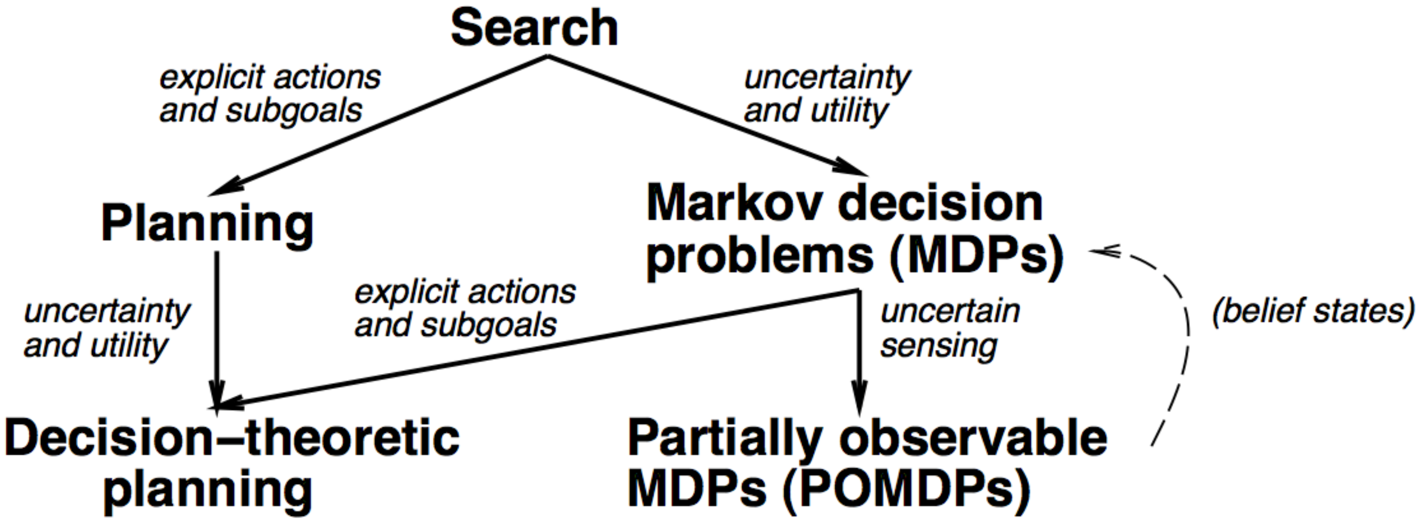
Probabilistic decision making

Markov Decision Processes (MDP) for Reinforcement Learning (RL)

- ◊ Decision problems
- ◊ Value iteration
- ◊ Policy iteration

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

Sequential decision problems



Sequential decision problems

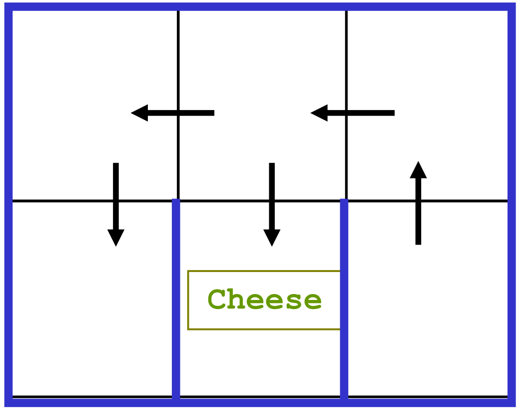
Add uncertainty to state-space search $\rightarrow$ MDP

Add sequentiality to Bayesian decision making $\rightarrow$ MDP
I.e., any environment in which rewards are not immediate

Examples:

- Tetris, spider solitaire
- Inventory and purchase decisions, call routing, logistics, etc. (OR)
- Elevator control
- Choosing insertion paths for flexible needles
- Motor control (stochastic optimal control)
- Robot navigation, foraging

How Might a Mouse Search a Maze for Cheese?



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

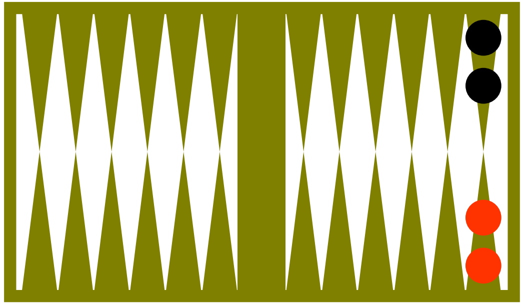
What is missing?

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.

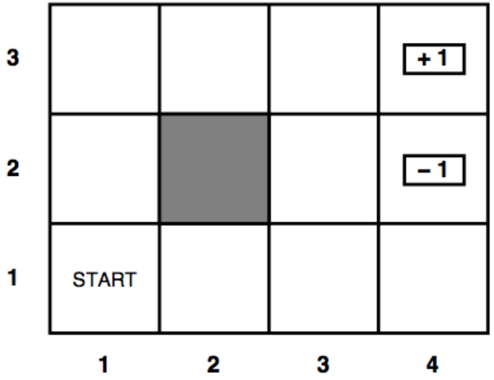
➔ Currently, roughly equal to best human player.

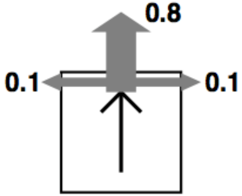
MDP Examples: Aerial Robotics [Feron et al.]
Computing a Solution from a Continuous Model

Markov Decision Processes

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

Example MDP





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

Sometimes written
T(s, a, s'): prob of transition from s to s' due to a

Each state has a *reward* $R(i)$ or R(i, a), or R(i, a, i')

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

$= \pm 1$ for terminal states

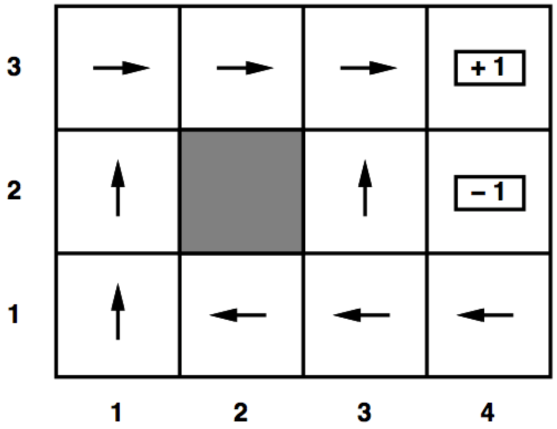
(reward is received when agent executes an action)

Example MDP

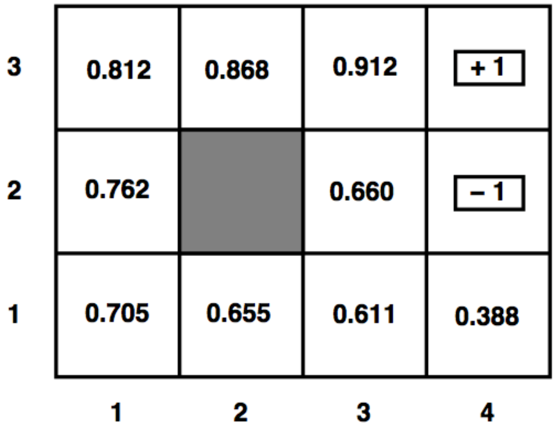
In search problems, aim is to find an optimal *sequence*

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 $R(i)$:

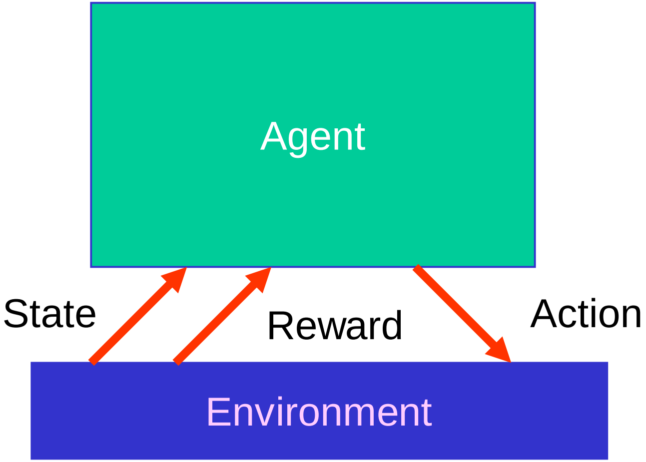


Policy's actions



Computed values of states

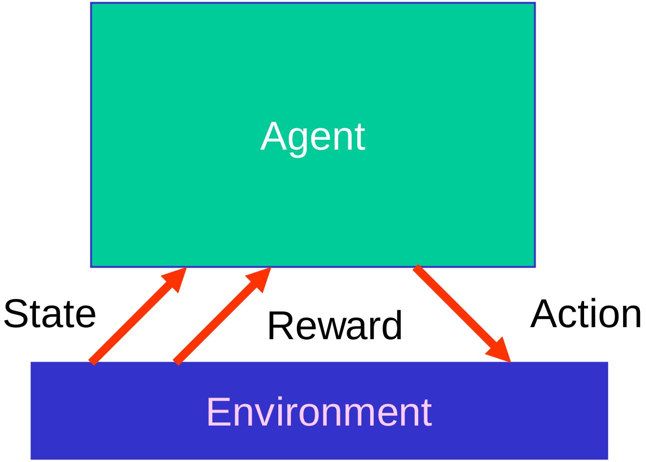
MDP Problem



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

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)

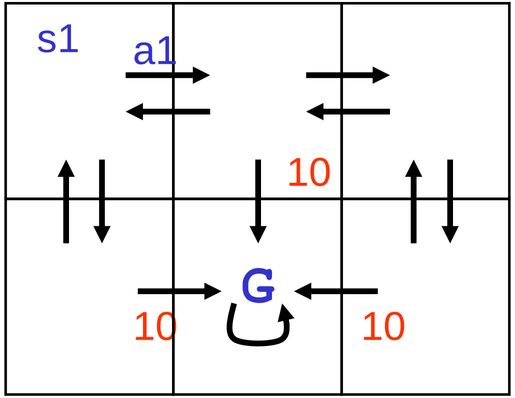
Model:

- 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:



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

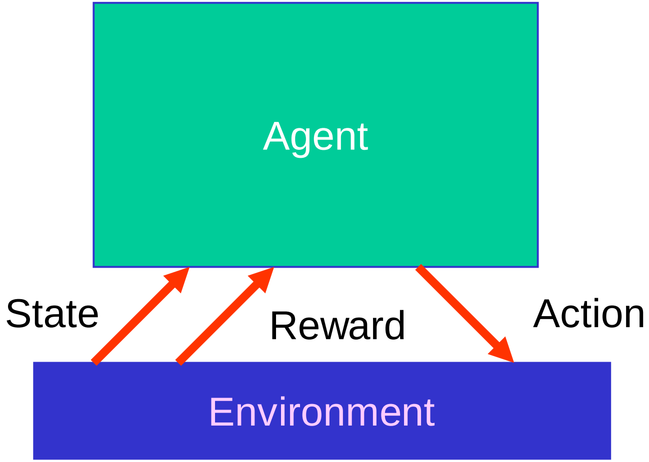
- Legal transitions shown
- Reward on unlabeled transitions is 0.

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

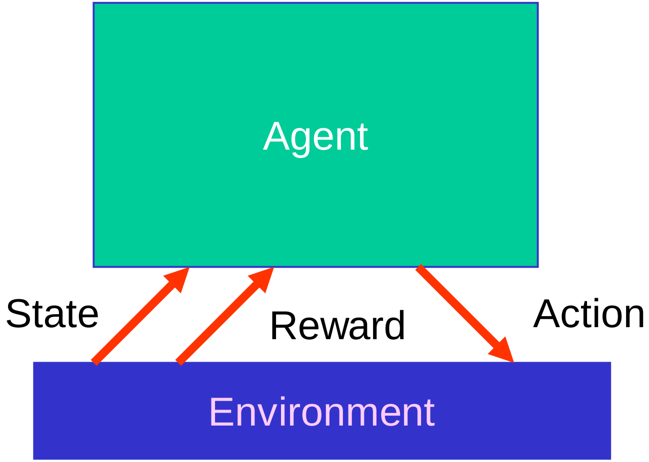
MDP Problem: Model



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

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.

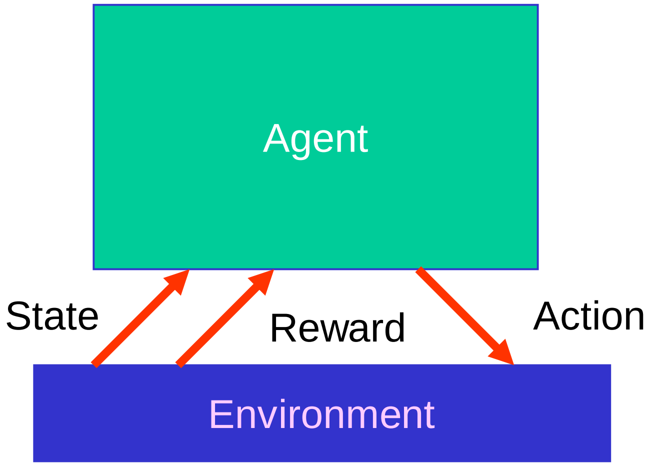
Lifetime Reward

- 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 for $\gamma<1$
  - Will make the math work

MDP Problem: Lifetime Reward

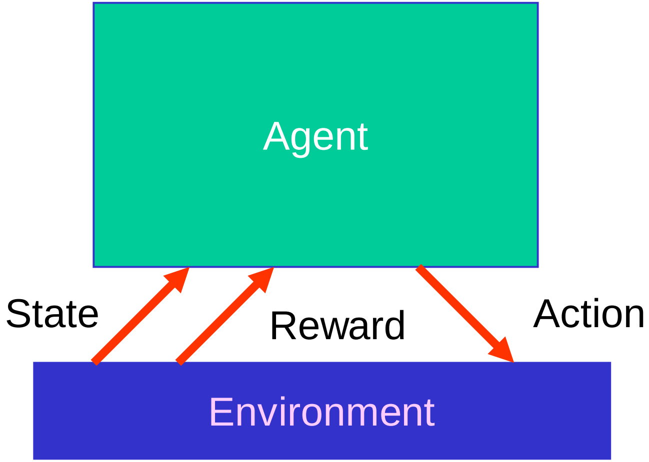


$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3$$

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 <u>policy</u> for acting that maximizes lifetime reward

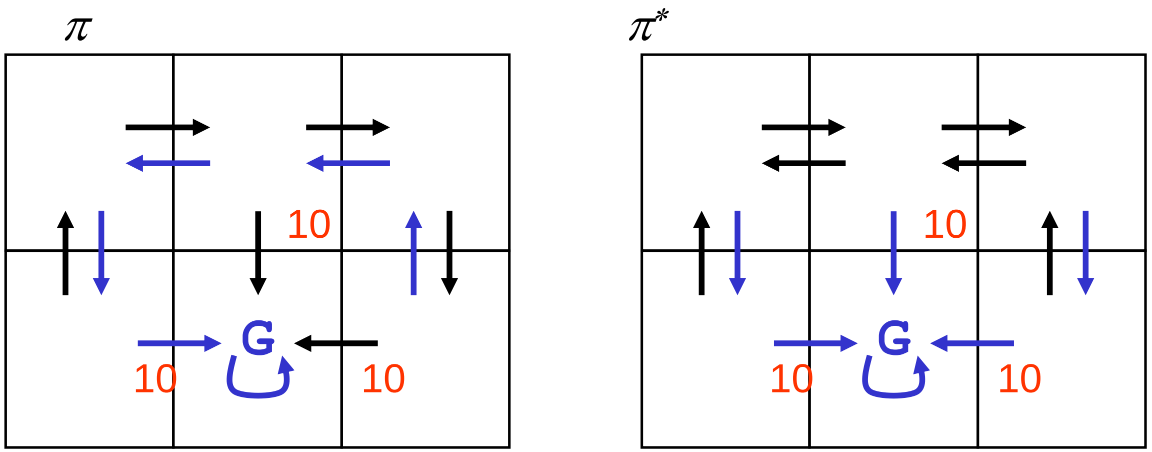
$$V = r_0 + \gamma\, r_1 + \gamma^2 r_2 \; . \; . \; .$$

Policy $\pi : S \rightarrow A$

- Selects an action for each state.

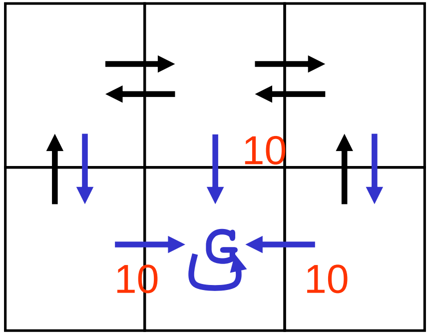
Optimal policy $\pi^* : S \rightarrow A$

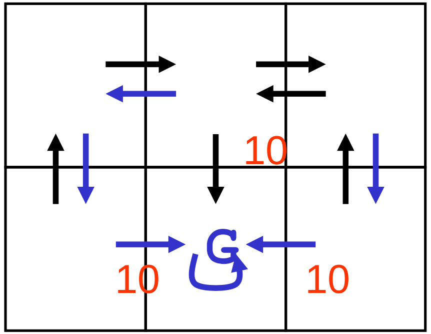
- Selects action for each state that maximizes lifetime reward.

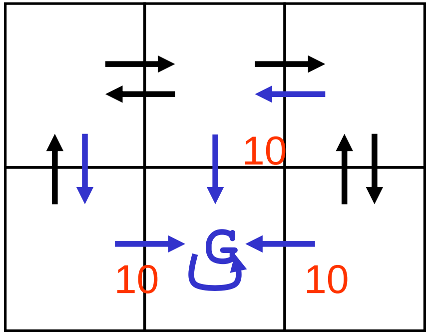


Note: with infinite horizon, policy is stationary and independent of start state

- 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

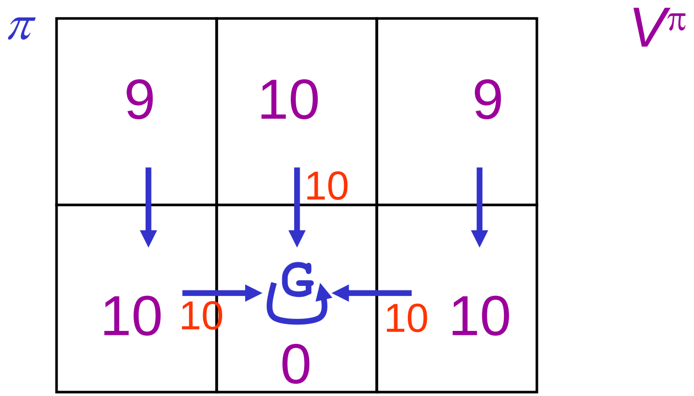
Value Function V^π for a Given Policy π

- $V^\pi(s_t)$ 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 $\gamma = .9$



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:

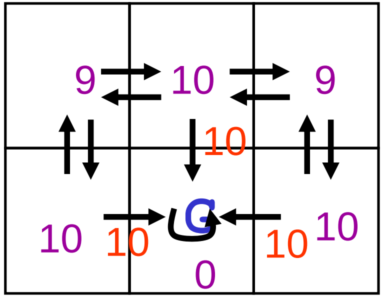
- the immediate reward for taking action: $r(s,a)$ …
- plus lifetime reward starting in target state: $V(\delta(s, a))$ …
- discounted by γ.

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

Must Know:

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

π

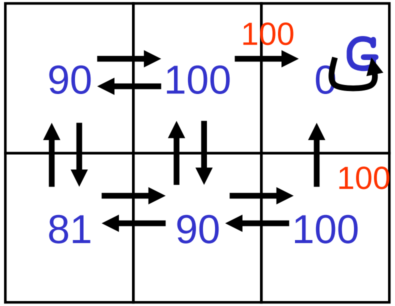


Example: Mapping Value Function to Policy

- Agent selects optimal action from V:

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

Model + V:



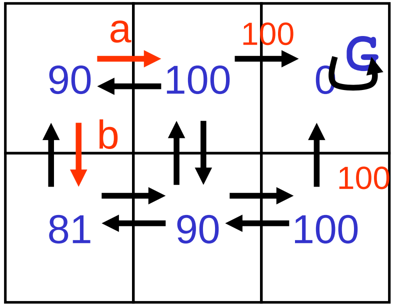
$\gamma = 0.9$

Example: Mapping Value Function to Policy

- Agent selects optimal action from *V*:

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

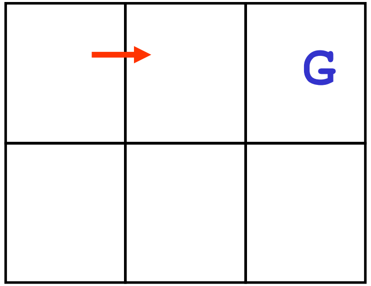
Model + V:



$\gamma = 0.9$

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

π:



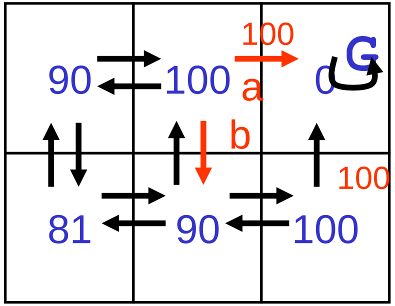
Example: Mapping Value Function to Policy

- Agent selects optimal action from *V*:

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

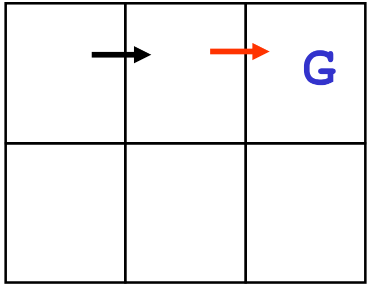
Model + V:

$\gamma = 0.9$



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

π:

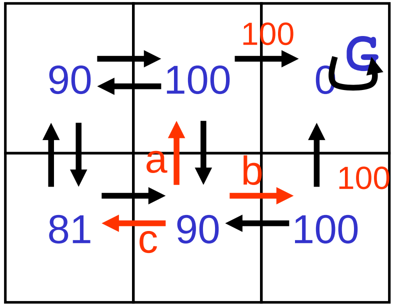


Example: Mapping Value Function to Policy

- Agent selects optimal action from V:

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

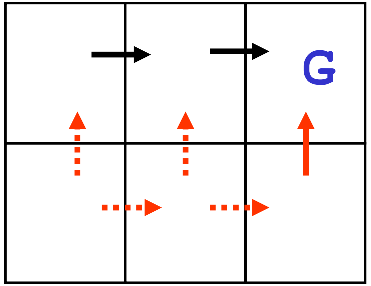
Model + V:



$\gamma = 0.9$

- a: ?
- b: ?
- c: ?
- ➢ select ?

π:

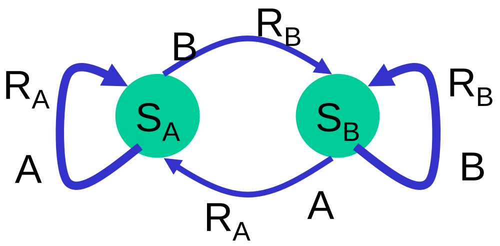


Markov Decision Processes

- 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

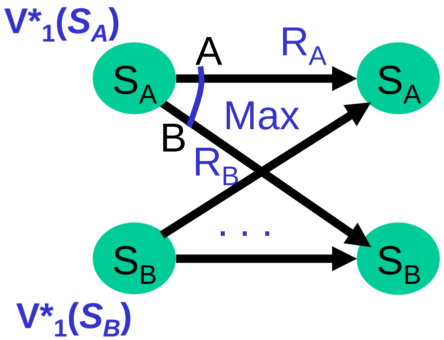
Value Function V^* for an optimal policy π^*

Example



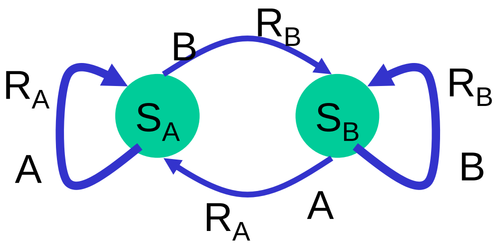
- Optimal value function for a one step horizon:

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



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

Example



- Optimal value function for a one step horizon:

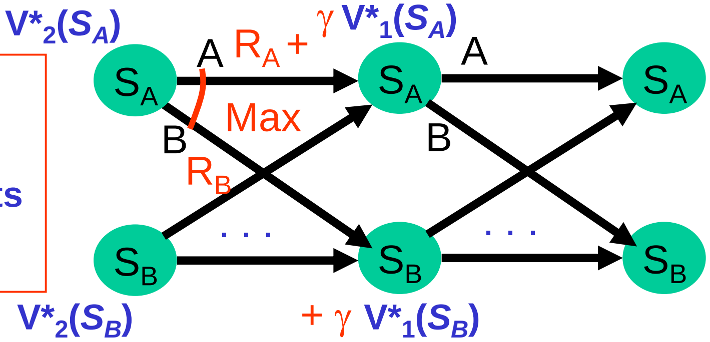
$$V^*_1(s) = \max_{a_i} [r(s,a_i)]$$

- Optimal value function for a two step horizon:

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

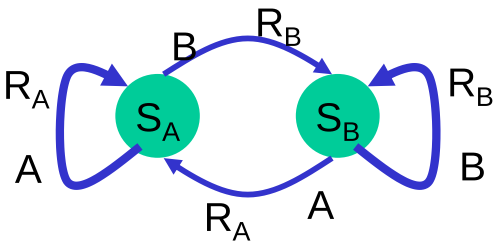
Instance of the Dynamic Programming Principle:

- **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:

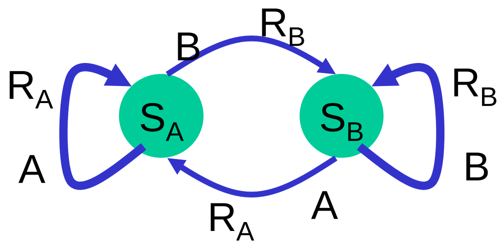
$$V^*_2(s) = \max_{a_i} [r(s,a_i) + \gamma V_1^*(\delta(s, a_i))]$$

- 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:

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

Bellman equation

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

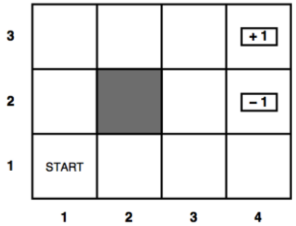
expected sum of rewards
= current reward
+ expected sum of rewards after taking best action

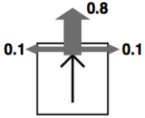
Bellman equation (1957):

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

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

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





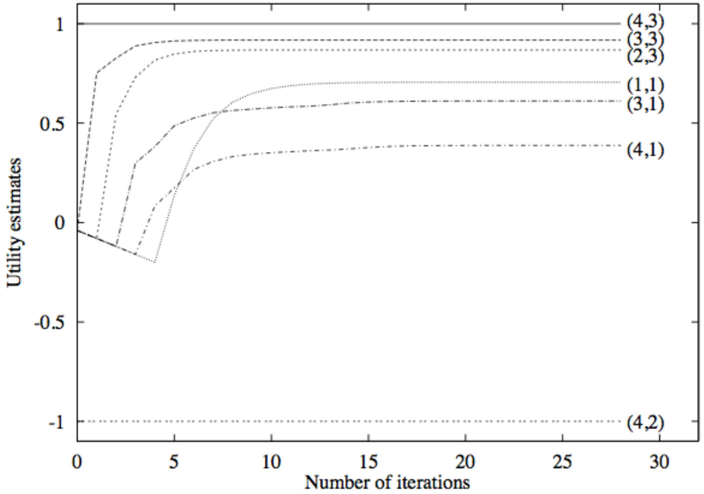
One equation per state = n nonlinear equations in n unknowns

Value iteration algorithm

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

repeat until "no change"

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



Convergence

Define the max-norm $||U|| = \max_s |U(s)|$,
so $||U - V||$ = maximum difference between U and V

Let U^t and U^{t+1} be successive approximations to the true utility U

Theorem: For any two approximations U^t and V^t

$$||U^{t+1} - V^{t+1}|| \leq \gamma \, ||U^t - V^t||$$

I.e., Bellman update is a **contraction**: any distinct approximations must get closer to each other
so, in particular, any approximation must get closer to the true U
and value iteration converges to a unique, stable, optimal solution

But MEU policy using U^t may be optimal long before convergence of values

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"

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

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

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

Convergence of Value Iteration

- If terminate when values are "close enough"

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

Then:

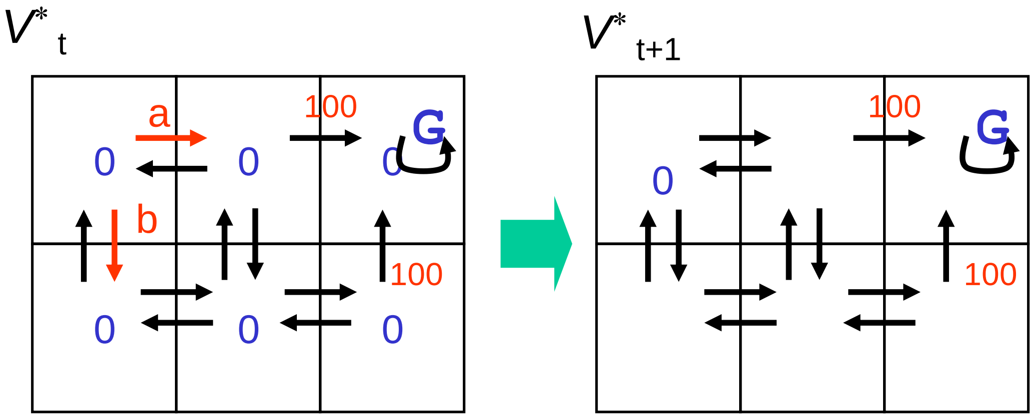
$$\mathrm{Max}_{s \text{ in } S} |V_{t+1}(s) - V^*(s)| < 2\varepsilon\gamma/(1 - \gamma)$$

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

Example of Value Iteration

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

$\gamma = 0.9$

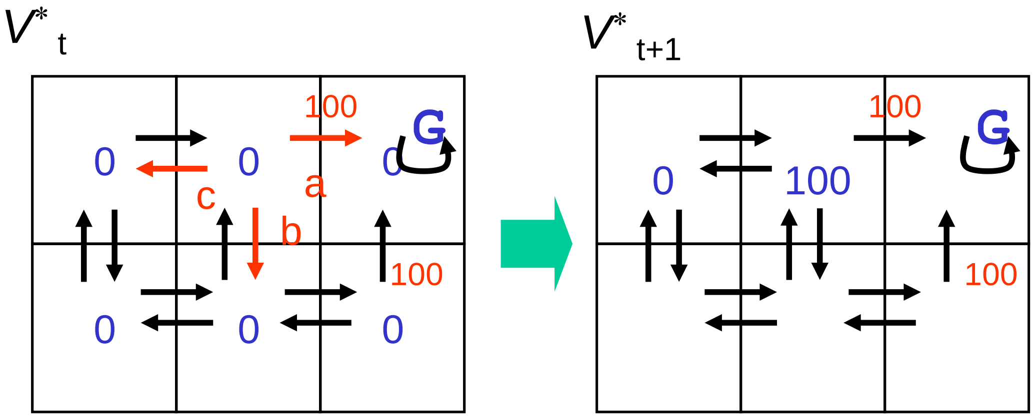


- a: 0 + 0.9 x 0 = 0
- b: 0 + 0.9 x 0 = 0
- ➢ Max = 0

Example of Value Iteration

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

$\gamma = 0.9$



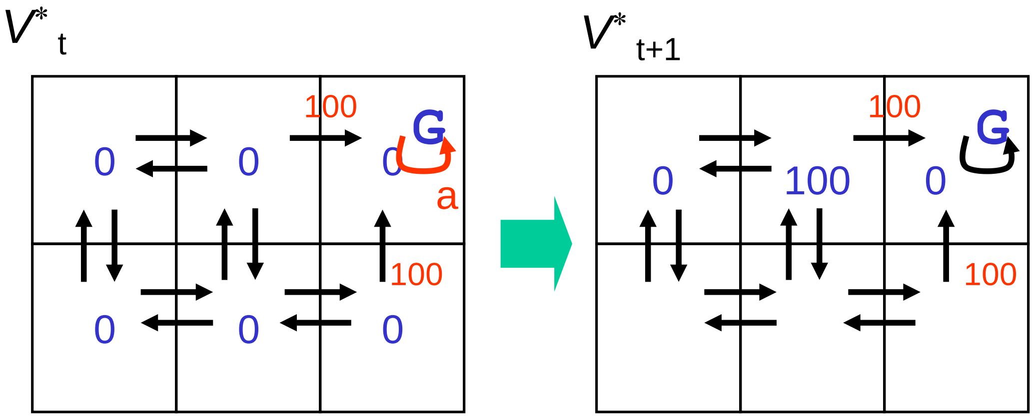
- a: 100 + 0.9 x 0 = 100
- b: 0 + 0.9 x 0 = 0
- c: 0 + 0.9 x 0 = 0

➢ Max = 100

Example of Value Iteration

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

$\gamma = 0.9$

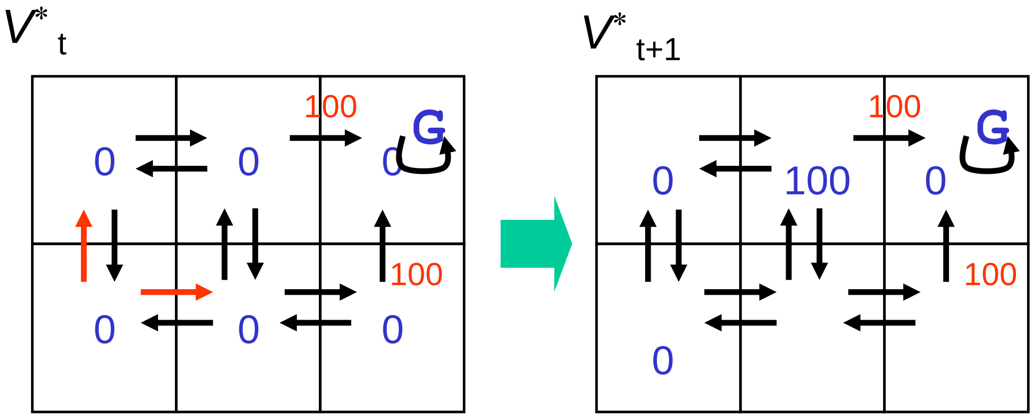


- a: 0 + 0.9 x 0 = 0
  - Max = 0

Example of Value Iteration

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

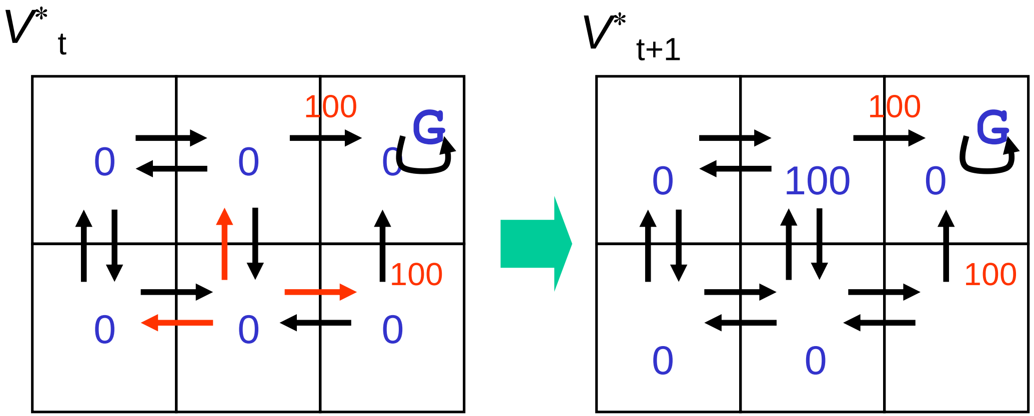
$\gamma = 0.9$



Example of Value Iteration

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

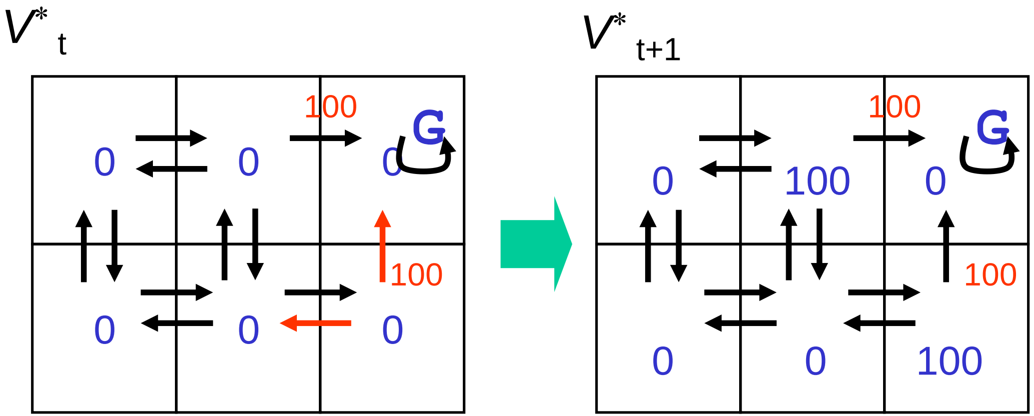
$\gamma = 0.9$



Example of Value Iteration

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

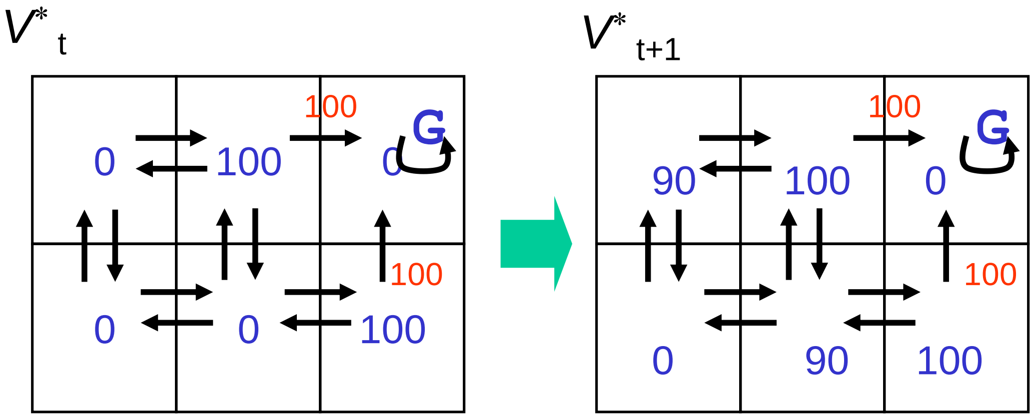
$\gamma = 0.9$



Example of Value Iteration

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

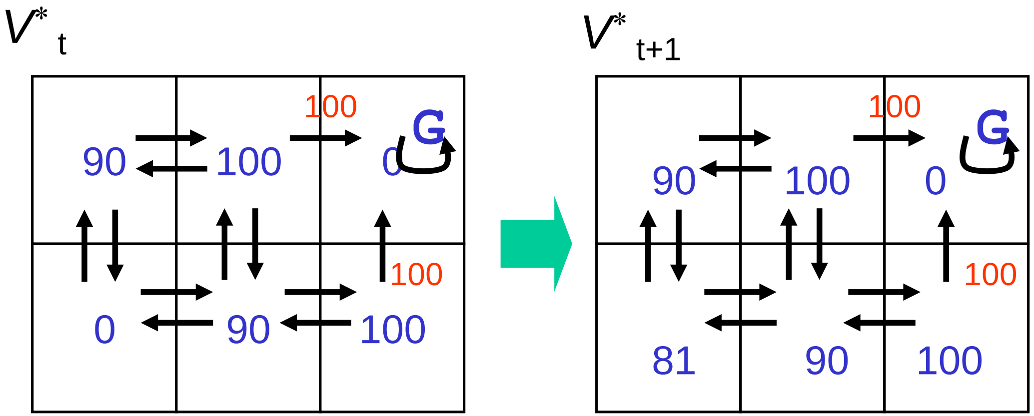
$\gamma = 0.9$



Example of Value Iteration

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

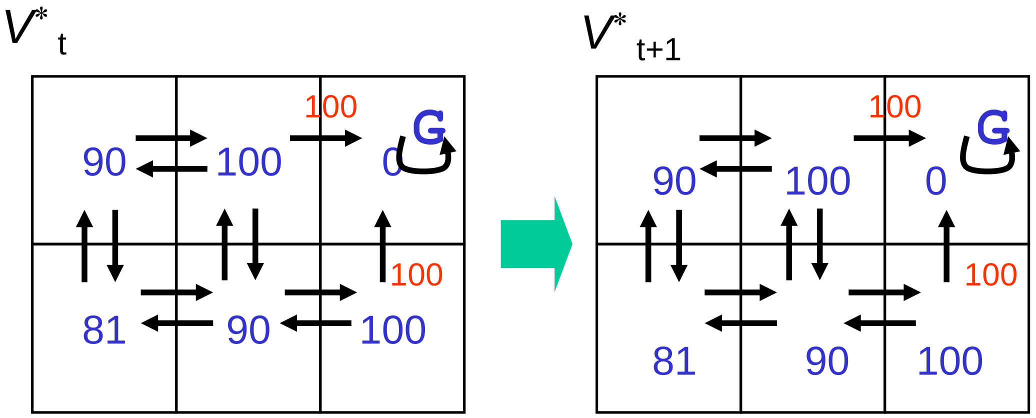
$\gamma = 0.9$



Example of Value Iteration

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

$\gamma = 0.9$



No more update = converged

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.

POMDP

POMDP has an observation model $O(s, e)$ defining the probability that the agent obtains evidence e when in state s

Agent does not know which state it is in
$\Rightarrow$ makes no sense to talk about policy $\pi(s)$!!

Theorem (Astrom, 1965): the optimal policy in a POMDP is a function $\pi(b)$ where b is the belief state ($P(S|e_1, \ldots, e_t)$)

Can convert a POMDP into an MDP in (continuous, high-dimensional) belief-state space,
where $T(b, a, b')$ is essentially a filtering update step

Solutions automatically include information-gathering behavior

The real world is an unknown POMDP

Issues

Complexity: polytime in number of states (by linear programming)
but number of states is exponential in number of state variables
$\rightarrow$ Boutilier *et al*, Parr & Koller: use structure of states
(but U, Q summarize infinite sequences, depend on everything)
$\rightarrow$ reinforcement learning: sample S, approximate $U/Q/\pi$
$\rightarrow$ hierarchical methods for policy construction (next lecture)

Unknown transition model: agent cannot solve MDP w/o $T(s, a, s')$
$\rightarrow$ reinforcement learning

Missing state: there are state variables the agent doesn't know about
$\rightarrow$ [your ideas here]

Reinforcement learning

Agent is in an unknown MDP or POMDP environment

Only feedback for learning is percept + reward

Agent must learn a policy in some form:

- transition model $T(s, a, s')$ plus value function $U(s)$
- action-value function $Q(a, s)$
- policy $\pi(s)$

Deep reinforcement learning

Sometimes, dimensionality of state is very high (e.g., images)

-> Represent and learn the policy as a deep neural network.