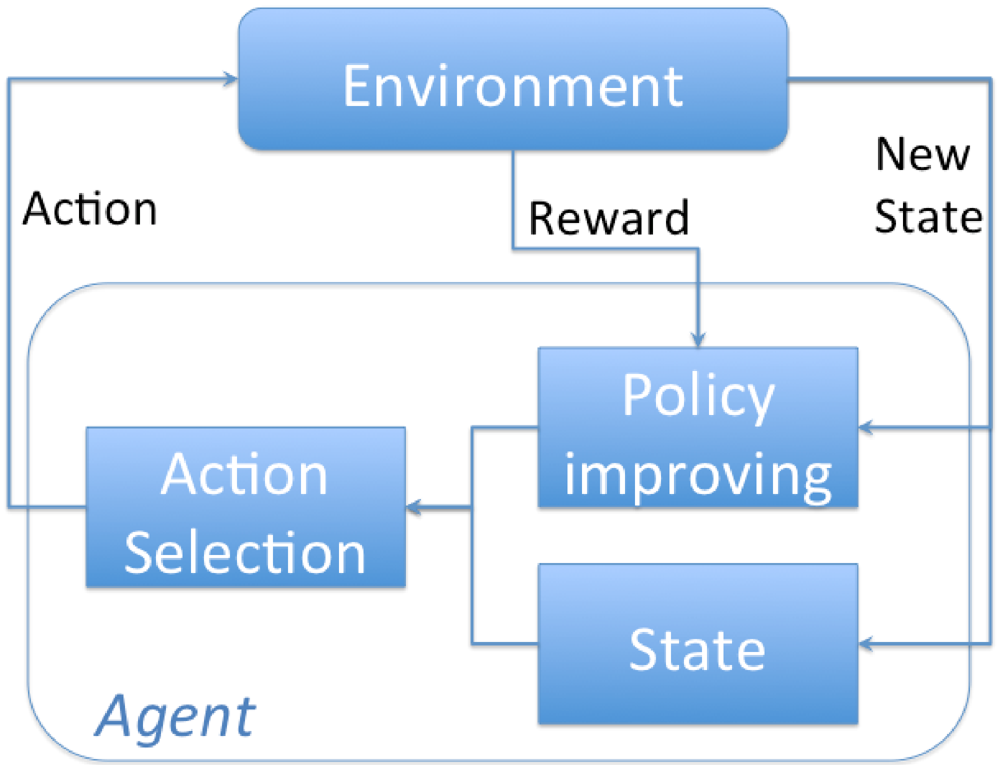
MARKOV DECISION PROCESSES (MDP) AND REINFORCEMENT LEARNING (RL)

Versione originale delle slide fornita dal Prof. Francesco Lo Presti

Historical background

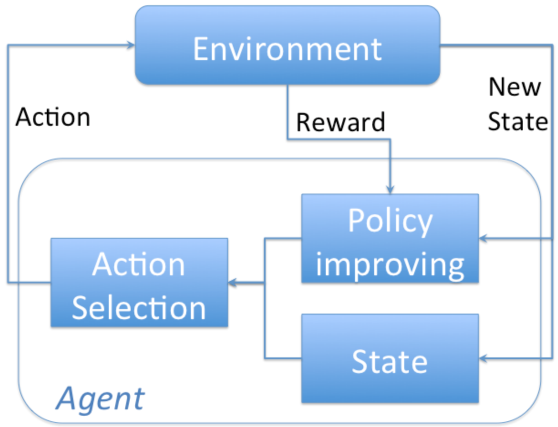
- Original motivation: animal learning
- Early emphasis: neural net implementations and heuristic properties
- Now appreciated that it has close ties with
  - operations research
  - optimal control theory
  - dynamic programming
  - AI state-space search
- Best formalized as a set of techniques to handle: Markov Decision Processes (MDPs) or Partially Observable Markov Decision Processes (POMDPs)

Reinforcement learning task



Reinforcement learning task

$$s(0) \xrightarrow[]{\substack{a(1)\\ r(1)}} s(1) \xrightarrow[]{\substack{a(2)\\ r(2)}} s(2) \xrightarrow[]{\substack{a(3)\\ r(3)}} s(3)\ldots$$



- **Goal:** learn to choose actions that maximize the cumulative reward

$$r(0) + \gamma r(1) + \gamma r(2) + \gamma r(3) + \ldots$$

where $0 \le \gamma < 1$

Foresighted Optimization

- The key feature of this framework is that actions affect *immediate* and *future* system performance
  - We optimize the system "on the long run"
- In scenarios in which actions/decisions affect immediate and future performance, myopic heuristic solutions are suboptimal because they ignore the expected future utility
- Dramatic improvements can be achieved using *long term* optimization

Stochastic Process

- Quick definition: a random process
- Often viewed as a collection of indexed random variables $\{X_t : t \in T\}$
  - $X_t \in S$ is the system state
  - t is the time
- Useful to characterize "environment" dynamics
  - Set of states with probability law governing evolution of states over time
- We will focus on *discrete-time* stochastic *chains*
  - Time is discrete
  - State set S is discrete

Stochastic Process Dynamics

- The process *dynamics* can be defined using the transition probabilities
- They specify the stochastic evolution of the process through its states
- For a discrete time process, transition probabilities can be defined as follows

$$P(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0)$$

Markov Property

- The term **Markov property** refers to the memoryless property of a stochastic process:
- For a discrete time process, the Markov property is defined as:

$$P(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \dots, X_0 = x_0)$$
$$=$$
$$\boldsymbol{P(X_{t+1} = x_{t+1} \mid X_t = x_t)}$$

- **Definition:** a stochastic process that satisfies the Markov property is called Markov process
- If the state space is discrete, we refers to these processes as Markov Chains

Markov Property

- Why useful?
  - Simple model of temporal correlation of environment dynamics
  - Current state contains all information needed to predict distribution of future state(s)
- Does it hold in the real world?
- It's an ideal
  - Will allow us to prove properties of algorithms
  - Algorithms mostly still work

Markov Chain

- Let $\{X_t : t = 1,2,3, \dots\}$ be a Markov chain
- Dynamics is defined using the transition probability function

$$P(X_{t+1} = s' | X_t = s)$$
$$t \geq 0, \qquad s', s \in S$$

- Under some assumptions (see previous lesson), a Markov chain has a *stationary* transition probability function

$$\pi_j = \lim_{k \to \infty} \pi_j(k)$$

Markov Decision Processes

- Finite set of states S
- Finite set of actions $A(s), s \in S$
- Immediate reward function

$$R\colon S \times A \to R$$

- Transition (next-state) function

$$T\colon S \times A \to S$$

- More generally, R and T are treated as stochastic
  - We'll stick to the above notation for simplicity
  - In general case, treat the immediate rewards and next states as random variables, take expectations, etc.
- We will focus on *discrete time* MDPs

Markov Decision Processes

- Markov Property for MDPs

$$P(s' \mid s, a)$$
$$s', s \in S, \qquad a \in A(s)$$

- Next state is a function of current *state* and the *action taken*!

Markov Decision Processes

- If no rewards and only one action, this is just a *Markov chain*
- Sometimes also called a Controlled Markov Chain
- Overall objective is to determine a policy

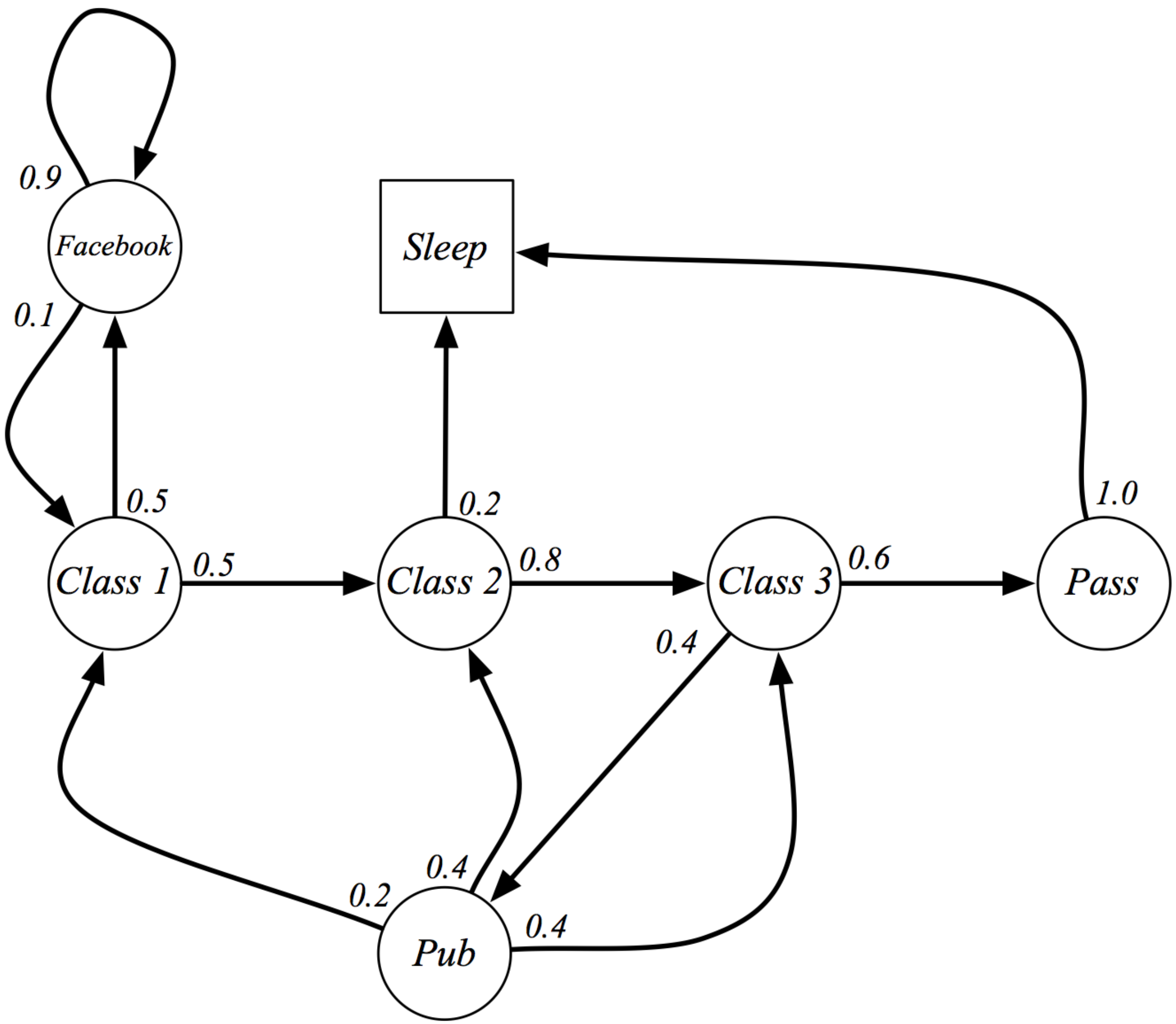
$$\pi : S \rightarrow A$$

  such that some measure of cumulative reward is optimized
  - E.g, $r(0) + \gamma r(1) + \gamma r(2) + \gamma r(3) + \ldots$

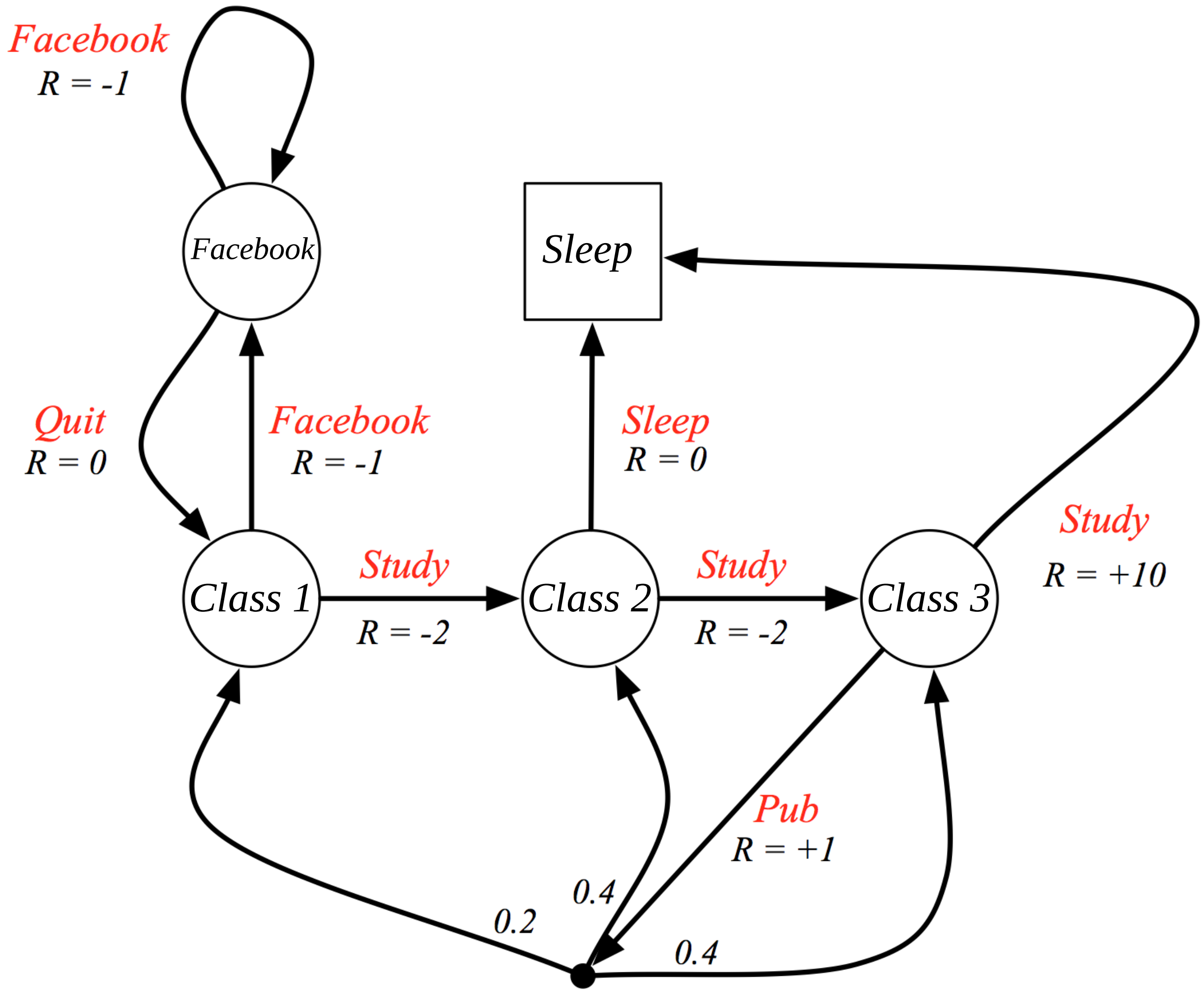
What's a policy?

If agent is in this state	Then a good action is
s_1	a_3
s_2	a_7
s_3	a_1
s_4	a_3
. . .	. . .

Student Markov Chain



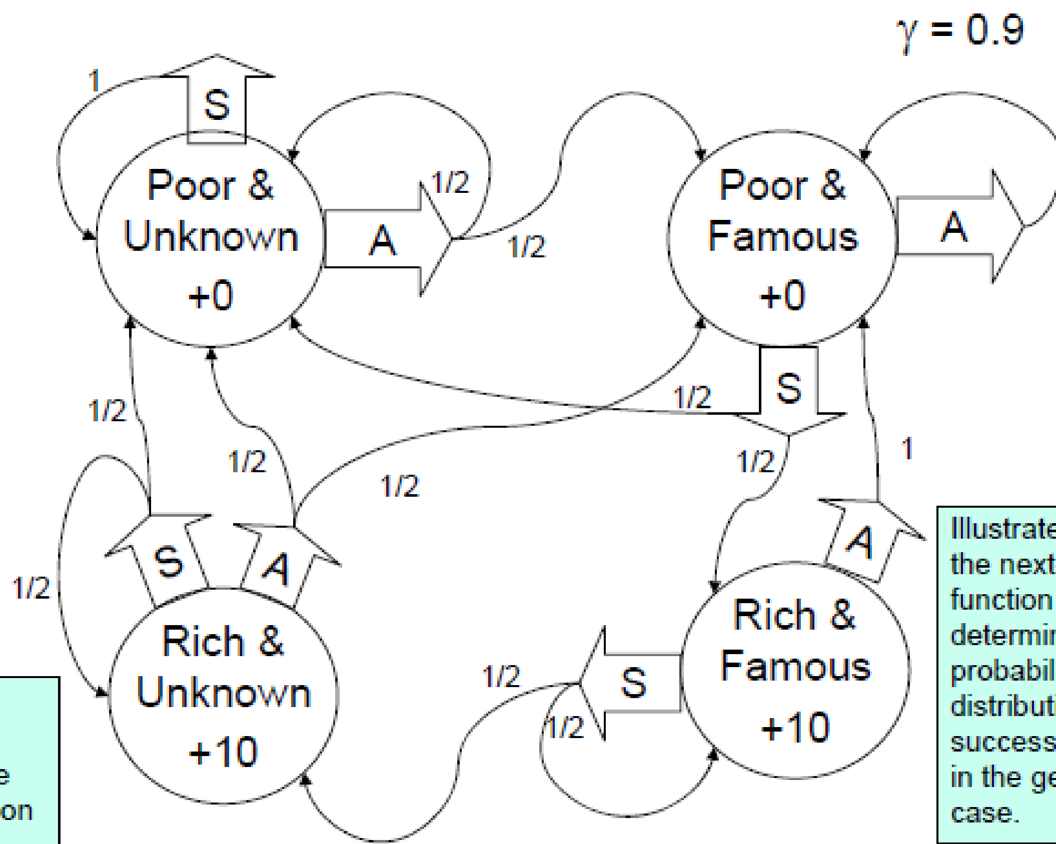
Student Markov Decision Process



A Markov Decision Process

You run a startup company.

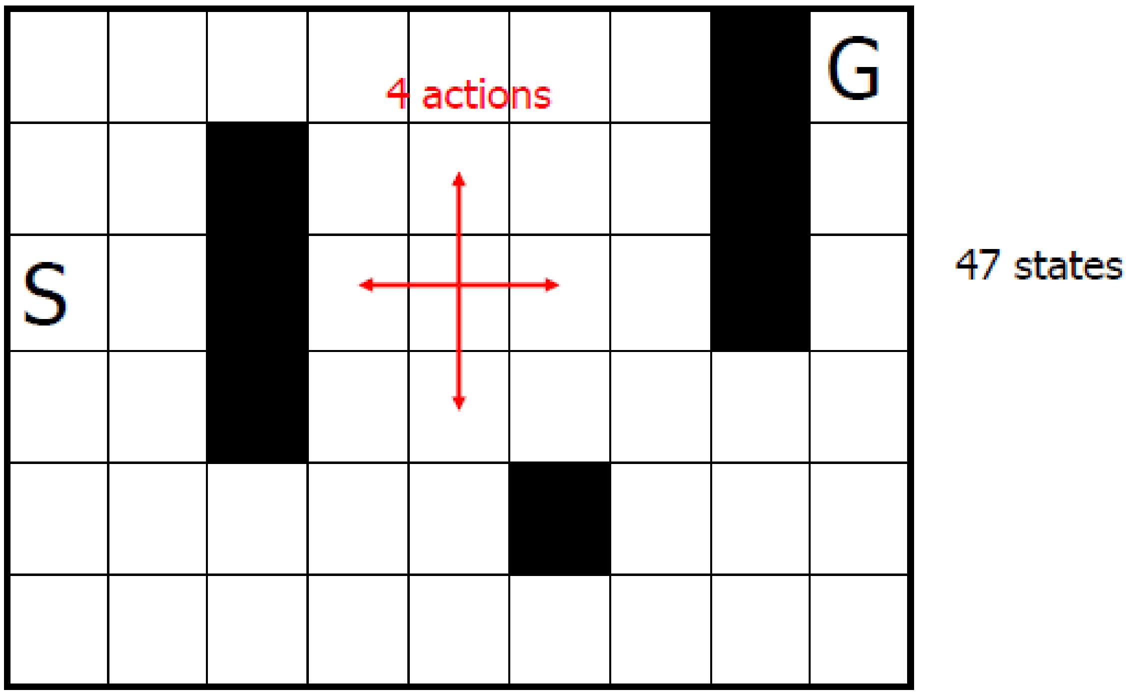
In every state you must choose between Saving money or Advertising.



Here the reward shown inside any state represents the reward received upon entering that state.

Illustrates that the next-state function really determines a probability distribution over successor states in the general case.

Another MDP



Reward = -1 at every step $\gamma = 1$

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic

Applications of MDPs

Many important problems are MDPs….

… Robot path planning

… Travel route planning

… Elevator scheduling

… Autonomous aircraft navigation

… Manufacturing processes

… Network switching & routing

And many of these have been successfully handled using RL methods

Brief summary of concepts

- The *agent* and its *environment* interact over a sequence of discrete time steps

- The specification of their interface defines a particular task:
  - the *actions* are the choices made by the agent
  - the *states* are the basis for making the choices
  - the *rewards* are the basis for evaluating the choices

- A *policy* is a stochastic rule by which the agent selects actions as a function of states

- The agent's objective is to maximize the amount of reward it receives over time

Value Functions

- It turns out that
  - RL theory
  - MDP theory
  - AI game-tree search

  all agree on the idea that evaluating states is a useful thing to do.
- A (state) value function V is any function mapping states to real numbers:

$$V : S \rightarrow \mathbb{R}$$

What's a value function?

If agent starts in this state	Return when following given policy should be
s_1	13
s_2	-1
s_3	22.6
s_4	6
. . .	. . .

A special value function: the return

- For any policy π, define the return to be the function $V^{\pi}\colon S \to \mathbb{R}$ assigning to each state the quantity

Reminder: Use expected values in the stochastic case.

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^{t} r(t)$$

where

- $s(0) = s$
- each action $a(t)$ is chosen according to policy π
- each subsequent $s(t+1)$ arises from the transition function T
- each immediate reward $r(t)$ is determined by the immediate reward function R
- γ is a given discount factor in $[0, 1]$

Why the discount factor γ?

- Models idea that future rewards are not worth as much as immediate rewards
  - Used in economic models
  - Uncertainty about the future
- Also models situations where there is a nonzero fixed probability $1-\gamma$ of termination at any time
- Tradeoff between myopic ($\gamma = 0$) vs foresighted optimization (γ close to 1)
- …and makes the math work out nicely
  - with bounded rewards, sum guaranteed to be finite even in infinite-horizon case

Technical remarks

- If the next state and/or immediate reward functions are stochastic, then the $r(t)$ values are random variables and the return is defined as the expectation of this sum
- If the MDP has absorbing states, the sum may actually be finite
  - In that case $\gamma = 1$ is allowed, i.e., no discount
  - We stick with this infinite sum notation for the sake of generality
  - The formulation we use is called infinite-horizon

Optimal Policies

- Objective: Find a policy π^* such that

$$V^{\pi^*}(s) \geq V^{\pi}(s)$$

  for any policy π and any state s

- Such a policy is called an optimal policy

- We define:

$$V^* = V^{\pi^*}$$

Interesting fact

- For every MDP such that S is discrete and $A(s)$ is finite there exists an optimal policy
  - This theorem can be easily extended to the case in which $A(s)$ is a *compact* set
- It's a policy such that for every possible start state there is no better option than to follow the policy

Finding an Optimal Policy

- Idea One:

  Run through all possible policies.

  Select the best.

- What's the problem ??

Finding an Optimal Policy

- Dynamic Programming approach:
  - Determine the optimal value function for each state
  - Select actions according to this optimal value function V^*
- How do we compute V^*?
  - Magic words: Bellman equation(s)

Simple derivation of the Bellman equation

- Given the state transition $s \rightarrow s'$

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r(t)$$

$$= r(0) + \gamma \sum_{t=0}^{\infty} \gamma^t r(t+1)$$

$$= r(0) + \gamma V^{\pi}(s')$$

Bellman equations

- For any state s and policy π

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(T(s, \pi(s)))$$

- For any state s, the optimal value function is

$$V^{*}(s) = \max_{a}\{R(s, a) + \gamma V^{*}(T(s, a))\}$$

- Recurrence relations
  - Can be used to compute the return from a given policy or to compute the optimal return via value iteration

Bellman equations: general form

- For completeness, here are the Bellman equations for stochastic and discrete time MDPs:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P_{ss'}(\pi(s)) V^{\pi}(s')$$
$$V^{*}(s) = \max_{a} \{ R(s,a) + \gamma \sum_{s'} P_{ss'}(a) V^{*}(s') \}$$

where $R(s, a)$ now represents $E(R \mid s, a)$ and

$P_{ss'}(a)$ = probability that the next state is s' given that action a is taken in state s

From values to policies

- Given the optimal value function V^* it follows from Bellman equation that the optimal policy can be computed as:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\{R(s,a) + \gamma V^*(s')\}$$

- An optimal policy is said to be <u>greedy</u> for V^*

- If π is not optimal then a greedy policy for V^π will yield a larger return than π
  - Not hard to prove
  - Basis for another DP approach to finding optimal policies: policy iteration

Finding an optimal policy

<u>Value Iteration Method</u>

Choose any initial state value function V_0

Repeat for all $n \geq 0$

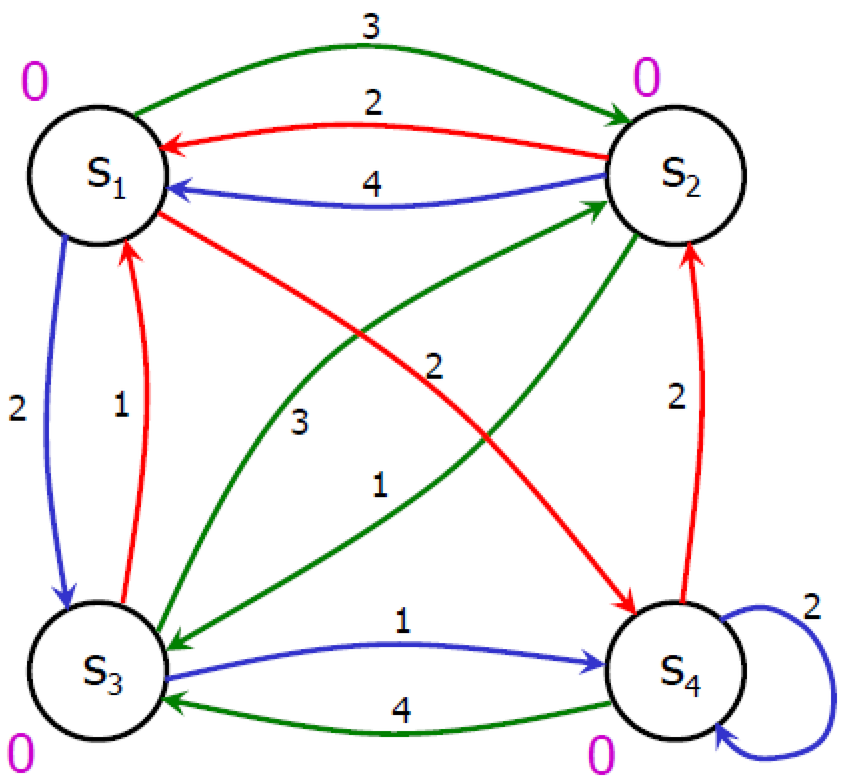
For all s

$$V_{n+1}(s) \leftarrow \max_{a}\{R(s,a) + \gamma V_n\big(T(s,a)\big)\}$$

Until convergence

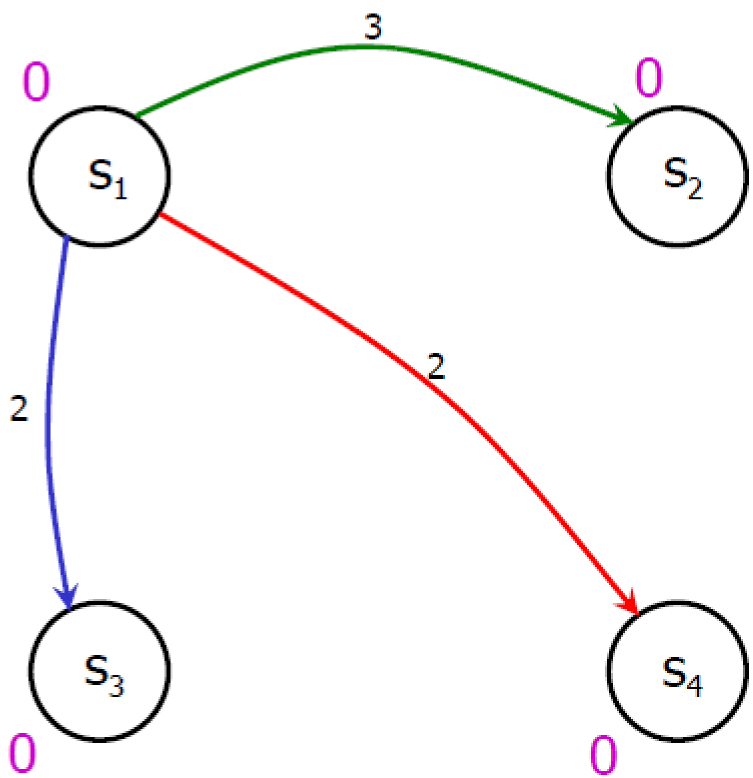
- This converges to V^* and any greedy policy with respect to it will be an optimal policy

Value Iteration



Arbitrary initial value function V_0

Value Iteration



Arbitrary initial value function V_0

Computing a new value for s_1 using 1-step lookahead with previous values:

For action a_1 lookahead value is $2 + (.9)(0) = 2$

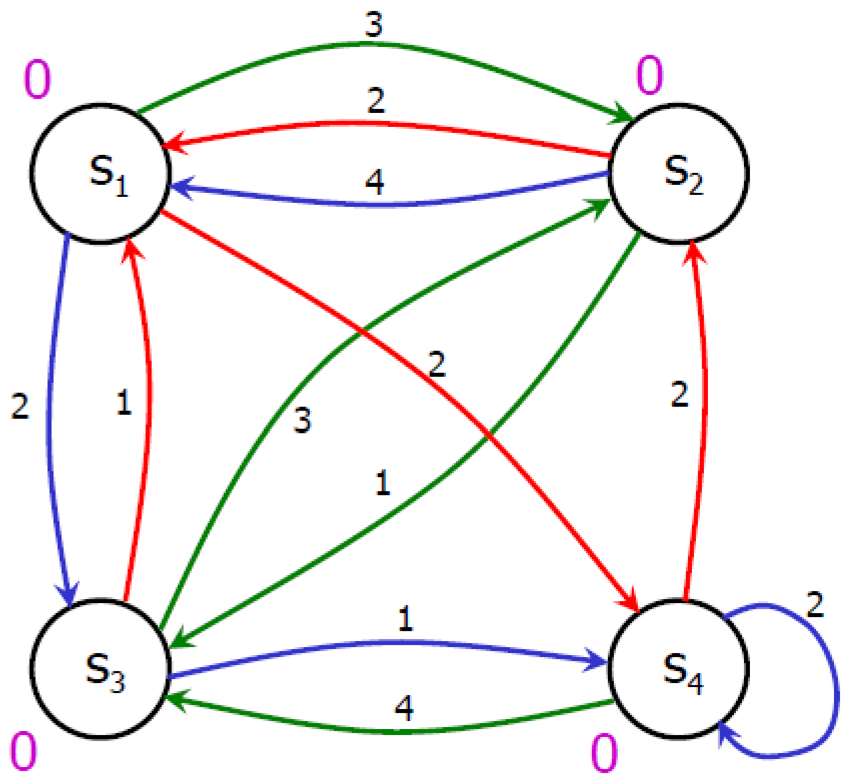
For action a_2 lookahead value is $3 + (.9)(0) = 3$

For action a_3 lookahead value is $2 + (.9)(0) = 2$

a_1	a_2	a_3
2	3	2

$$V_1(s_1) = \max\{2, 3, 2\} = 3$$

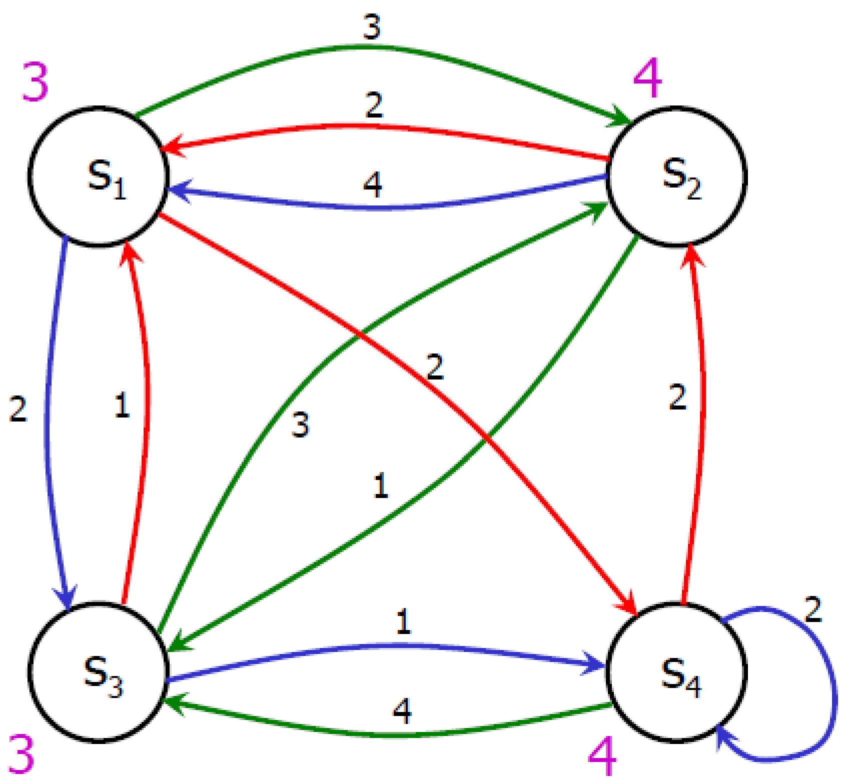
Value Iteration



Arbitrary initial value function V_0

Lookahead value along action				
	a_1	a_2	a_3	**max**
s_1	2	3	2	**3**
s_2	2	1	4	**4**
s_3	1	3	1	**3**
s_4	2	4	2	**4**

Value Iteration



Updated approximation to V*:

$V_1(s_1)=3$

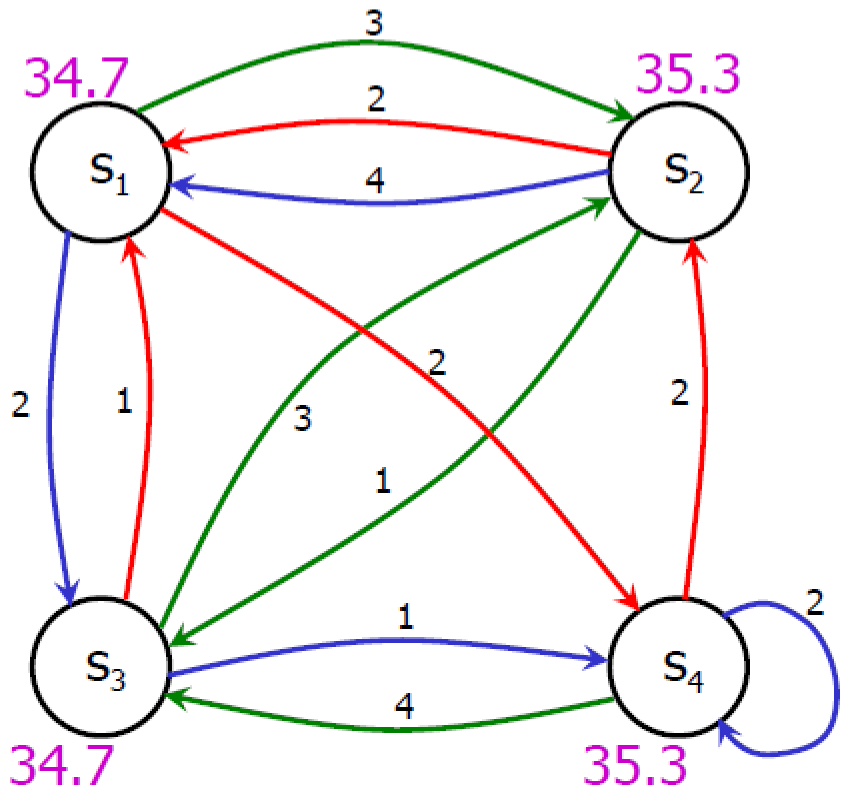
$V_1(s_2)=4$

$V_1(s_3)=3$

$V_1(s_4)=4$

New value function V_1 after one step of value iteration

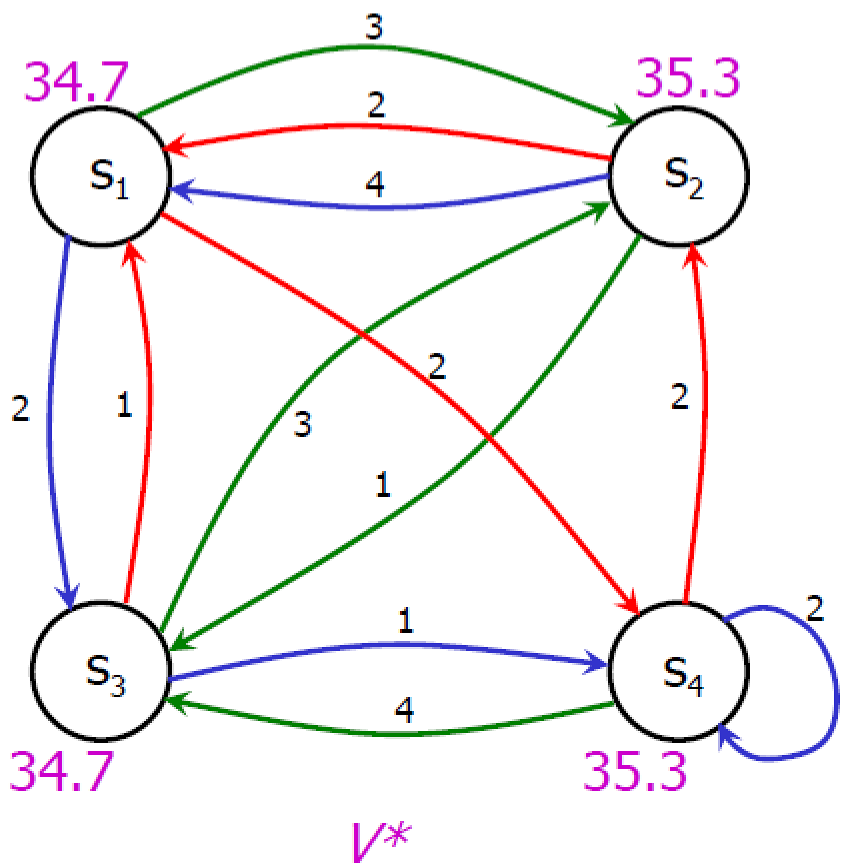
Value Iteration



	s_1	s_2	s_3	s_4
V_0	0	0	0	0
V_1	3	4	3	4
V_2	6.6	6.7	6.6	6.7
V_3	9.0	9.9	9.0	9.9
V_4	11.9	12.1	11.9	12.1
V_5	13.9	14.8	13.9	14.8
. . .				
V^*	34.7	35.3	34.7	35.3

Keep doing this until it converges to V^*

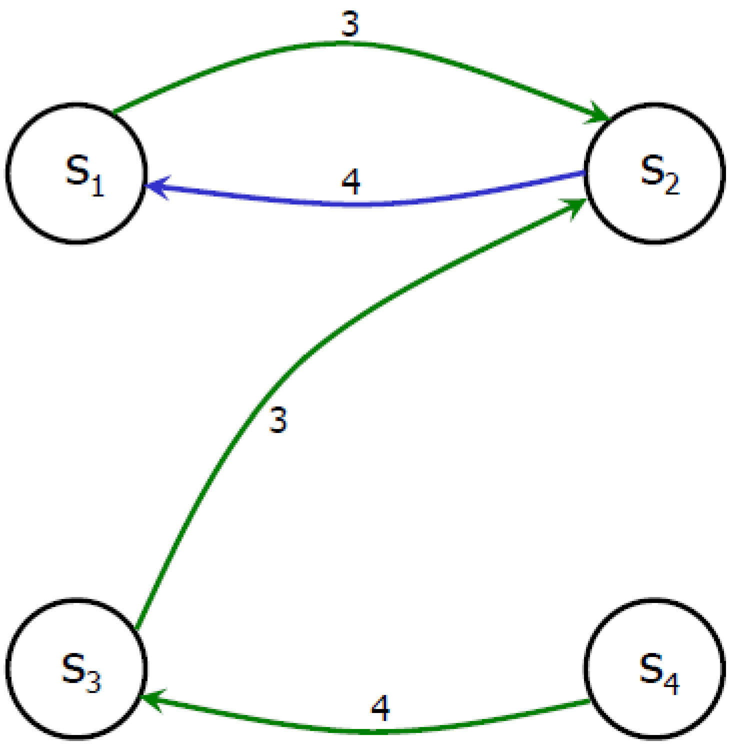
Value Iteration



Determining a greedy policy for $V*$

Lookahead value along action				
	a_1	a_2	a_3	**best**
s_1	33.8	34.8	33.2	a_2
s_2	33.2	32.2	35.2	a_3
s_3	32.2	34.8	32.8	a_2
s_4	33.8	35.2	33.8	a_2

Value Iteration



Optimal policy

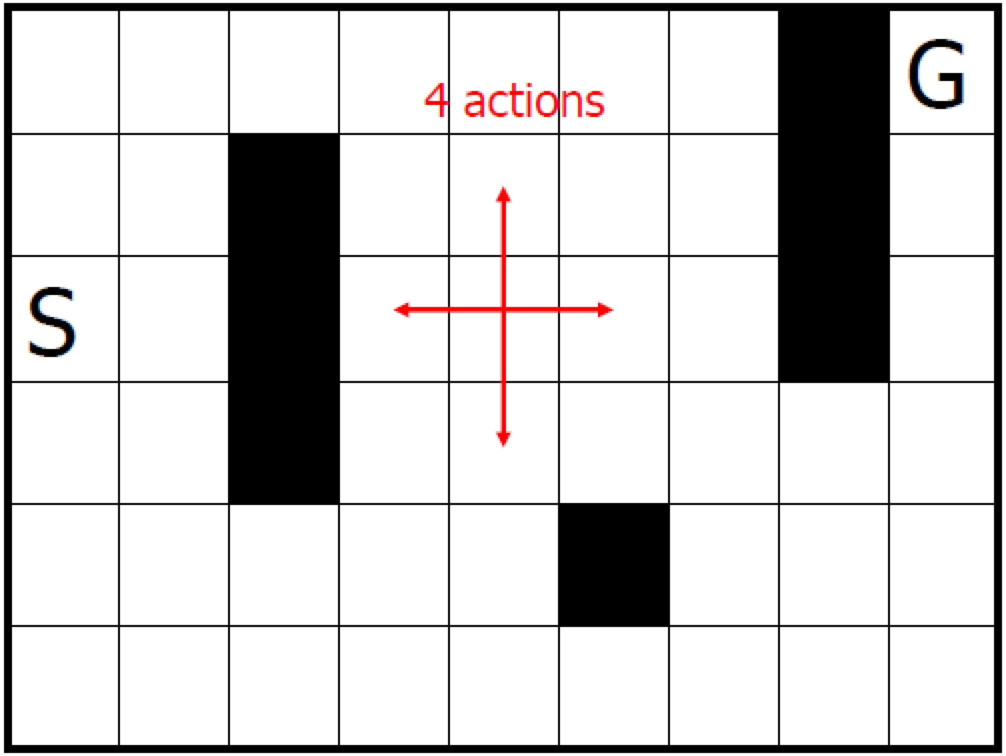
Value iteration – Full Version

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

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
    $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Maze Task

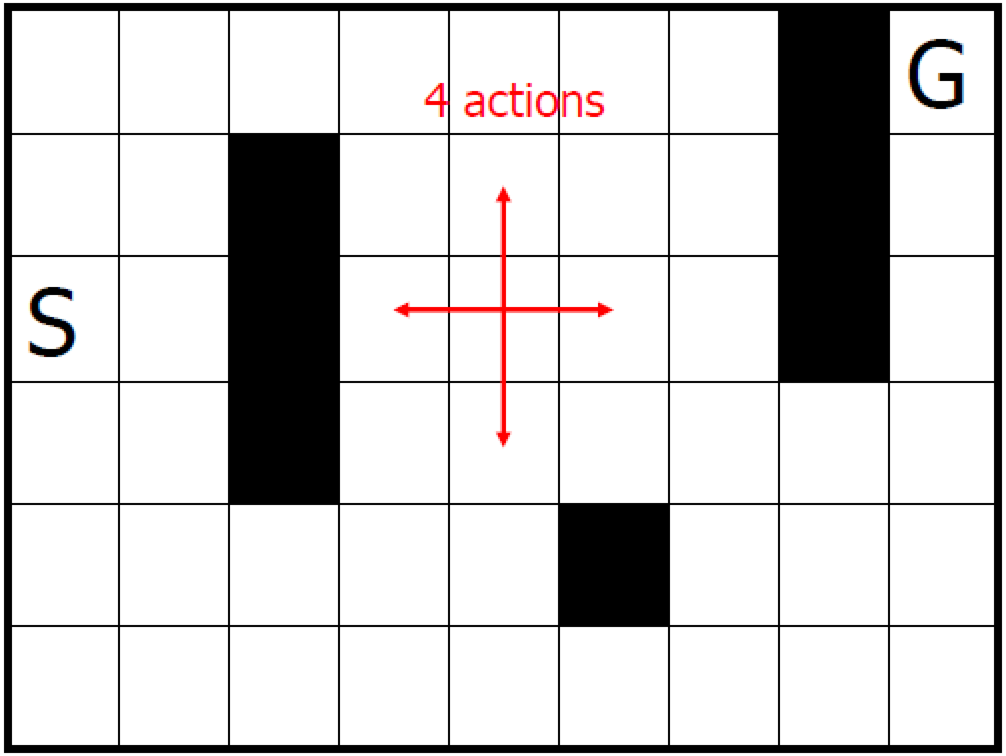


Reward = -1 at every step $\gamma = 1$

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic

Maze Task



How would you model the MDP?

Reward = -1 at every step $\gamma = 1$

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic

Maze Task – MDP model

- State is a couple: $s = (x, y), \quad x, y \in \{1, \dots, 9\}$ defining the robot position

- Actions: $A(s) = \{\mathrm{up}, \mathrm{down}, \mathrm{left}, \mathrm{rig}ht\}$

  (except for those states near the black squares)

- Reward Function: $R(s) = \begin{cases} -1, & \forall s \neq G \\ 100, & \text{if } s = G \end{cases}$

- Transition function: $s' = \begin{cases} (x+1, y) & \text{if } a = \mathrm{right} \\ (x-1, y) & \text{if } a = \mathrm{left} \\ (x, y+1) & \text{if } a = \mathrm{up} \\ (x, y-1) & \text{if } a = \mathrm{down} \end{cases}$

Maze Task – Value function

	86	87	88	89	90	91	92		100	G
	85	86		90	91	92	93		99	
S	86	87		91	92	93	94		98	
	87	88		92	93	94	95	96	97	
	88	89	90	91	92		94	95	96	
	87	88	89	90	91	92	93	94	95	

V^*

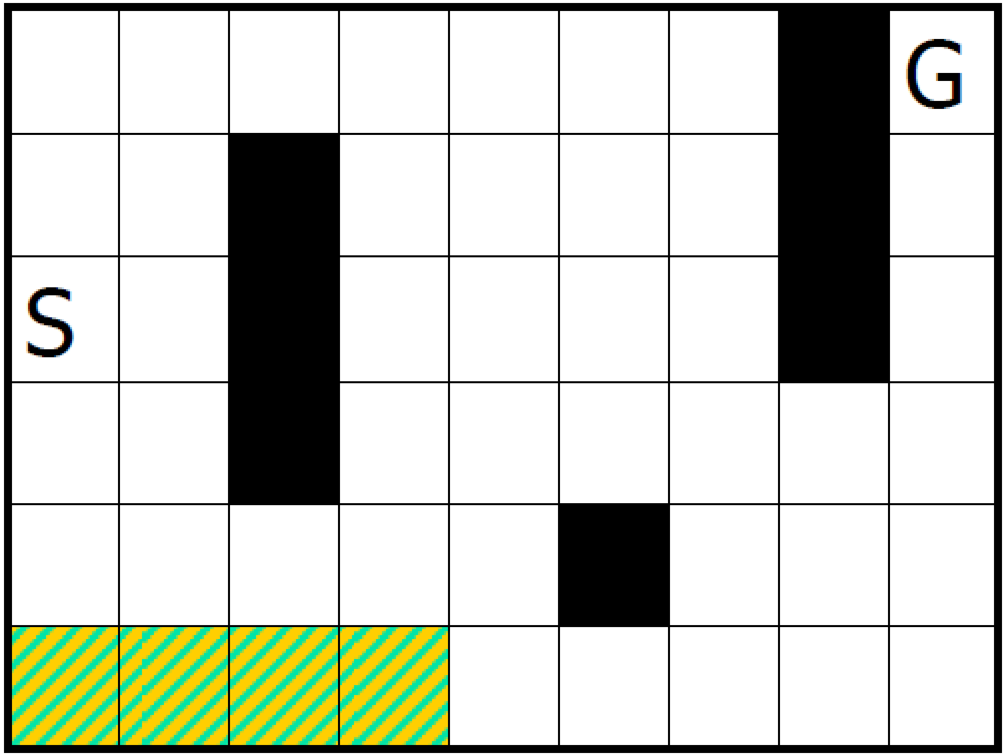
What's an optimal path from S to G?

Maze Task – Optimal Path

	86	87	88	89	90	91	92		100	G
	85	86		90	91	92	93		99	
S	86	87		91	92	93	94		98	
	87	88		92	93	94	95	96	97	
	88	89	90	91	92		94	95	96	
	87	88	89	90	91	92	93	94	95	

V^*

Another Maze Task



Now what's an optimal path from S to G?

With:
P=0.1 to the right
P=0.1 to the left

Everything else same as before, except:

With some nonzero probability, a small wind gust might displace the agent one cell to the right or left of its intended direction of travel on any step

Entering any of the 4 patterned cells at the southwest corner yields a reward of -100

Another Maze Task – MDP model

- Reward function: same as before, except that

$$R(s) = -100, \qquad \forall s \in S: x \leq 4, y = 1$$

- Transition function:

$$s = (x, y), a = \text{up}$$
$$P(s' = (x, y + 1) \mid s, a) = 0.8$$
$$P(s' = (x + 1, y) \mid s, a) = 0.1$$
$$P(s' = (x - 1, y) \mid s, a) = 0.1$$

Another Maze Task

	86.04	87.14	88.14	89.05	89.96	90.86	91.69		100	G
	85.15	86.13		89.93	90.87	91.87	92.78		99.00	
S	84.25	85.03		90.83	91.85	92.87	93.88		98.00	
	83.33	84.95		91.44	92.61	93.70	94.89	95.99	97.00	
	82.39	82.89	81.8	90.66	91.61		93.98	94.98	95.90	
	81.44	81.73	81.78	90.21	91.17	92.17	93.08	93.97	94.81	

V^*

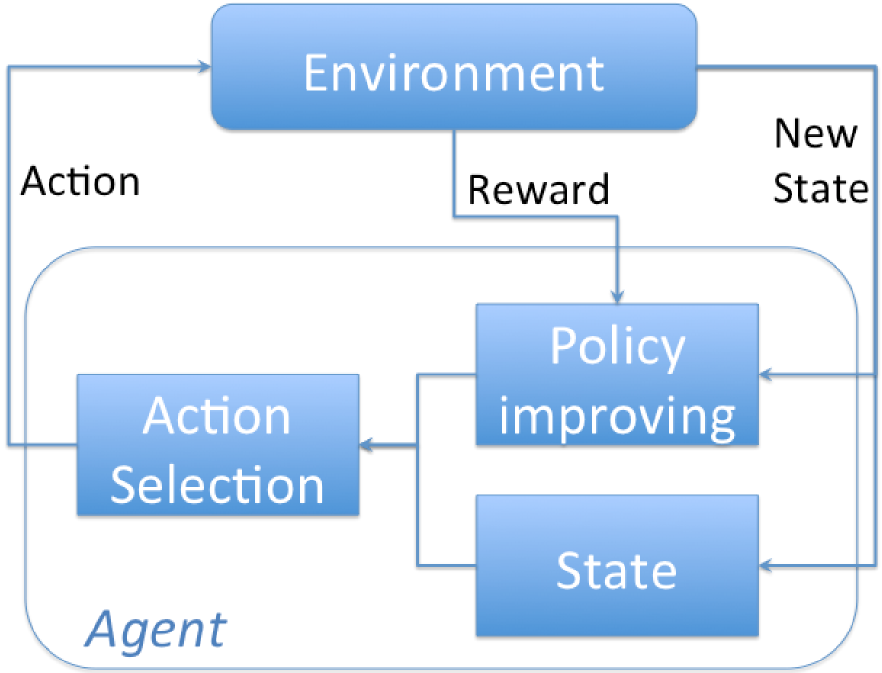
With probability 0.2, a small wind gust might displace the agent one cell to the right or left of its intended direction of travel on any step

Entering any of the 4 patterned cells at the southwest corner yields a reward of -100

Why is On-line Learning Important?

- So far, we assumed all system parameters to be known
- Finding the optimal policy becomes a straightforward computational problem
  - E.g., value iteration, but even policy iteration, linear programming, ecc…
- What if rewards/transitions probabilities are unknown? Can we compute the optimal policy?
- We have to deal with a *Reinforcement Learning problem*!

Agent-Environment Interaction



- Everything *inside* the agent is completely known and

controllable by the agent

- Everything *outside* is incompletely controllable but may or may not be completely known

Agent Knowledge

- A reinforcement learning problem can be posed in a variety of different ways depending on assumptions about the level of knowledge initially available to the agent

- In problems of *complete knowledge*, the agent has a complete and accurate model of the environment's dynamics

- If the environment is an MDP, then such a model consists of the one-step *transition probabilities* and *expected rewards* for all states and their allowable actions

- In problems of *incomplete knowledge*, a complete and perfect model of the environment is not available

Q-Values

- For any policy π, define $Q^{\pi}: S \times A \rightarrow R$ by

$$Q^{\pi}(s, a) = \sum_{t=0}^{\infty} \gamma^t r(t)$$

Again, the correct expression for a general MDP should use expected values here

  - $s(0) = s$ is the initial state,
  - $a(0) = a$ is the action taken,
  - and all subsequent states, actions, and rewards arise following policy π

- Just like V^{π} except that action a is taken at the very first step and only after this, policy π is followed
- Bellman equations can be rewritten in terms of Q-values

What are Q-values?

If agent is in this state	And starts with this action and then follows the policy	Return should be
s_1	a_1	-5
s_1	a_2	3
s_2	a_1	17.1
s_2	a_2	10
. . .	. . .	. . .

Q-Values

- Relationship between Value function and Q-function (given the state transition $s \rightarrow s'$)

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(s')$$

versus

$$Q^{\pi}(s, a) = R(s, a) + \gamma V^{\pi}(s')$$

Q-Values

- Relationship between Value function and Q-function

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(s')$$

versus

$$Q^{\pi}(s, a) = R(s, a) + \gamma V^{\pi}(s')$$

Q-Values

- Define $Q^* = Q^{\pi^*}$, where π* is an optimal policy

$$Q^*(s,a) = R(s,a) + \gamma V^*(s')$$

- Since:

$$V^*(s) = \max_a \{R(s,a) + \gamma V^*(s')\}$$

- Then:

$$V^*(s) = \max_a Q^*(s,a)$$

- And:

$$Q^*(s,a) = R(s,a) + \gamma \max_{a'} Q^*(s',a')$$

Q-Values

- The optimal policy π^* is greedy for Q^*, that is

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$$

[it follows from $V^*(s) = \max_a Q^*(s,a)$]

Q-learning Algorithm

- Q is the estimated utility function
  - It tells us how good an action is, given a certain state
  - It includes immediate reward for making an action + best utility (Q) for the resulting state (future utility)
  - It allows to compute the optimal policy
- Q-learning is based on an online estimation of the Q function

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,[\,r(s,a) + \gamma \max_{a'} Q(s',a')]$$

Q-learning Algorithm

Initialize $Q(s,a)$ arbitrarily

Repeat (for each decision epoch)

Initialize s

Repeat (for each step of episode)

Choose a from s using a policy derived from Q

Take action a, observe $r(s,a)$,

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,[\,r(s,a) + \gamma \max_{a'} Q(s',a')]$$

$s \leftarrow s'$

until s is terminal

Exploitation and exploration

- Q-learning algorithm does not specify what the agent should actually do. The agent learns a Q-function that can be used to determine an optimal action. There are two things that are useful for the agent to do:
  - **exploit** the knowledge that it has found for the current state *s* by doing one of the actions *a* that maximizes Q*[s,a]*.
  - **explore** in order to build a better estimate of the optimal Q-function. That is, it should select a different action from the one that it currently thinks is best.

Exploitation and exploration

Choose *a* from *s* using a policy derived from *Q*

- Simple Approach: ε-greedy policy
  - ε small number, e.g., 0.1

Generate a random number p

if $p \leq \varepsilon$

  Choose an action at random $\rightarrow$ explore

else

  Choose the greedy action $a^* = \arg\max_a Q(s,a) \rightarrow$ exploit

end

Q-learning discussion

- Q-learning is guaranteed to converge to the optimal Q-values if all Q(s,a) values are updated infinitely often (Watkins and Dayan 1992)
- It follows that exploration is necessary
  - A common approach is the ε-greedy strategy
- Q-learning can be very slow to converge to the optimal policy, especially if the state space is large
- One of the biggest challenges in the RL field is to *speed up* the learning process

Learning or planning?

- Classical DP emphasis for optimal control
  - Dynamics and reward structure known
  - Off-line computation
- Traditional RL emphasis
  - Dynamics and/or reward structure initially unknown
  - On-line learning
- Computation of an optimal policy off-line with known dynamics and reward structure can be regarded as planning

More information

- You can find more information about MDPs and learning here:

  https://webdocs.cs.ualberta.ca/~sutton/book/the-book.html