Utility Value of States ⬄ Goal Information

1. Actions
2. Percepts (observations)
3. States
4. Appearance: states → observations
5. Transitions: (states, actions) → states
6. Current State
7. Rewards: R(s) or R(s,a) (related to the goals, given to the agent from env)

(8) Utility Value of States: $U(s)$ (how good for the goals, must compute)

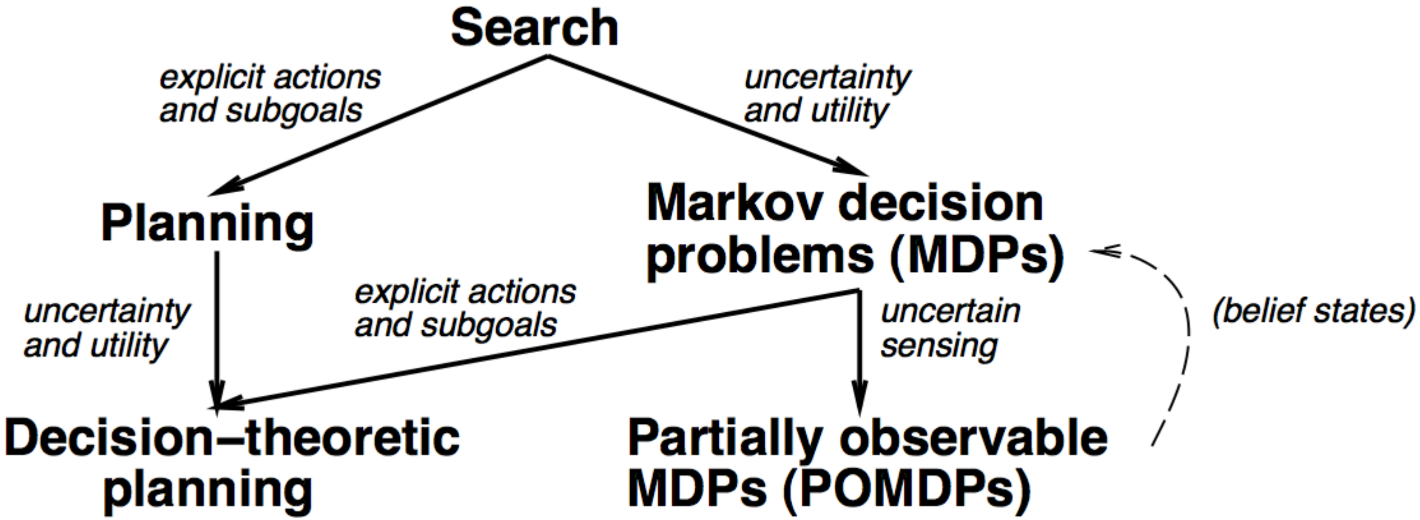


Note that rewards can be given from states R(s) or state-action R(s,a)

The Key Representations

- Model the environment by States, Actions, Percepts, and
- Transition model $\phi = P(s_{t+1}|s_t,a_t)$ may be probabilistic
- Sensor Model $\theta = P(z|s)$ may be probabilistic
- States may have Utility Value U(s) or V(s)
- Agents may receive Reward ***r*** (implicit "goals")
  - When entering a state: $s \rightarrow r$
  - When performing an action in a state: $(s, a) \rightarrow r$
- Behaviors may be represented as Policy: $s \rightarrow a$
- Objective: find the optimal policy based on utilities or rewards

Sequential Decision Problems



Markov Decision Process (MDP)

- A MDP consists of
  - State S
  - Actions A
  - Transition Model Φ(s'|s,a)
  - Initial State s_0 Probability Distribution π
  - ~~Sensor Model θ(z|s)~~ // a typical MDP has no sensor model
  - A reward function R(s) or R(s, a) // we use R(s, a) in today's lecture

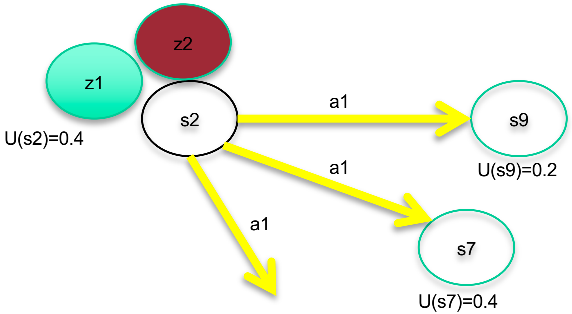
Maximum Expected Utility (MEU) and Rational Agents

- Every state has a utility value *U*(*s*)
- The <u>expected utility of an action</u> given the current evidence or observation *e*, is the average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$EU(a \mid e) = \sum_{s'} P(result(a) = s' \mid a, e) U(s')$$

- The principle of maximum expected utility (MEU) is that a **rational** agent should choose the action that maximizes its expected utility:

$$action = \arg\max_{a} EU(a \mid e)$$



POMDP: Sensor Model, Belief States

- A Partially Observable MDP (POMDP) is defined as follows:
  - A set of states S (with an initial state s_0)
  - A set of Actions: A(s) of actions in each state s
  - A transition model P(s'|s,a), or T(s,a,s')
  - A reward function R(s), or R(s,a)
  - A sensor model P(*e*|*s*) 🡐 🡐
  - A belief of what the current state is *b*(*s*) 🡐 🡐
- Belief States (where am I now? What is my current state?):
  - If *b(s)* was the previous belief state, and the robot does action "*a*" and then perceives a new evidence "*e*", then the new belief state:

$$b'(s') = \alpha P(e \mid s') \sum P(s' \mid s, a) b(s)$$

where α is a normalization constant making the belief states sum to 1

Goals, Rewards, Utilities, Policies

- Goals
  - Given to the agent from the problem statements
- Rewards
  - Given to the agent, designed based on the goals
- Utility values for states
  - Computed by the agent, based on the rewards
- Policies
  - Computed or learned by the agent
  - Used by the agent to select its actions
  - The better a policy, the more rewards it collects

Compute Utilities from Rewards over Time

- Utility Value = sum of all future rewards
  - Add the rewards as they are

$$U_h = ([s_0, s_1, s_2, ...]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

  - Discount the far-away rewards in the future

$$U_h = ([s_0, s_1, s_2, ...]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

- The expected future utility value $U^{\pi}(s)$ obtained by executing a policy *π* starting from *s*

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

The Optimal Policy

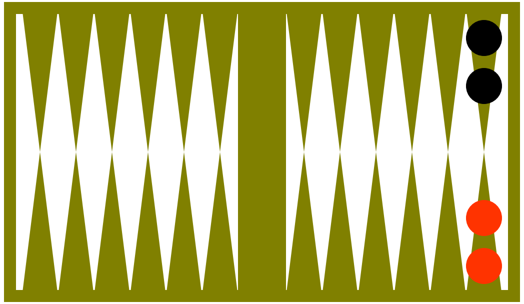
$$\pi_s^* = \arg\max_{\pi} U^{\pi}(s)$$

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



<u>States:</u>

- Board configurations (10^{20})

<u>Actions:</u>

- Moves

<u>Rewards:</u>

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

- Trained by playing 1.5 million games against self.

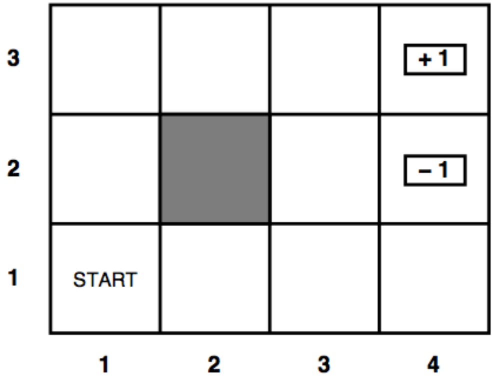
➔ Currently, roughly equal to best human player.

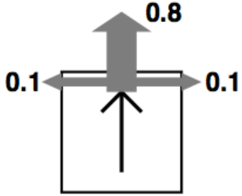
MDP Examples: Aerial Robotics [Feron et al.]
Computing a Solution from a Continuous Model

Markov Decision Processes

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

Example MDP





Model $M_{ij}^{a} \equiv P(j|i,a) =$ probability that doing a in i leads to j

Each state has a *reward* $R(i)$

$=$ -0.04 (small penalty) for nonterminal states

$= \pm 1$ for terminal states

Example MDP

In search problems, aim is to find an optimal *sequence*

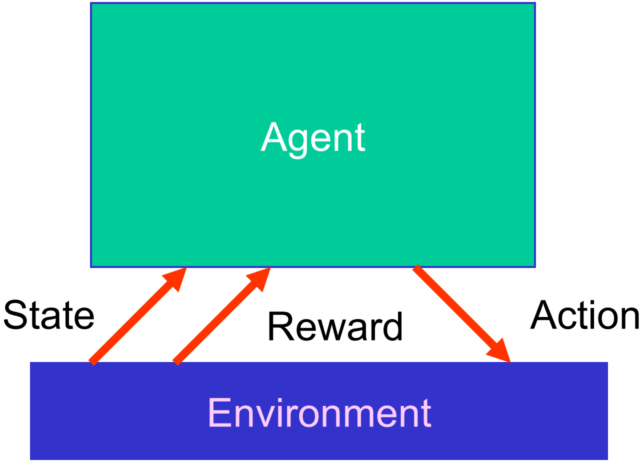
In MDPs, aim is to find an optimal *policy*
i.e., best action for every possible state
(because can't predict where one will end up)

Optimal policy and state values for the given $R(i)$:

	1	2	3	4
3	→	→	→	+ 1
2	↑		↑	− 1
1	↑	←	←	←

	1	2	3	4
3	0.812	0.868	0.912	+ 1
2	0.762		0.660	− 1
1	0.705	0.655	0.611	0.388

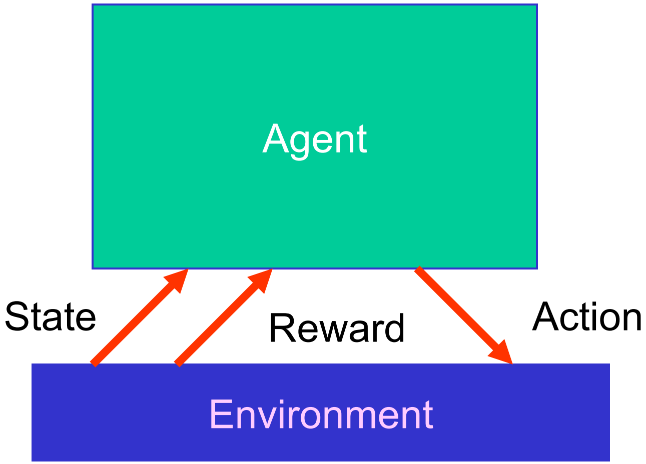
MDP Problem



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

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

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)

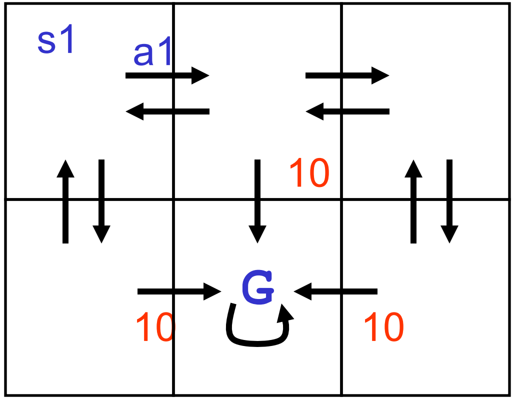
Model:

- 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:

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

Example:



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

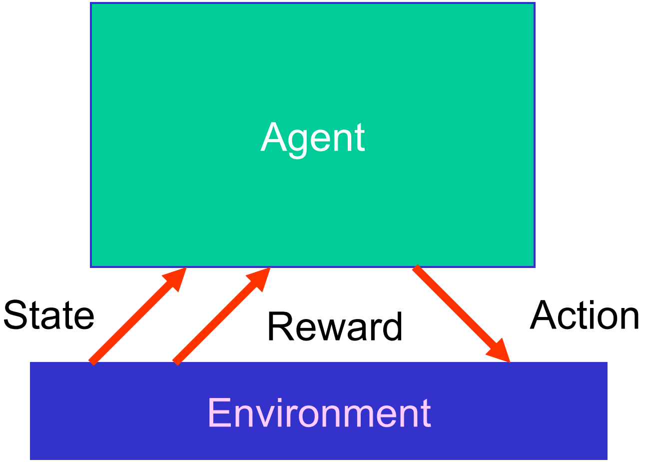
- Legal transitions shown
- Reward on unlabeled transitions is 0.

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

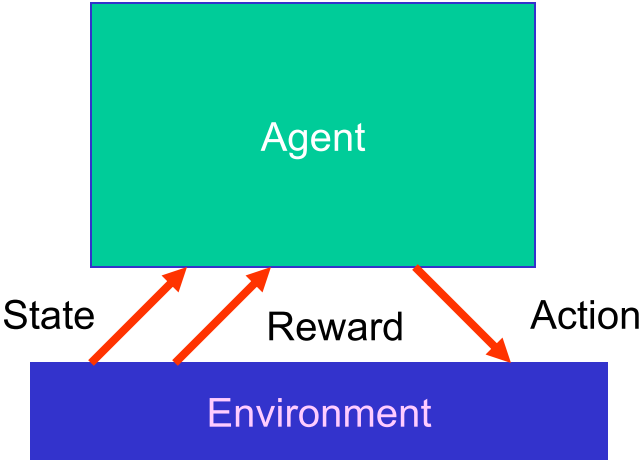
MDP Problem: Model



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

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

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

Utility (aka value)

In *sequential* decision problems, preferences are expressed between *sequences* of states

Usually use an *additive* utility function:

$$U([s_1, s_2, s_3, \ldots, s_n]) = R(s_1) + R(s_2) + R(s_3) + \cdots + R(s_n)$$

(cf. path cost in search problems)

Utility of a *state* (a.k.a. its *value*) is defined to be

$$U(s_i) = \underline{\text{expected sum of rewards until termination}}$$
$$\underline{\text{assuming optimal actions}}$$

Given the utilities of the states, choosing the best action is just MEU: choose the action such that the expected utility of the immediate successors is highest.

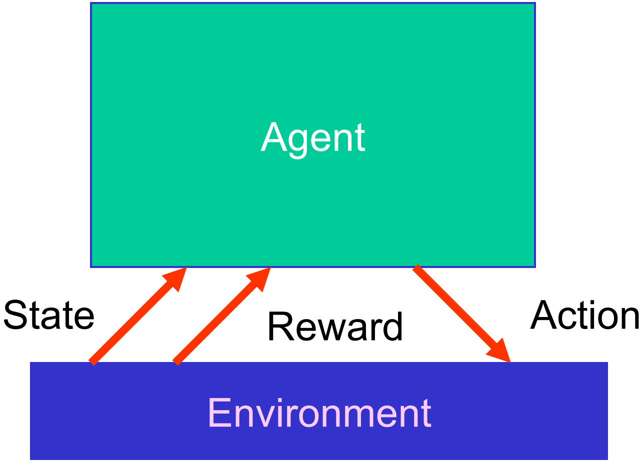
Lifetime Reward

- 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 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

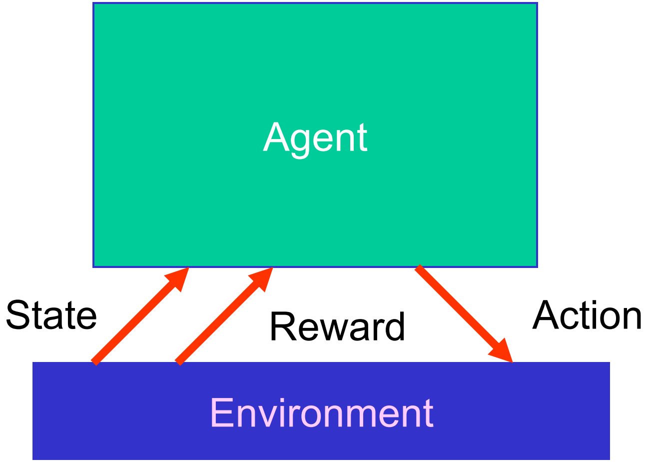


$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3$$

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

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

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

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

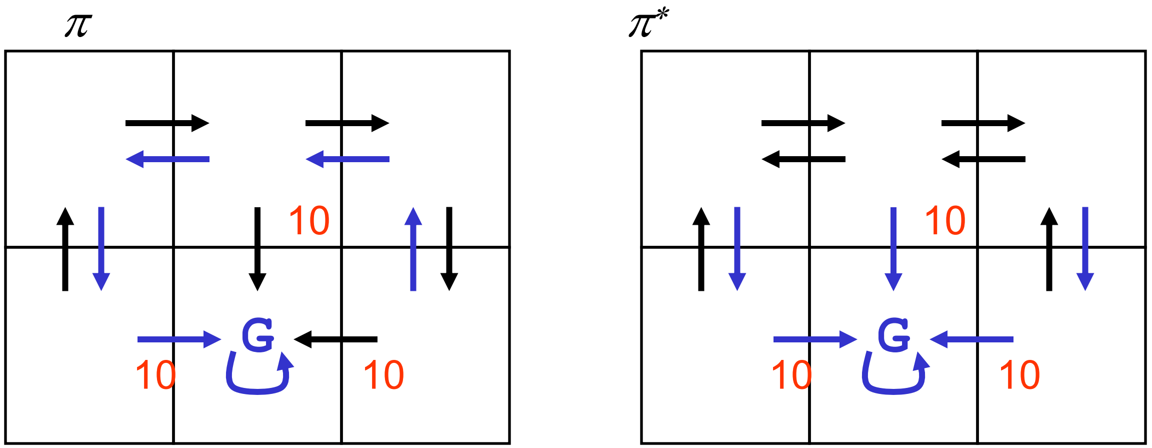
Assume deterministic world

Policy $\pi : S \rightarrow A$

- Selects an action for each state.

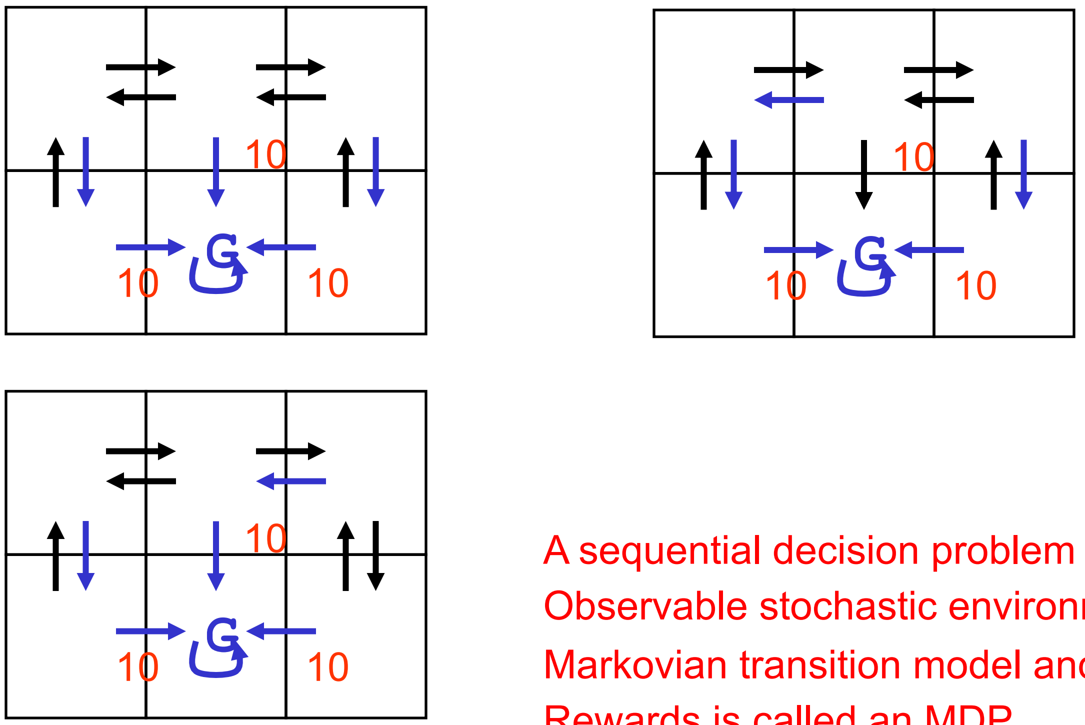
Optimal policy $\pi^* : S \rightarrow A$

- Selects action for each state that maximizes lifetime reward.



Note: with infinite horizon, policy is stationary and independent of start state

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



A sequential decision problem for a fully Observable stochastic environment with Markovian transition model and additive Rewards is called an MDP

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

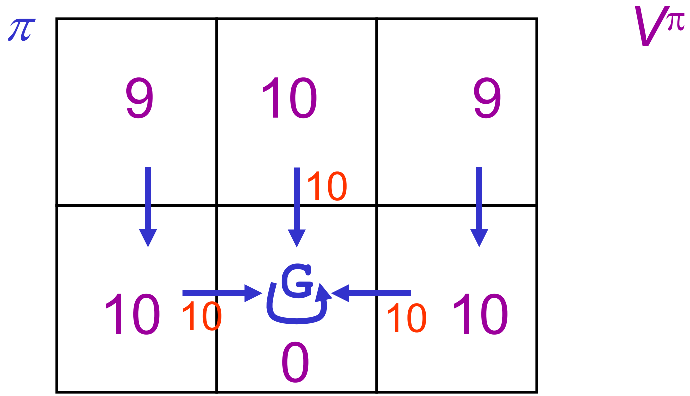
- $V^\pi(s_t)$ is the accumulated lifetime reward resulting from starting in state s_t and repeatedly executing policy π:

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

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

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

Assume γ = .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.

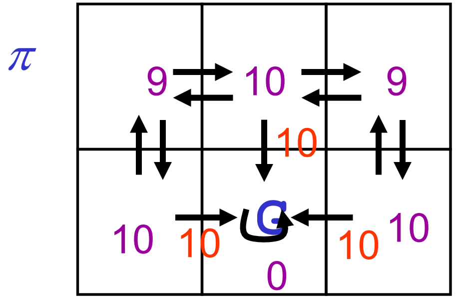
Lifetime reward $Q(s, a_i)$ is:

- the immediate reward for taking action $r(s,a)$ …
- plus lifetime 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.
  - $\delta : S \times A \rightarrow S$
  - $r : S \times A \rightarrow \Re$

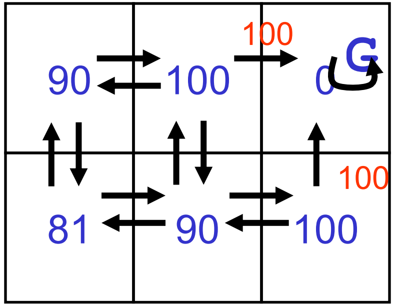


Example: Mapping Value Function to Policy

- Agent selects optimal action from V:

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

Model + V:



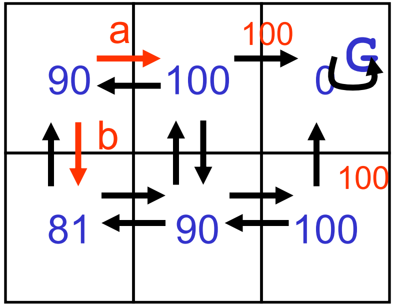
$\gamma = 0.9$

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:

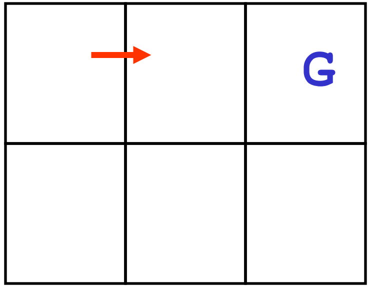


$\gamma = 0.9$

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

➢ select a

π:

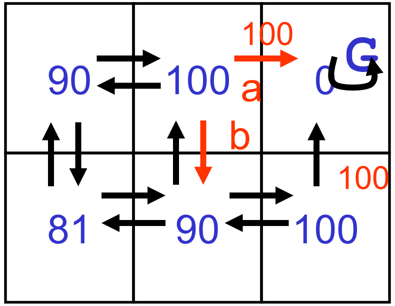


Example: Mapping Value Function to Policy

- Agent selects optimal action from *V*:

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

Model + V:

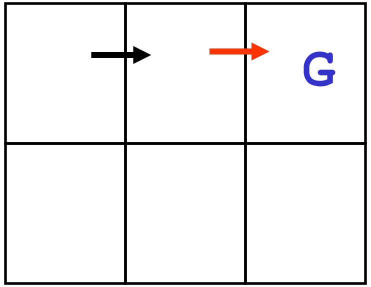


$\gamma = 0.9$

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

➢ select a

π:



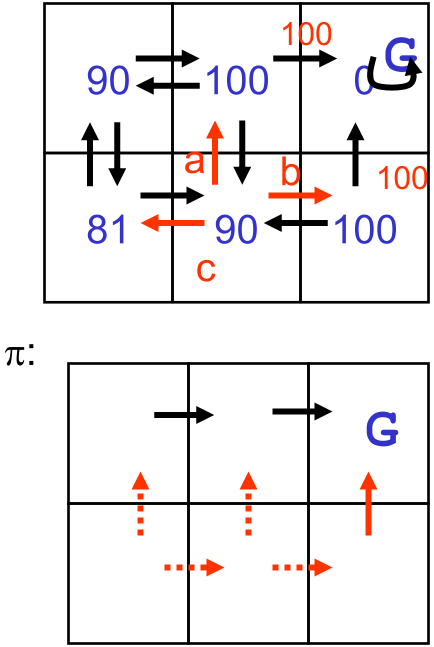
Example: Mapping Value Function to Policy

- Agent selects optimal action from V:

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

Model + V:

$\gamma = 0.9$



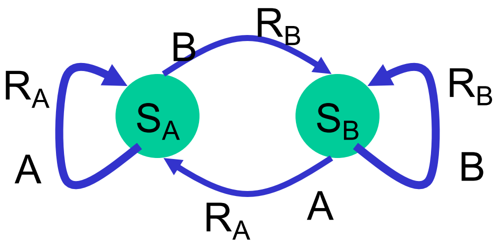
- a: ?
- b: ?
- c: ?
- ➢ select ?

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

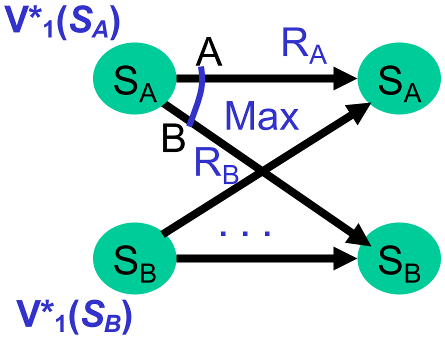
Value Function V^* for an optimal policy π^*

Example



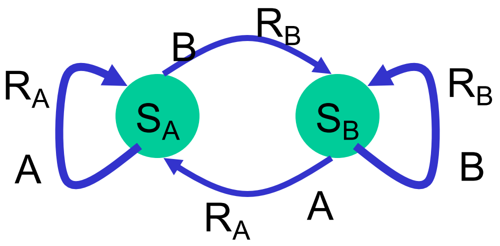
- Optimal value function for a one step horizon:

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



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

Example



- Optimal value function for a one step horizon:

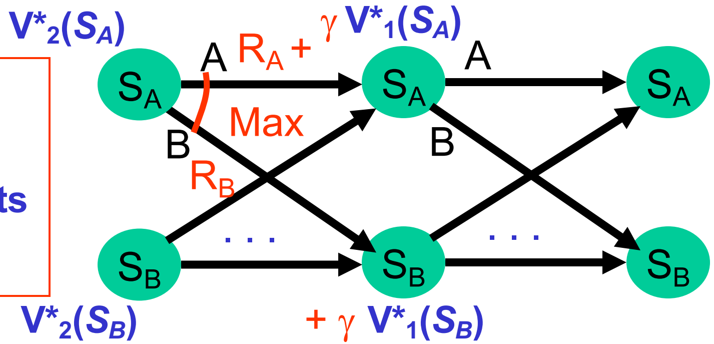
$$V^*_1(s) = \max_{a_i} [r(s,a_i)]$$

- Optimal value function for a two step horizon:

$$V^*_2(s) = \max_{a_i} [r(s,a_i) + \gamma V_1^*(\delta(s, a_i))]$$

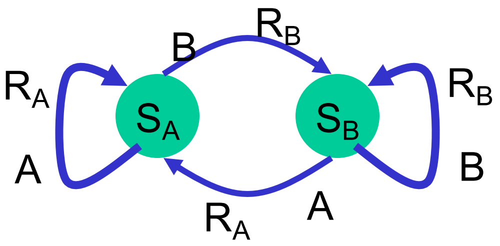
Instance of the Dynamic Programming Principle:

- **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:

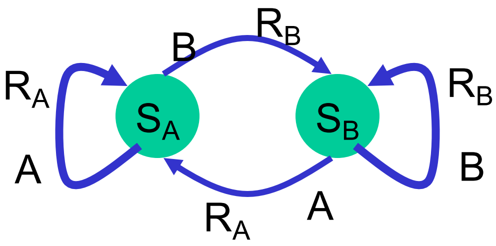
$$V^*_2(s) = \max_{a_i} [r(s,a_i) + \gamma V_1^*(\delta(s, a_i))]$$

- 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:

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

Bellman equation

Definition of utility of states leads to a simple relationship among utilities of neighboring states:

<u>expected sum of rewards</u>
 = <u>current reward</u>
 + <u>expected sum of rewards after taking best action</u>

Bellman equation (1957):

Model $M_{ij}^{a} \equiv P(j|i,a) =$ probability that doing a in i leads to j

$$U(i) = R(i) + \max_a \Sigma_j U(j) M_{ij}^a$$

$$\begin{aligned} U(1,1) = -0.04 & \\ + \max\{ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && \quad up \\ & 0.9U(1,1) + 0.1U(1,2) && \quad left \\ & 0.9U(1,1) + 0.1U(2,1) && \quad down \\ & 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\} && \quad right \end{aligned}$$

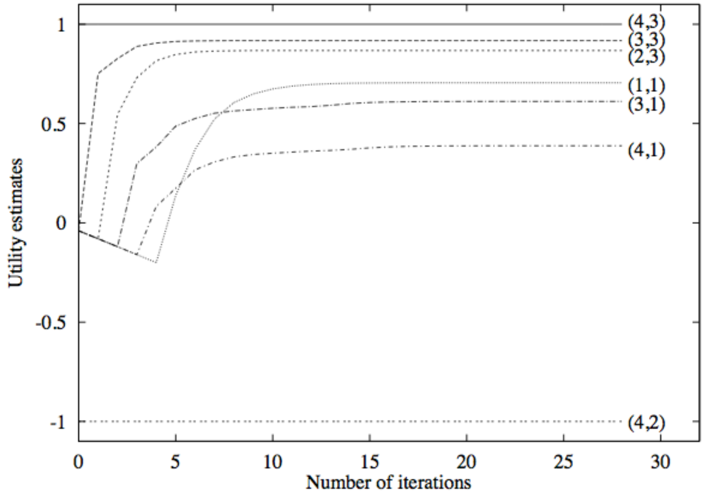
One equation per state $= n$ <u>nonlinear</u> equations in n unknowns

Value iteration algorithm

Idea: Start with arbitrary utility values
Update to make them locally consistent with Bellman eqn.
Everywhere locally consistent ⇒ global optimality

repeat until "no change"

$$U(i) \leftarrow R(i) + \max_a \Sigma_j U(j) M^a_{ij} \qquad \text{for all } i$$



Solving MDPs by Value Iteration

Insight: Can 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 lookahead on V^*:

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

Convergence of Value Iteration

- If terminate when values are "close enough"

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

Then:

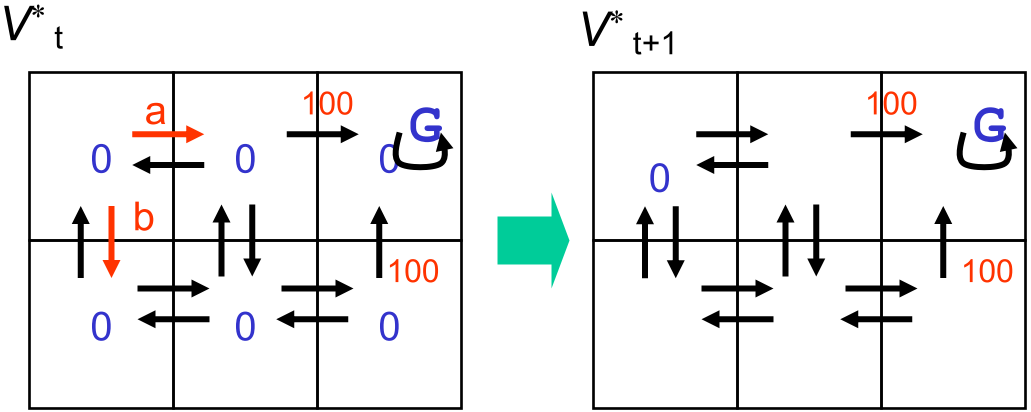
$$\text{Max}_{s \text{ in } S} |V_{t+1}(s) - V^*(s)| < 2\varepsilon\gamma/(1 - \gamma)$$

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

Example of Value Iteration

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

$$\gamma = 0.9$$



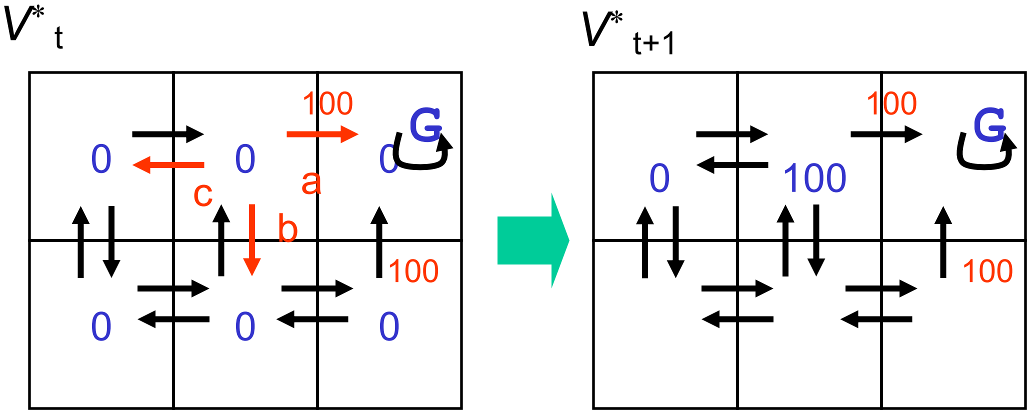
- a: 0 + 0.9 x 0 = 0
- b: 0 + 0.9 x 0 = 0

➢ Max = 0

Example of Value Iteration

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

$$\gamma = 0.9$$



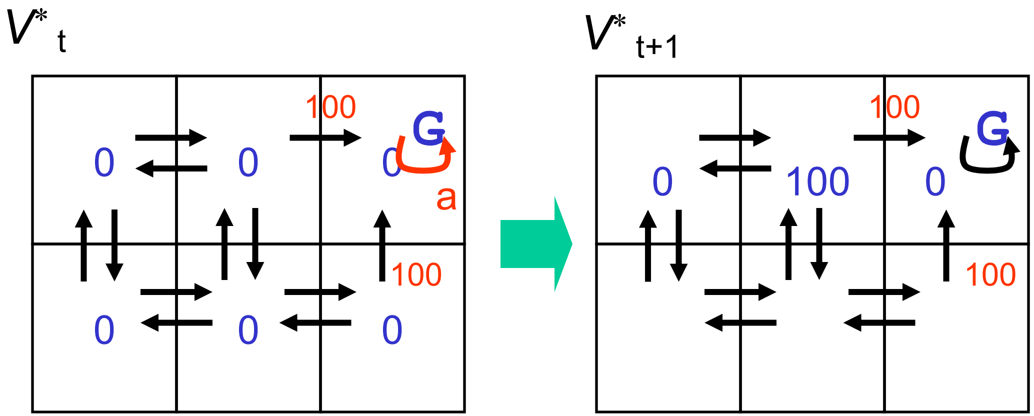
- a: 100 + 0.9 x 0 = 100
- b: 0 + 0.9 x 0 = 0
- c: 0 + 0.9 x 0 = 0

➢ Max = 100

Example of Value Iteration

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

$\gamma = 0.9$

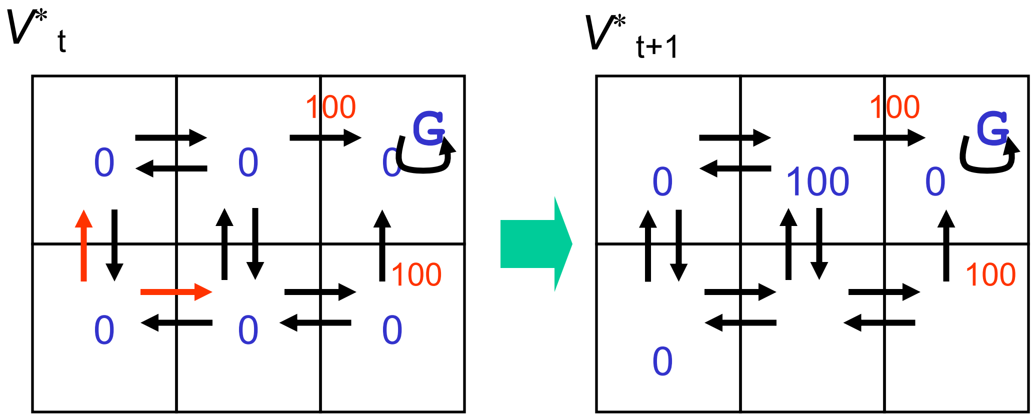


- a: 0 + 0.9 x 0 = 0
  - ➢ Max = 0

Example of Value Iteration

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

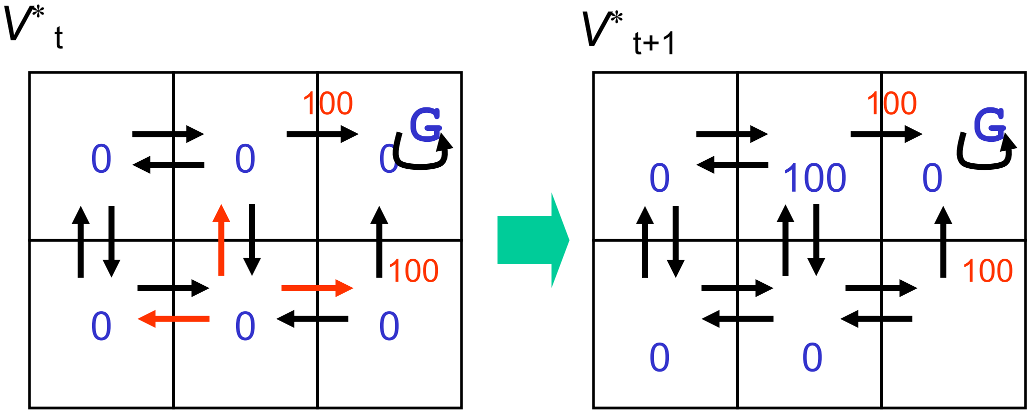
$\gamma = 0.9$



Example of Value Iteration

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

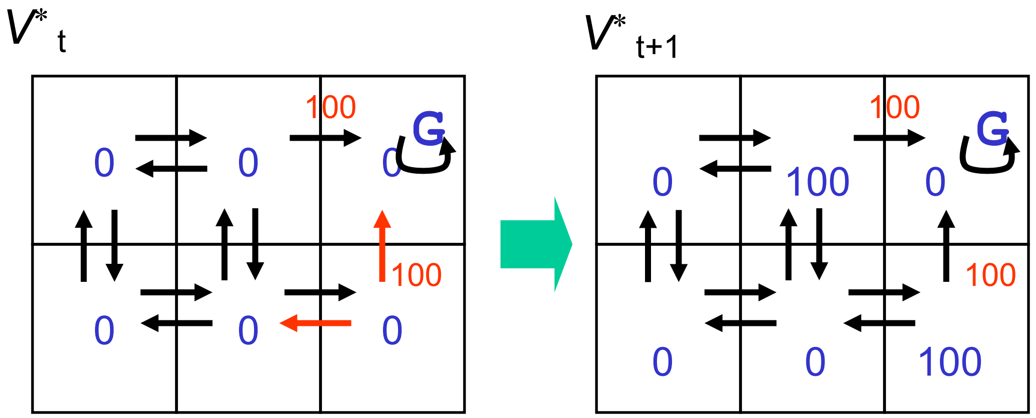
$\gamma = 0.9$



Example of Value Iteration

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

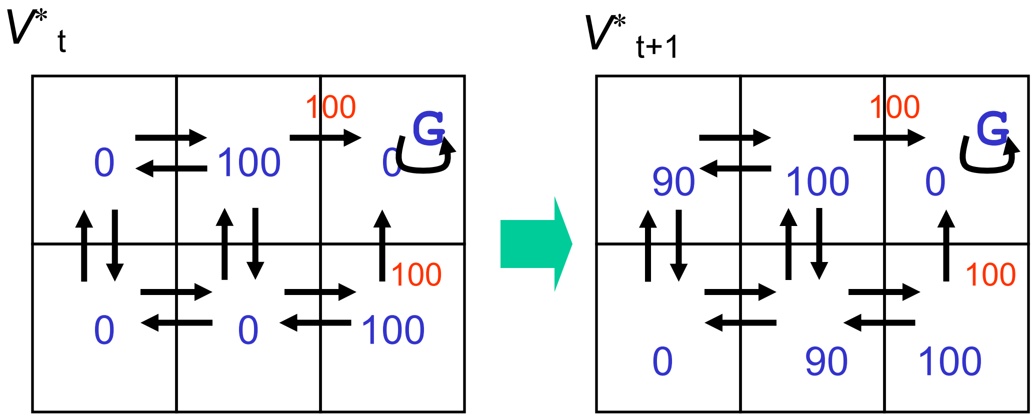
$$\gamma = 0.9$$



Example of Value Iteration

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

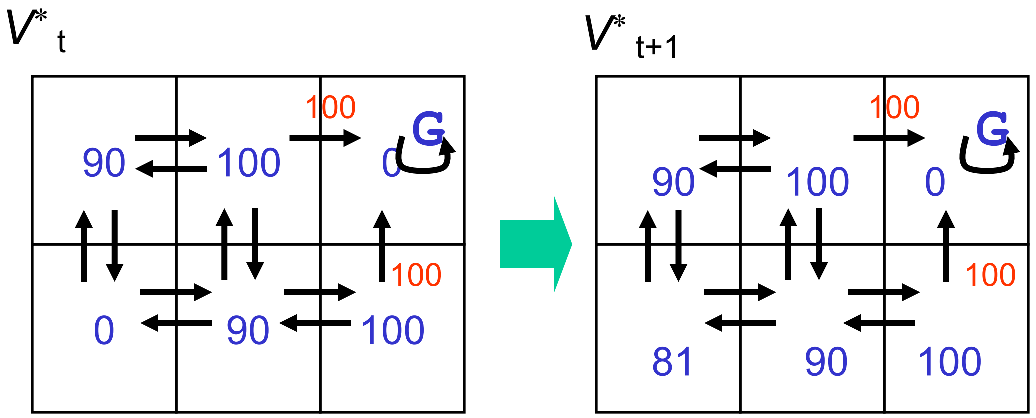
$$\gamma = 0.9$$



Example of Value Iteration

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

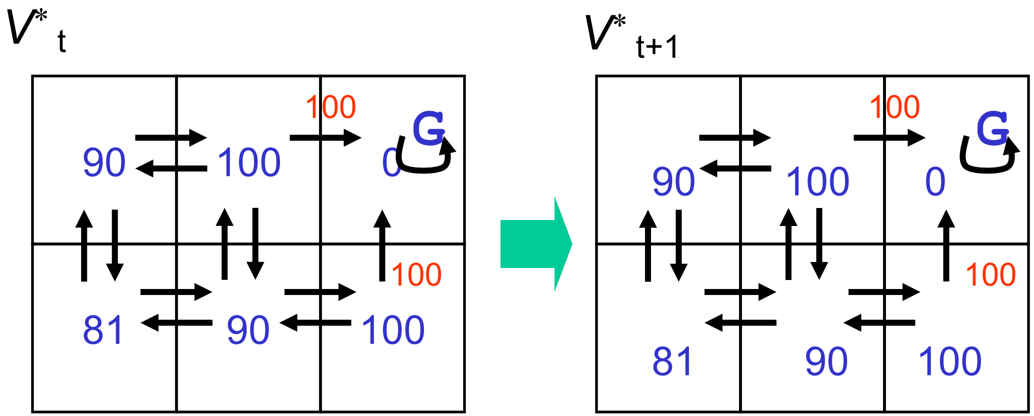
$\gamma = 0.9$



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
- Summary

Policy iteration

Idea: search for optimal policy and utility values simultaneously

Algorithm:

$\pi \leftarrow$ an arbitrary initial policy
repeat until no change in π
compute utilities given π
update π as if utilities were correct (i.e., local MEU)

To compute utilities given a fixed π:

$$U(i) = R(i) + \Sigma_j U(j) M_{ij}^{\pi(i)} \qquad \text{for all } i$$

i.e., n simultaneous linear equations in n unknowns, solve in $O(n^3)$

- Why use policy iteration? May converge faster; convergence guarantees.

Policy Iteration

Idea: Iteratively improve the policy

1. Policy Evaluation: Given a policy π_i calculate $V_i = V^{\pi i}$, the utility of each state if π_i were to be executed.
2. Policy Improvement: Calculate a new maximum expected 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.