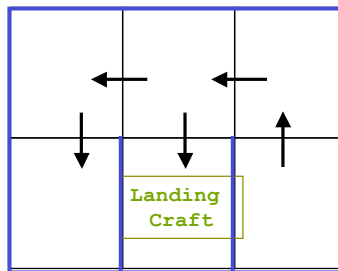


Markov Decision Processes (and a small amount of reinforcement learning)

Nicholas Roy
16.410/13
Session 23

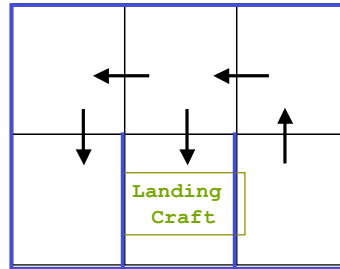
Slides adapted from:
Brian Williams, MIT
Manuela Veloso,
Andrew Moore,
Reid Simmons, &
Tom Mitchell, CMU

How Should a Rover Search for its Landing Craft?



- State Space Search?
- As a Constraint Satisfaction Problem?
- Goal-directed Planning?
- Linear Programming?
- Is the real world well-behaved?

How Should a Rover Search for its Landing Craft?

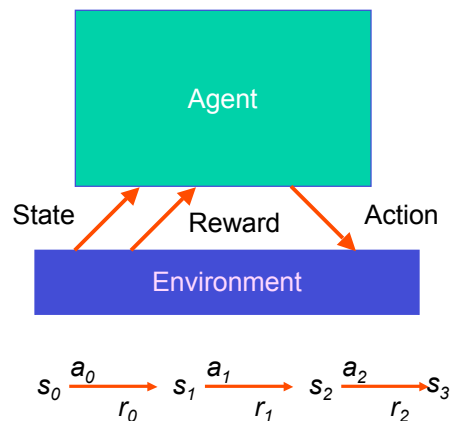


- What if each action can have one of a set of different outcomes?
- What if the outcomes occur probabilistically?

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 Problem: Model



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

Markov Decision Processes (MDPs)

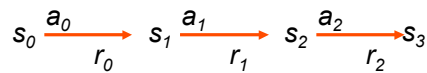
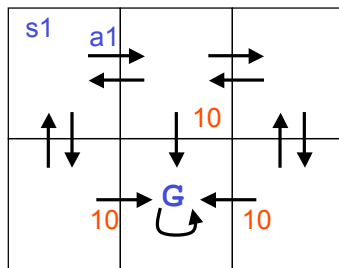
Model:

- Finite set of states, S
- Finite set of actions, A
- (Probabilistic) state transitions, $T(s_i, a_j, s_k)$
- Reward for each state and action, $R(s_i, a_j)$

Process:

- Observe state s_t in S
- Choose action a_t in A
- Receive immediate reward r_t
- State changes to some s_{t+1} according to $T(s_t, a_t, s_{t+1})$

Example:



- Legal transitions shown
- Rewards on unlabeled transitions are 0.

MDP Environment Assumptions

• Markov Assumption:

Next state and reward is a function only of the current state and action:

- $p(s_{t+1} | a_t, s_t, a_{t-1}, s_{t-1}, a_{t-2}, \dots) = p(s_{t+1} | a_t, s_t)$
- $r(s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, \dots) = r(s_t, a_t)$

• Uncertain and Unknown Environment:

$p(s_{t+1} | a_t, s_t)$ and r may be nondeterministic and unknown

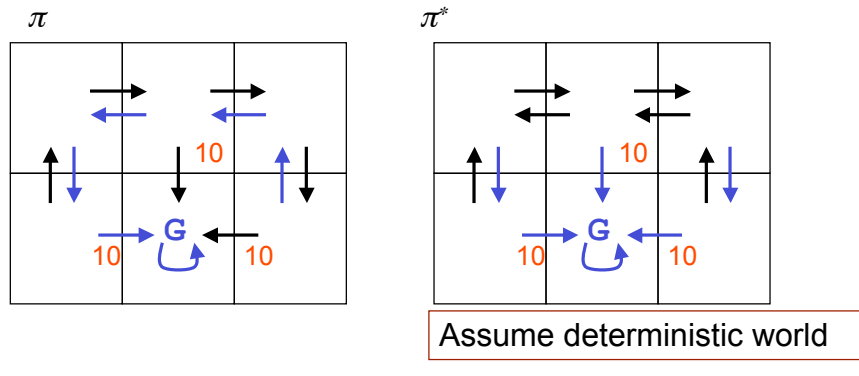
So what is the 'solution' to an MDP?

An MDP solution is a policy $\pi: S \rightarrow A$

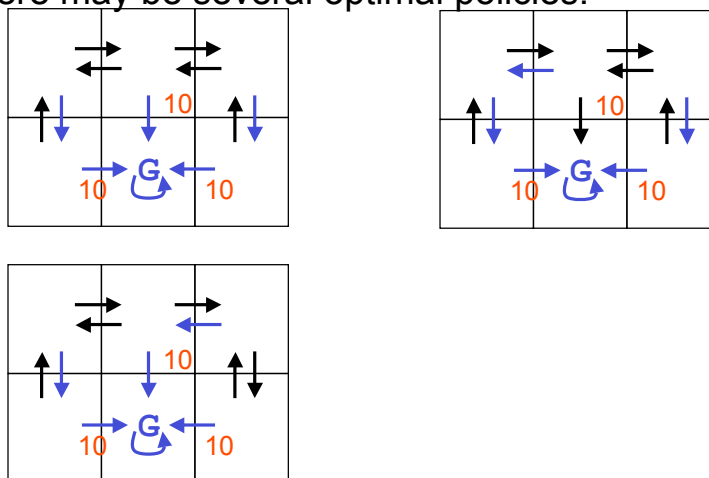
- Selects an action for each state.

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

- Selects action for each state that maximizes lifetime reward.



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



What is this “lifetime reward”?

- Optimal policy maximizes expected reward of agent over the lifetime of the agent.

$$\pi^*(s) = \arg \max_{a \in A} E_{s_0, s_1, \dots} \left[\int_{t=0}^{\infty} r(s_t, a_t) dt \right]$$

- How long will the agent live?
- 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

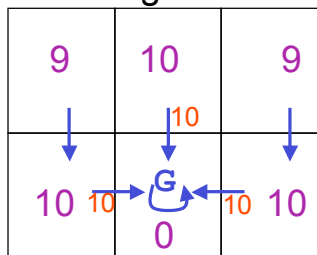
Value Function V^π for a Given Policy π

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

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

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

where $r_t, r_{t+1}, r_{t+2} \dots$ are generated by following π , starting at s_t .
Assume $\gamma = .9$



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

Notice: Suppose, given state s , we knew the lifetime rewards for all other states?

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

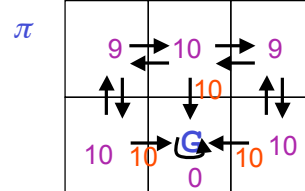
Lifetime reward $Q(s, a_i)$ is:

- the immediate reward for taking action $r(s, a) \dots$
- probability of posterior state s' : $p(s'|s, a)$
- life time reward starting in target state $V(s')$...
- discounted by γ .

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

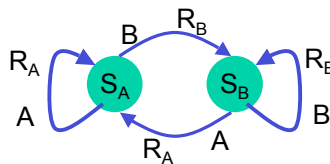
Must Know:

- Value function
- Environment model.
 - $p : S \times A \times S \rightarrow \mathfrak{R}$
 - $r : S \times A \rightarrow \mathfrak{R}$



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:

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

- Optimal value function for an n step horizon:

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

➤ Optimal value function for an infinite horizon:

$$V^*(s) = \max_{a_i} [r(s, a_i) + \gamma \sum_{s'} V^*(s') p(s'|s, a_i)]$$

Solving MDPs by Value Iteration

Insight: Calculate optimal values iteratively using DP

Algorithm:

1. Label all states: for each state s
 - $V_0(s) \leftarrow \max_a r(s, a)$
 2. Iteratively calculate value using Bellman's Equation: for each state s
 - $V_{t+1}(s) \leftarrow \max_a [r(s, a) + \gamma \sum_{s'} V_t(s') p(s'|s, a)]$
 3. Terminate when values are "close enough"

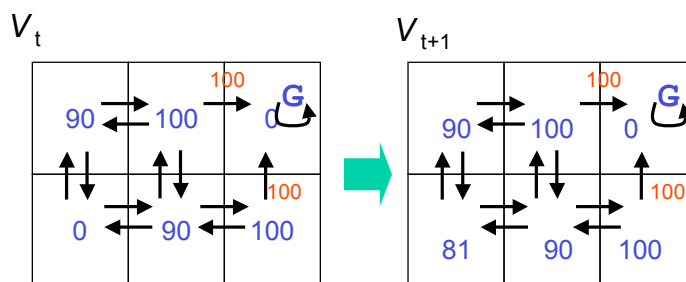
$$|V_{t+1}(s) - V_t(s)| < \epsilon$$
 4. Return $V^* = V_{t+1}$
- Policy Execution: agent selects optimal action by one step lookahead on V :

$$\pi(s) = \operatorname{argmax}_a [r(s, a) + \gamma \sum_{s'} V_{t(s)}(s') p(s'|s, a)]$$

Example of Value Iteration

$$V_{t+1}(s) \leftarrow \max_a [r(s, a) + \gamma \sum_{s'} V_t(s') p(s'|s, a)]$$

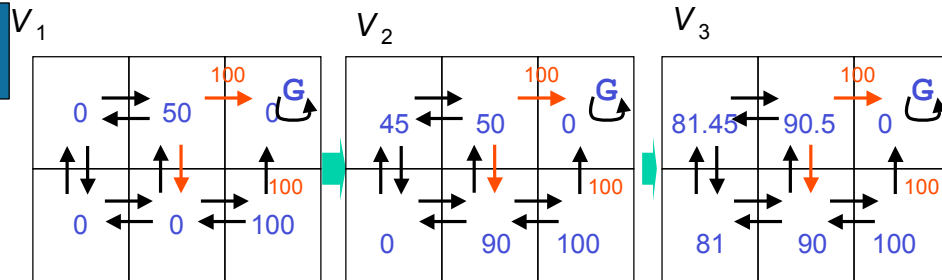
$\gamma = 0.9$, $p(s'|s, a)$ is deterministic



Example of Value Iteration

$$V_{t+1}(s) \leftarrow \max_a [r(s,a) + \gamma \sum_{s'} V_t(s') p(s'|s,a)]$$

$\gamma = 0.9$, $p(s'|s, a)$ is non-deterministic (red arcs occur with 50% probability)



Convergence of Value Iteration

- If terminate when values are “close enough”

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

Then:

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

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

Ideas in this lecture

- Objective is to accumulate rewards, rather than goal states.
- Objectives are achieved along the way, rather than at the end.
- Policies can be described by value functions, which describe the greatest lifetime reward achievable at every state.
- Value iteration is a fast algorithm for computing the value function under certain assumptions.

Appendix: 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):
$$V_{t+1}^*(s) \leftarrow r(s, \pi_t(s)) + \gamma \sum_{s'} V_t^*(s') p(s'|s, \pi_t(s))$$
 - Solve linear equations in $O(N^3)$
 - Solve iteratively, similar to value iteration.

Reinforcement Learning Problem

Given: Repeatedly...

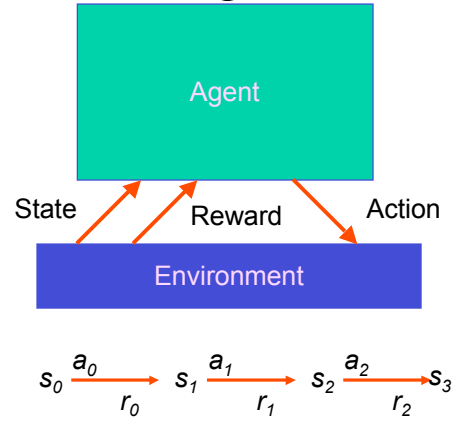
- Executed action
- Observed state
- Observed reward

Learn action policy $\pi: S \rightarrow A$

- Maximizes life reward
 $r_0 + \gamma r_1 + \gamma^2 r_2 \dots$
from any start state.
- Discount: $0 < \gamma < 1$

Note:

- Unsupervised learning
- Delayed reward
- Model not known



Goal: Learn to choose actions that maximize life reward

$$r_0 + \gamma r_1 + \gamma^2 r_2 \dots$$

How About Learning the Model?

- Certainty Equivalence
 1. Explore the world
 2. Count how often reward r occurs when in state s :
 $\text{SumR}[s_i] \leftarrow r, \text{Count}[s_i] \leftarrow 1.$
 3. Count how often state s' occurs when in state s
 $\text{Trans}[S_i, S_j] \leftarrow 1.$
 4. At any time:
 - $r_{\text{est}}(S_j) = \text{SumR}[S_j] / \text{Count}[S_j]$
 - $T_{ij}^{\text{est}} = \text{Trans}[S_i, S_j] / \text{Count}[S_i]$
 5. So at any time we can solve for V^{est}

Certainty Equivalence Costs

- Memory: $O(N^2)$
- Time to update counters: $O(1)$
- Time to re-evaluate V
 - $O(N^3)$ if use matrix inversion
 - $O(N^2 k_{\text{CRIT}})$ if use value iteration and we need k_{CRIT} iterations to converge
 - $O(N k_{\text{CRIT}})$ if use value iteration, and k_{CRIT} to converge, and T is Sparse (i.e. mean # successors is constant)
- Too expensive for some people.
- Prioritized sweeping will help, (see later), but first let's review a very inexpensive approach

Eliminating the Model with Q Functions

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

Key idea:

- Define function like V^* that encapsulates δ and r :

$$Q(s,a) = r(s,a) + \gamma \sum_{s'} V^*(s') p(s'|s, a)$$

- Then, if agent learns Q , it can choose an optimal action without knowing δ or r .

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

How Do We Learn Q?

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s'} V^*(s') p(s'|s, a_t)$$

- Need to eliminate V^* In update rule.
- Note Q and V^* are closely related:

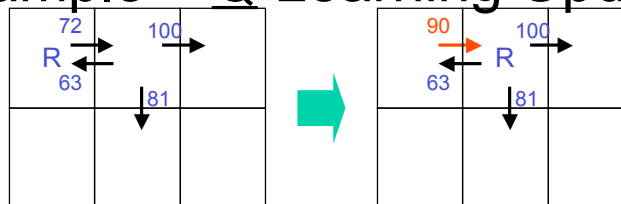
$$V^*(s) = \max_{a'} Q(s, a')$$

- Substituting Q for V^* :

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'} Q(s', a')$$

Example – Q Learning Update

$\gamma = 0.9$



$$\begin{aligned} \underline{Q}(s_1, a_{right}) &\leftarrow r(s_1, a_{right}) + \gamma \max_{a'} \underline{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max \{63, 81, 100\} \\ &\leftarrow 90 \end{aligned}$$

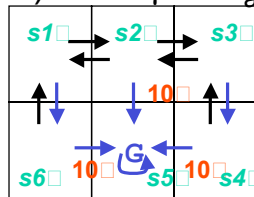
Note: if rewards are non-negative:

- For all $s, a, n, \underline{Q}_n(s, a) \leq \underline{Q}_{n+1}(s, a)$
- For all $s, a, n, 0 \leq \underline{Q}_n(s, a) \leq Q(s, a)$

Q-Learning Iterations

- Starts at top left corner – move clockwise around perimeter; Initially all values in Q table are zero; $\gamma = 0.8$

$$\underline{Q}(s, a) \leftarrow r + \gamma \max_{a'} \underline{Q}(s', a')$$



$Q(s1,E)$	$Q(s2,E)$	$Q(s3,S)$	$Q(s4,W)$
0	0	0	$r + \gamma \max_{a'} \{Q(s5,loop)\} = 10 + 0.8 \times 0 = 10$
0	0	$r + \gamma \max_{a'} \{Q(s4,W), Q(s4,N)\} = 0 + 0.8 \times \max\{10,0\} = 8$	10
0	$r + \gamma \max_{a'} \{Q(s3,W), Q(s3,S)\} = 0 + 0.8 \times \max\{0,8\} = 6.4$	8	10

Crib Sheet: Q-Learning for Deterministic Worlds

Let \underline{Q} denote the current approximation to Q .

Initially:

- For each s, a initialize table entry $\underline{Q}(s, a) \leftarrow 0$
- Observe current state s

Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\underline{Q}(s, a)$ as follows:

$$\underline{Q}(s, a) \leftarrow r + \gamma \max_{a'} \underline{Q}(s', a')$$

- $s \leftarrow s'$

Discussion

- How should the learning agent use the *intermediate Q values*?
- Exploration vs Exploitation
- Scaling up in the size of the state space
 - Function approximators (neural net instead of table)
 - Reuse, use of macros

NonDeterministic Case

- We redefine V , Q by taking expected values

$$V^\pi(s_t) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots]$$

$$V^\pi(s_t) = E[\sum \gamma^i r_{t+i}]$$

$$Q(s_t, a_t) = E[r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))]$$

Nondeterministic Case

- Alter training rule to

$$\underline{Q}_n(s, a) \leftarrow (1 - \alpha_n) \underline{Q}_{n-1}(s, a) + \alpha_n [r + \gamma \max_{a'} \underline{Q}_{n-1}(s', a')]$$

where $\alpha_n = 1/(1 + \text{visits}_n(s, a))$ and $s' = \delta(s, a)$.

Can still prove convergence of \underline{Q} [Watkins and Dayan, 92]

Ongoing Research

- Handling case where state is only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $\underline{\delta} : S \times A \rightarrow S$
- Relationship to dynamic programming
- Multiple learners – Multi-agent reinforcement learning