Course Logistics

- CS533: Intelligent Agents and Decision Making
  - M, W, F: 1:00—1:50
  - Instructor: Alan Fern (KEC2071)
  - Office hours: by appointment (see me after class or send email)
  - *Emailing me: include "CS533 Student" in subject and repeat if you don't hear back within a day*
  - Course website (link on instructor's home page) has
    - Lecture notes, Assignments, and HW Solutions
- Written Homework:
  - Assigned and collected regularly (submission via email is accepted)
  - Not graded for correctness, but rather for "completion with good effort" (no guarantee on when it will be returned---make a copy if you want one)
  - Must *"complete with good effort"* 90% of homework or a letter grade will be deducted from final grade
- Grade based on:
  - 25% Midterm exam (in class)
  - 25% Final exam (take home)
  - 25% Final Project (during last month)
  - 25% 3 mini-projects (require some implementation)

Some AI Planning Problems

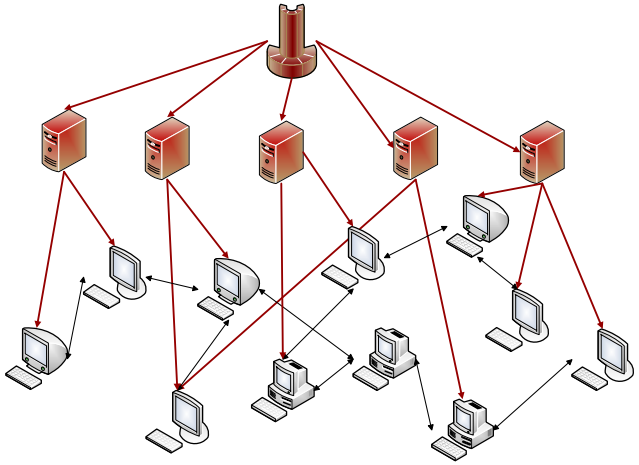
Fire & Rescue Response Planning

Solitaire

Real-Time Strategy Games

Helicopter Control

Legged Robot Control



Network Security/Control

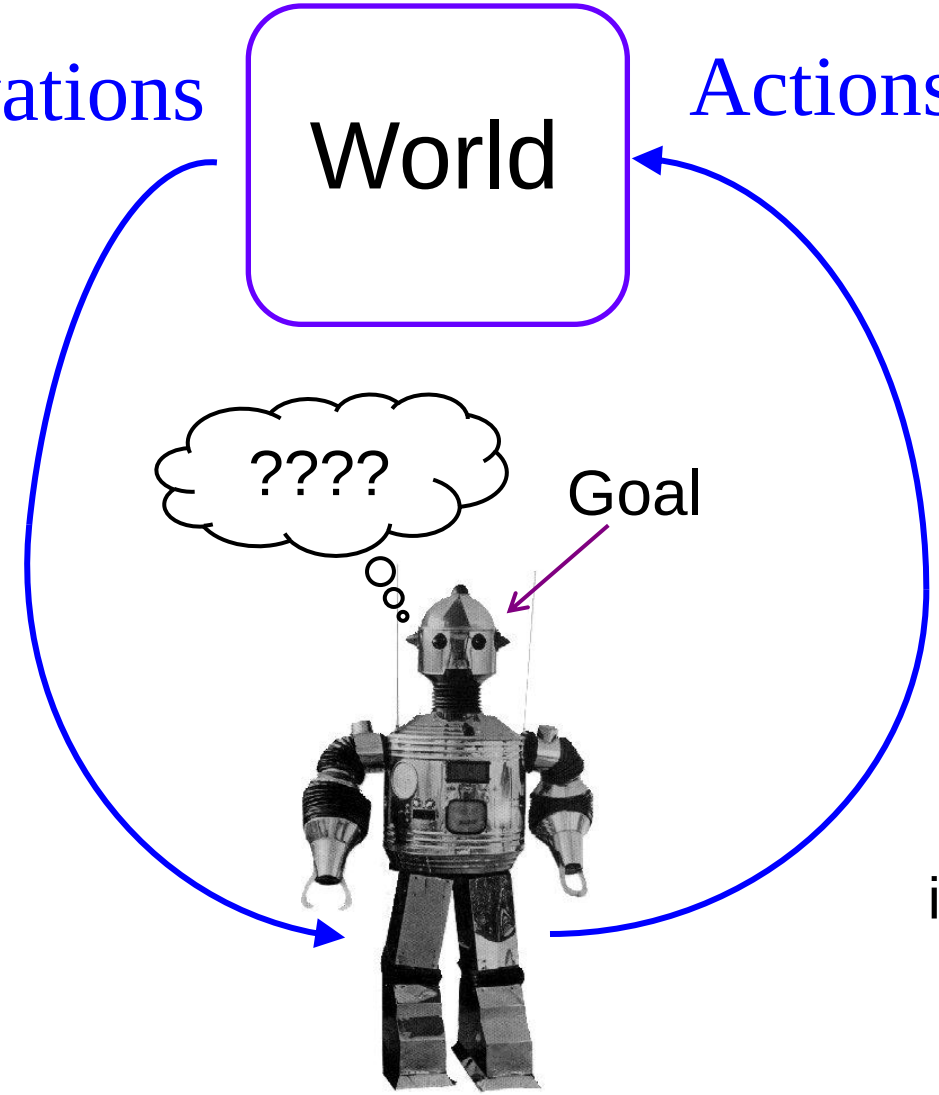
Some AI Planning Problems

- Health Care
  - Personalized treatment planning
  - Hospital Logistics/Scheduling
- Transportation
  - Autonomous Vehicles
  - Supply Chain Logistics
  - Air traffic control
- Assistive Technologies
  - Automated assistants for elderly/disabled
  - Household robots
- Sustainability
  - Smart grid
  - Forest fire management
- …..

Common Elements

- We have a controllable system that can change state over time (in some predictable way)
  - The state describes essential information about system (the visible card information in Solitaire)
- We have an objective that specifies which states, or state sequences, are more/less preferred
- Can (partially) control the system state transitions by taking certain actions
- **Problem:** At each moment must select an action to optimize the overall objective
  - Produce most preferred state sequences

Some Dimensions of AI Planning



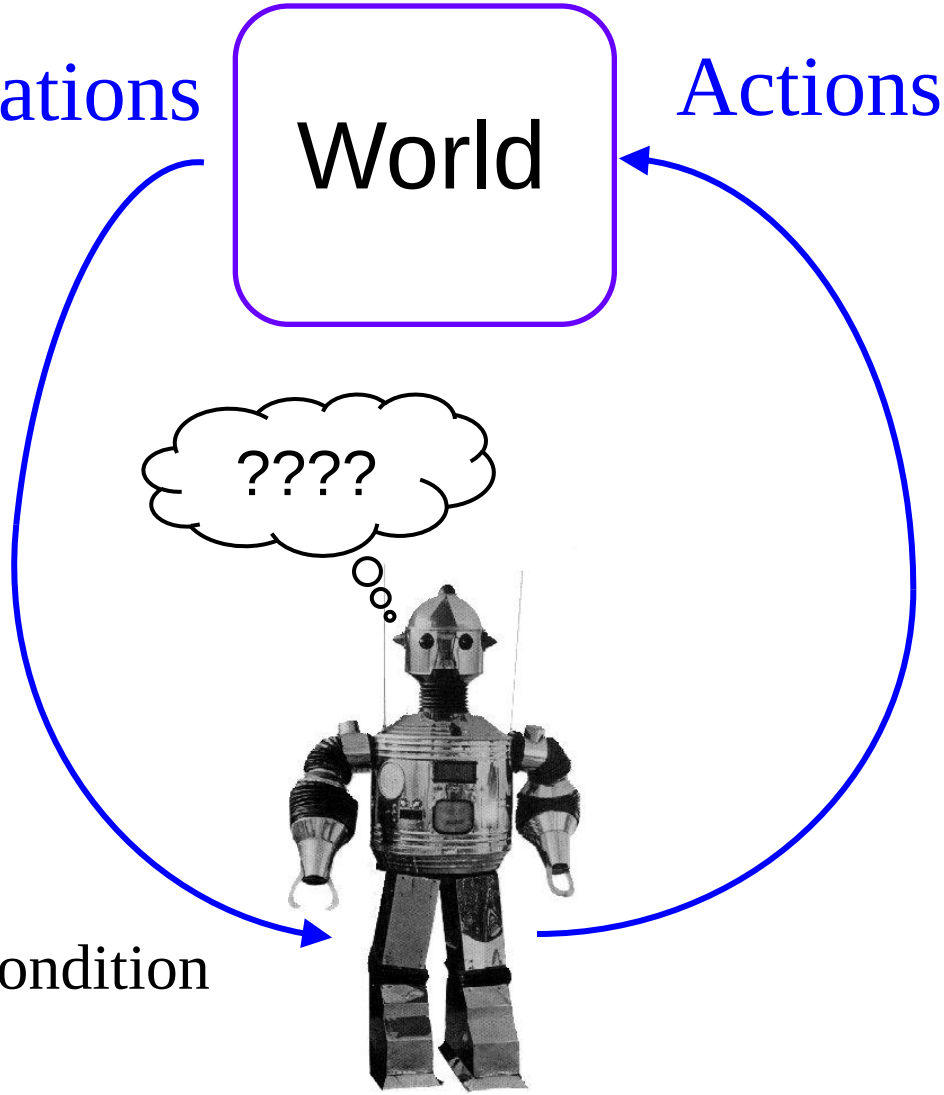
fully observable vs. partially observable

sole source of change vs. other sources

deterministic vs. stochastic

instantaneous vs. durative

Classical Planning Assumptions
(primary focus of AI planning until early 90's)



sole source of change

fully observable

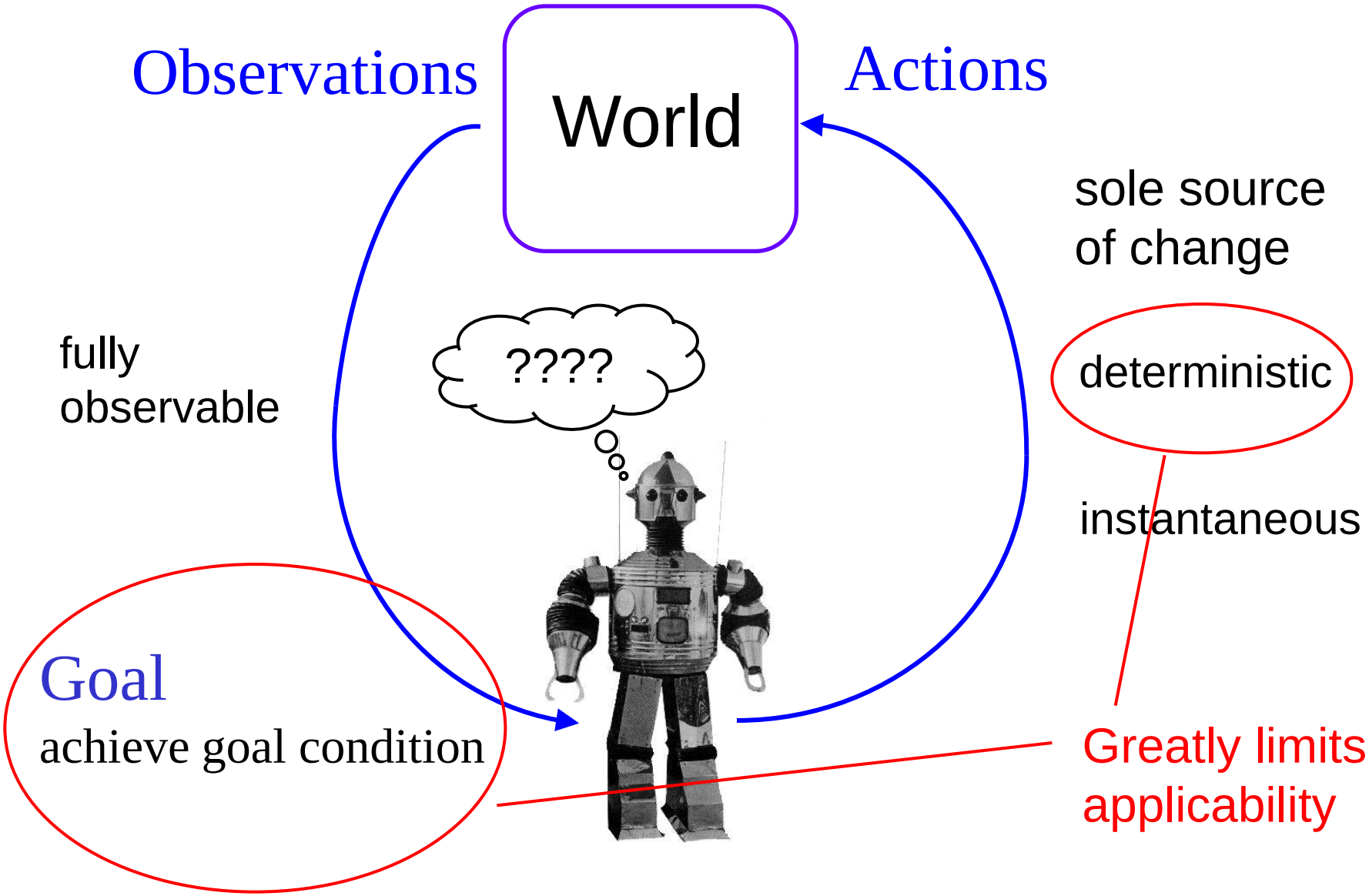
deterministic

instantaneous

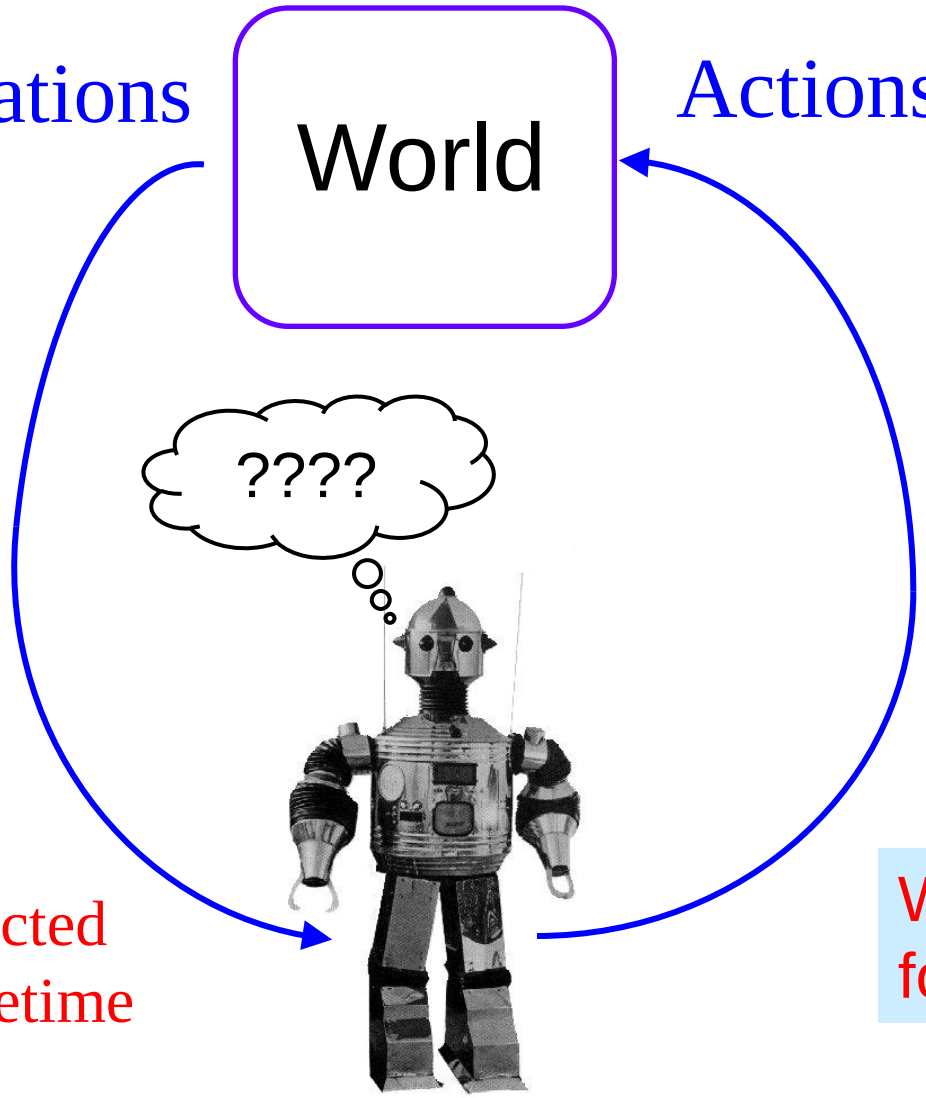
Goal

achieve goal condition

Classical Planning Assumptions
(primary focus of AI planning until early 90's)



Stochastic/Probabilistic Planning: Markov Decision Process (MDP) Model



fully observable

sole source of change

stochastic

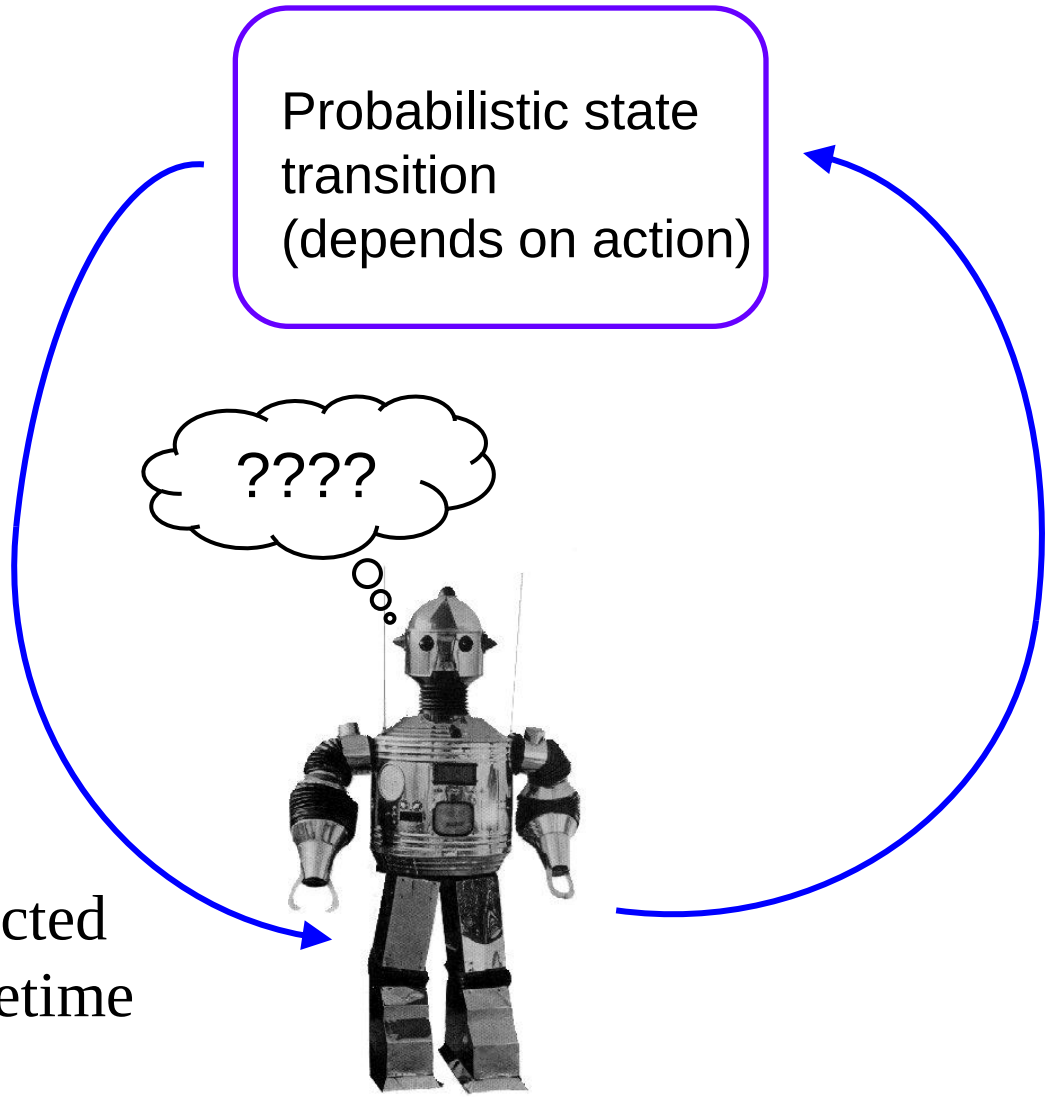
instantaneous

Goal

maximize expected reward over lifetime

We will primarily focus on MDPs

Stochastic/Probabilistic Planning: Markov Decision Process (MDP) Model

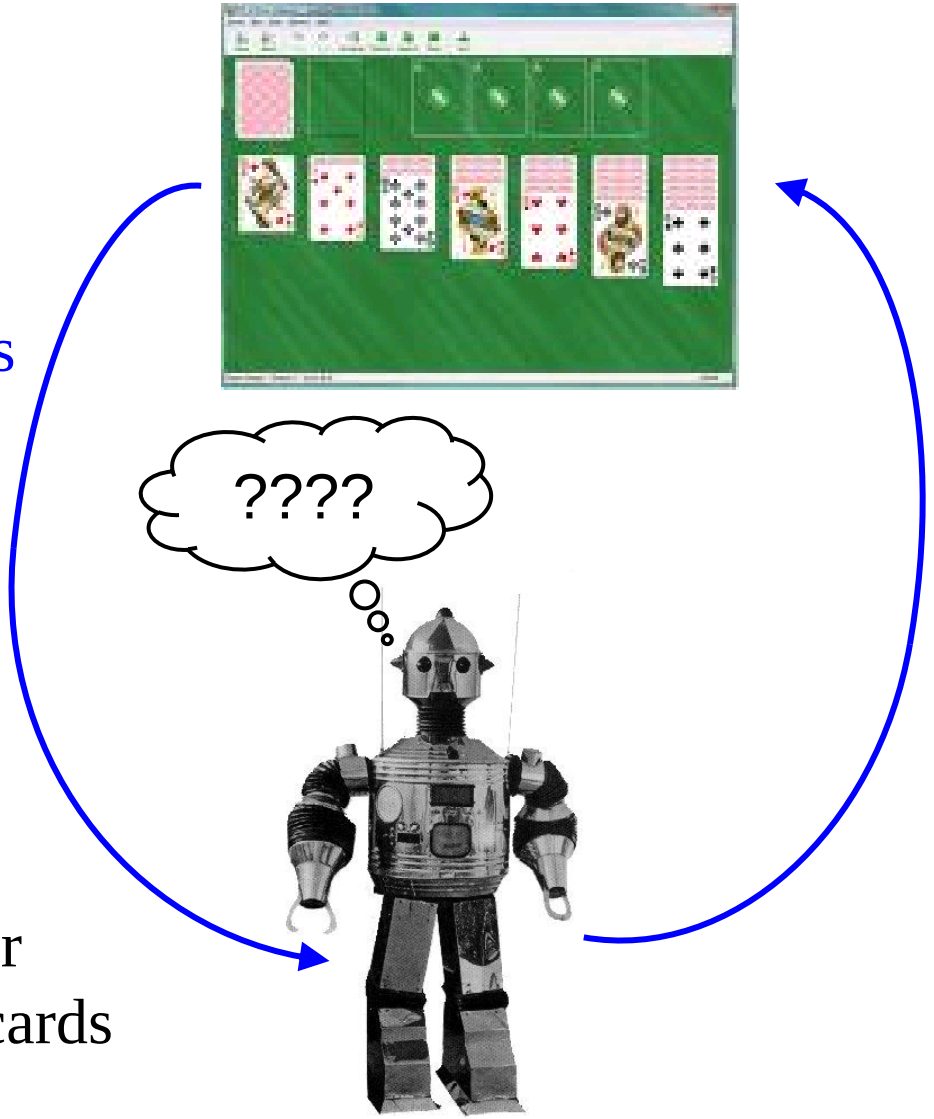


Goal

maximize expected reward over lifetime

Example MDP

State describes all visible info about cards



Action are the different legal card movements

Goal

win the game or
play max # of cards

Course Outline

- Structured around algorithms for solving MDPs
  - Different assumptions about knowledge of MDP model
  - Different assumptions about how MDP is represented

1. Markov Decision Processes (MDPs) Basics
   - Basic definitions and solution techniques
   - Assume an exact MDP model is known
   - **Exact solutions** for **small/moderate** size problems
2. Monte-Carlo Planning
   - Assumes an MDP simulator is available
   - **Approximate solutions** for **large** problems
3. Reinforcement learning
   - MDP model is not known to agent
   - **Exact solutions** for **small/moderate** problems
   - **Approximate solutions** for **large** problems
4. Planning w/ Factored Representations of Huge MDPs
   - Symbolic Dynamic Programming
   - Classical planning for deterministic problems

Markov Decision Processes

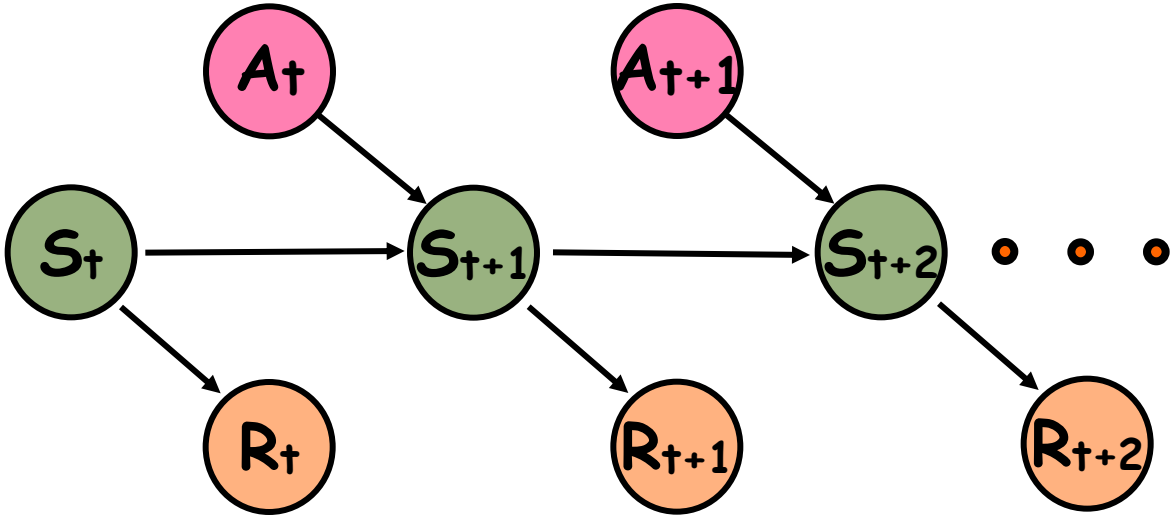
Alan Fern *

* Based in part on slides by Craig Boutilier and Daniel Weld

Markov Decision Processes

- An MDP has four components: S, A, R, T:
  - finite state set S (|S| = n)
  - finite action set A (|A| = m)
  - transition function T(s,a,s') = Pr(s' | s,a)
    - Probability of going to state **s'** after taking action **a** in state **s**
    - How many parameters does it take to represent?
      - m*n*(n-1)
  - bounded, real-valued reward function R(s)
    - Immediate reward we get for being in state s
    - Roughly speaking the objective is to select actions in order to maximize total reward
    - For example in a goal-based domain R(s) may equal 1 for goal states and 0 for all others (or -1 reward for non-goal states)

Graphical View of MDP



Assumptions

- **First-Order Markovian dynamics** (history independence)
  - $Pr(S^{t+1}|A^t,S^t,A^{t-1},S^{t-1},..., S^0) = Pr(S^{t+1}|A^t,S^t)$
  - Next state only depends on current state and current action
- **State-Dependent Reward**
  - $R^t = R(S^t)$
  - Reward is a deterministic function of current state
- **Stationary dynamics**
  - $Pr(S^{t+1}|A^t,S^t) = Pr(S^{k+1}|A^k,S^k)$ for all t, k
  - The world dynamics and reward function do not depend on absolute time
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, after the action is executed, we see what the state is

What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: ????

- Should the solution to an MDP be just a sequence of actions such as (a1,a2,a3, ….) ?
  - Consider a single player card game like Blackjack/Solitaire.
- No! In general an action sequence is not sufficient
  - Actions have stochastic effects, so the state we end up in is uncertain
  - This means that we might end up in states where the remainder of the action sequence doesn't apply or is a bad choice
  - A solution should tell us what the best action is for any possible situation/state that might arise

Policies ("plans" for MDPs)

- A solution to an MDP is a policy
  - Two types of policies: nonstationary and stationary
- Nonstationary policies are used when we are given a finite planning horizon H
  - I.e. we are told how many actions we will be allowed to take
- Nonstationary policies are functions from states and times to actions
  - $\pi: S \times T \rightarrow A$, where T is the non-negative integers
  - $\pi(s,t)$ tells us what action to take at state s when there are t stages-to-go (note that we are using the convention that t represents stages/decisions to go, rather than the time step)

Policies ("plans" for MDPs)

- What if we want to continue taking actions indefinately?
  - Use stationary policies

- A Stationary policy is a mapping from states to actions
  - $\pi: S \rightarrow A$
  - $\pi(s)$ is action to do at state s (regardless of time)
  - specifies a continuously reactive controller

- Note that both nonstationary and stationary policies assume or have these properties:
  - full observability
  - history-independence
  - deterministic action choice

What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: a policy such that ????

- We don't want to output just any policy
- We want to output a "good" policy
- One that accumulates lots of reward

Value of a Policy

- How good is a policy π?
  - How do we measure reward "accumulated" by π?
- **Value function** V: S →$\mathbb{R}$ associates value with each state (or each state and time for non-stationary π)
- $V_\pi(s)$ denotes value of policy at state s
  - Depends on immediate reward, but also what you achieve subsequently by following π
  - An **optimal policy** is one that is no worse than any other policy at any state
- The goal of MDP planning is to compute an optimal policy

What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: a policy that achieves an "optimal value"

- This depends on how we define the value of a policy
- There are several choices and the solution algorithms depend on the choice
- We will consider two common choices
  - Finite-Horizon Value
  - Infinite Horizon Discounted Value

Finite-Horizon Value Functions

- We first consider maximizing expected total reward over a finite horizon
- Assumes the agent has H time steps to live
- To act optimally, should the agent use a stationary or non-stationary policy?
  - I.e. Should the action it takes depend on absolute time?
- Put another way:
  - If you had only one week to live would you act the same way as if you had fifty years to live?

Finite Horizon Problems

- Value (utility) depends on stage-to-go
  - hence use a nonstationary policy

- $V_\pi^k(s)$ is k-stage-to-go value function for π
  - expected total reward for executing π starting in s for k time steps

$$V_\pi^k(s) = E\left[\sum_{t=0}^{k} R^t \mid \pi, s\right]$$

$$= E\left[\sum_{t=0}^{k} R(s^t) \mid a^t = \pi(s^t, k-t), s^0 = s\right]$$

- Here R^t and s^t are random variables denoting the reward received and state at time-step t when starting in s
  - These are random variables since the world is stochastic

Computational Problems

- There are two problems that we will be interested in solving

- **Policy evaluation**:
  - Given an MDP and a nonstationary policy π
  - Compute finite-horizon value function $V_\pi^k(s)$ for any k

- **Policy optimization**:
  - Given an MDP and a horizon H
  - Compute the optimal finite-horizon policy
  - We will see this is equivalent to computing optimal value function

- How many finite horizon policies are there?
  - $|A|^{Hn}$
  - So can't just enumerate policies for efficient optimization

Finite-Horizon Policy Evaluation

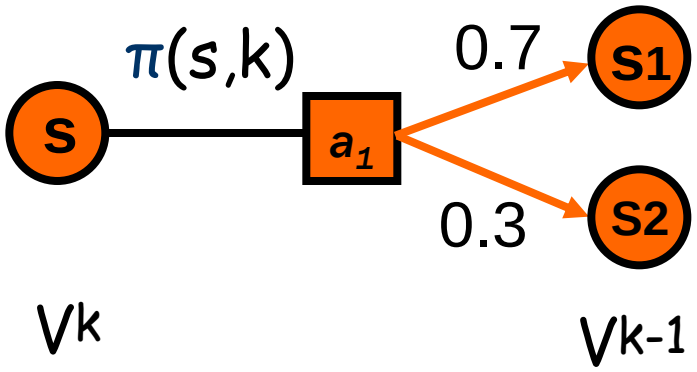
- Can use dynamic programming to compute $V_\pi^k(s)$
  - Markov property is critical for this

(a) $V_\pi^0(s) = R(s), \quad \forall s$

(b) $V_\pi^k(s) = R(s) + \sum_{s'} T(s, \pi(s,k), s') \cdot V_\pi^{k-1}(s')$

immediate reward

expected future payoff with *k*-1 stages to go



What is total time complexity?

$O(Hn^2)$

Finite-Horizon Policy Evaluation

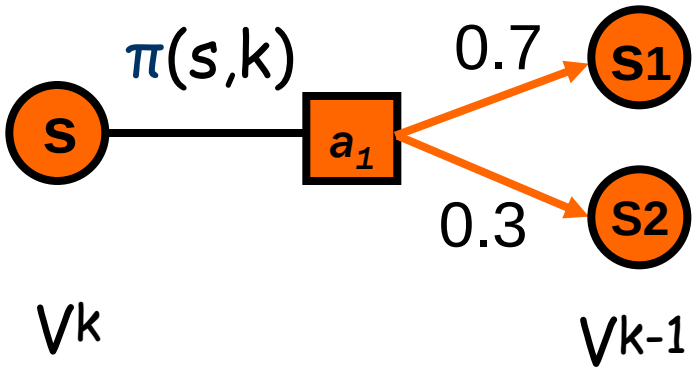
- Can use dynamic programming to compute $V_\pi^k(s)$
  - Markov property is critical for this

(a) $V_\pi^0(s) = R(s), \quad \forall s$

(b) $V_\pi^k(s) = R(s) + \sum_{s'} T(s, \pi(s,k), s') \cdot V_\pi^{k-1}(s')$

immediate reward

expected future payoff with *k*-1 stages to go

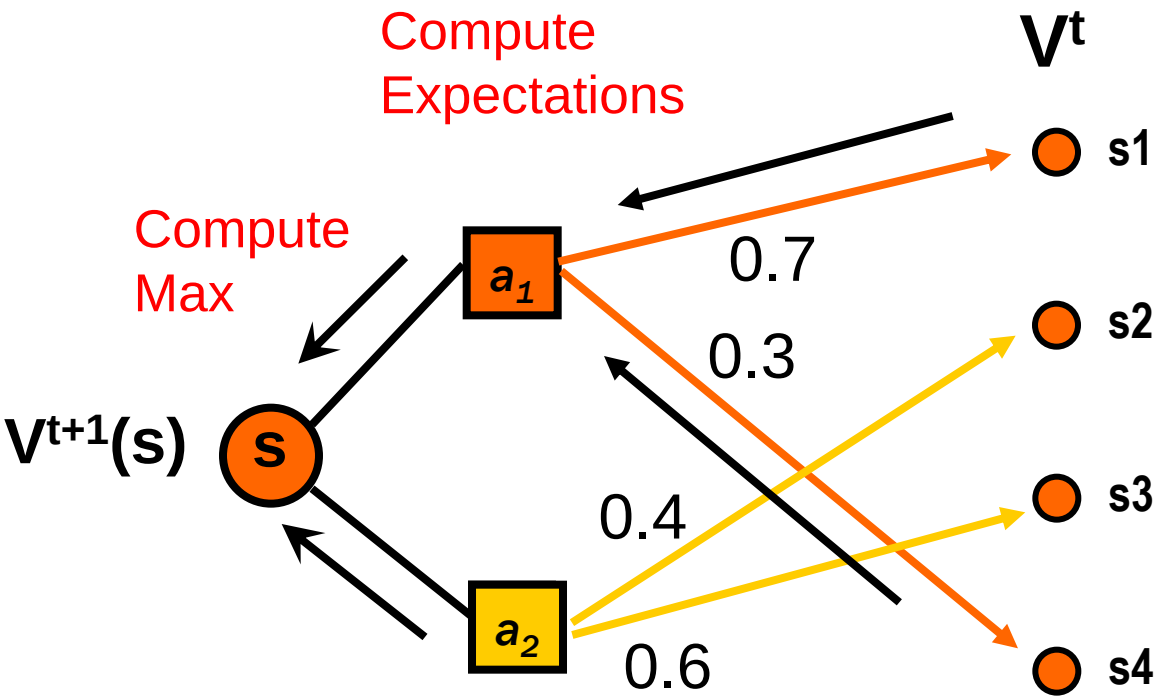


What is total time complexity?

$O(kn^2)$

Policy Optimization: Bellman Backups

How can we compute the optimal $V^{t+1}(s)$ given optimal V^t ?



$$V^{t+1}(s) = R(s) + \max \{ \; 0.7\, V^t(s1) + 0.3\, V^t(s4) \;\; , \;\; 0.4\, V^t(s2) + 0.6\, V^t(s3) \; \}$$

Value Iteration: Finite Horizon Case

- Markov property allows exploitation of DP principle for optimal policy construction
  - no need to enumerate $|A|^{Hn}$ possible policies
- Value Iteration

Bellman backup

$$V^0(s) = R(s), \quad \forall s$$

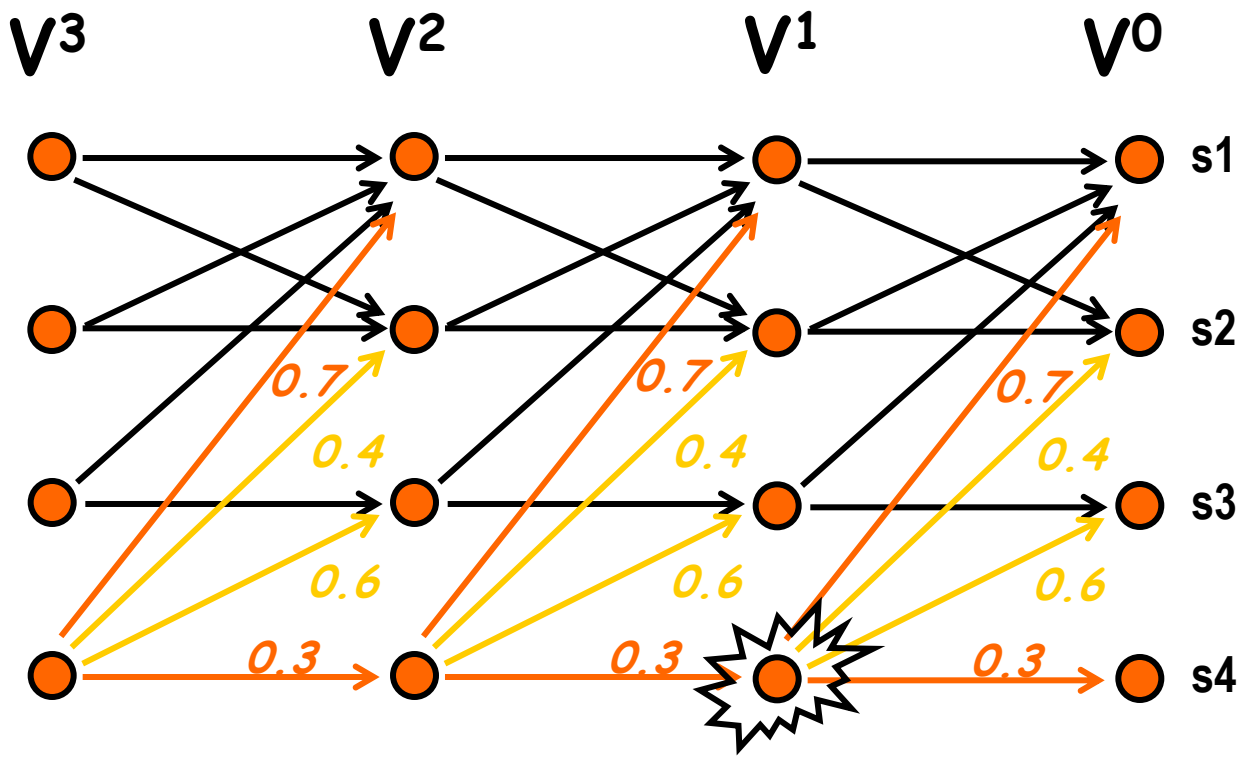
$$V^k(s) = R(s) + \max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

$$\pi^*(s,k) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

V^k is optimal k-stage-to-go value function

$\Pi^*(s,k)$ is optimal k-stage-to-go policy

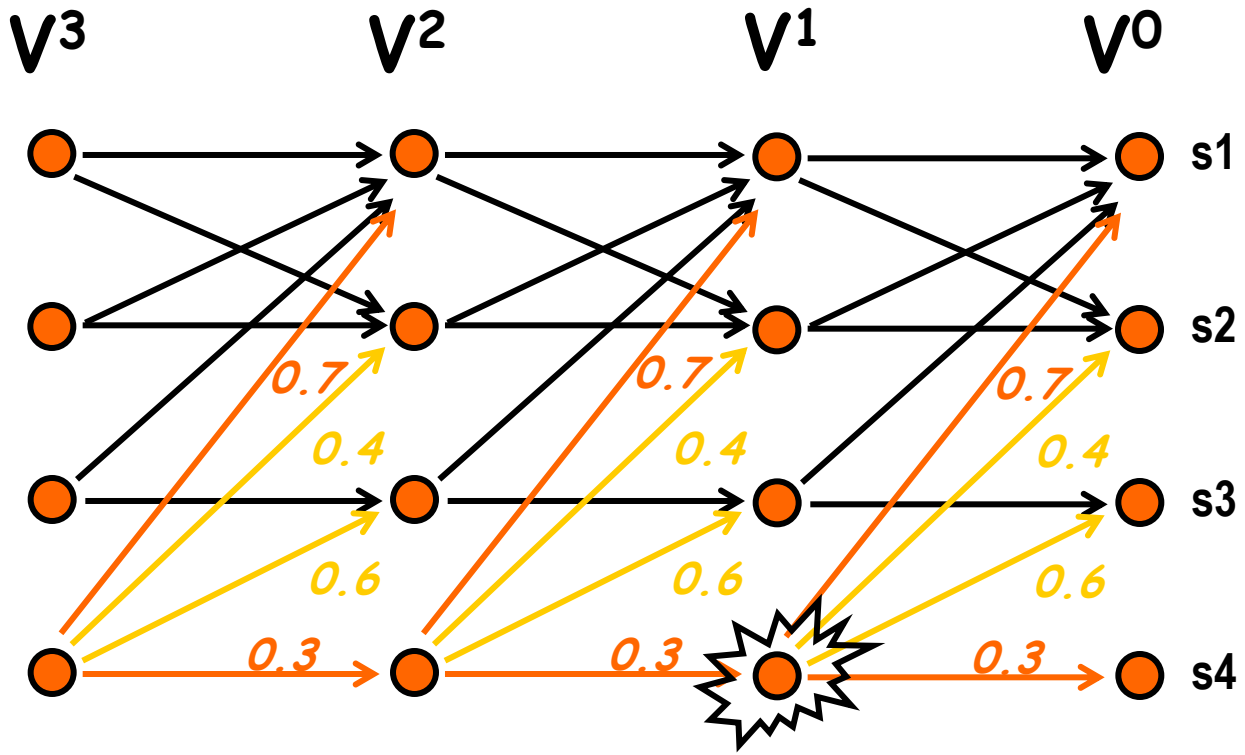
Value Iteration



$V^1(s4) = R(s4) + \max \{ 0.7\ V^0(s1) + 0.3\ V^0(s4)$

$0.4\ V^0(s2) + 0.6\ V^0(s3) \}$

Value Iteration



$\Pi^*(s4,t) = \max \{ \square \; \square \}$

Value Iteration: Complexity

- Note how DP is used
  - optimal soln to k-1 stage problem can be used without modification as part of optimal soln to k-stage problem
- What is the computational complexity?
  - H iterations
  - At each iteration, each of n states, computes expectation for |A| actions
  - Each expectation takes O(n) time
- Total time complexity: $O(H|A|n^2)$
  - Polynomial in number of states. Is this good?

Summary: Finite Horizon

- Resulting policy is optimal

$$V_{\pi^*}^{k}(s) \geq V_{\pi}^{k}(s), \quad \forall \pi, s, k$$

  - convince yourself of this (use induction on k)

- Note: optimal value function is unique, but optimal policy is not (can be ties between actions)
  - Many policies can have same value

Discounted Infinite Horizon MDPs

- Defining value as total reward is problematic with infinite horizons
  - many or all policies have infinite expected reward
  - some MDPs are ok (e.g., zero-cost absorbing states)
- "Trick": introduce discount factor $0 \leq \beta < 1$
  - future rewards discounted by β per time step

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \beta^t R^t \mid \pi, s\right]$$

- Note: $V_\pi(s) \leq E\left[\sum_{t=0}^{\infty} \beta^t R^{\max}\right] = \frac{1}{1-\beta} R^{\max}$

Bounded Value

- Motivation: economic? prob of death? convenience?

Notes: Discounted Infinite Horizon

- Optimal policies guaranteed to exist (Howard, 1960)
  - I.e. there is a policy that maximizes value at each state
- Furthermore there is always an optimal stationary policy
  - Intuition: why would we change action at s at a new time when there is always forever ahead
  - We define $V^*(s) = V_\pi(s)$ for some optimal stationary π

Policy Evaluation

- Value equation for fixed policy
  - Immediate reward + Expected discounted future reward

$$V_\pi(s) = R(s) + \beta \sum_{s'} T(s, \pi(s), s') \cdot V_\pi(s')$$

derive this from original definition

- How can we compute the value function for a policy?
  - we are given R and T
  - linear system with n variables and n constraints
    - Variables are values of states: V(s1),…,V(sn)
    - Constraints: one value equation (above) per state
  - Use linear algebra to solve for V (e.g. matrix inverse)

Policy Evaluation via Matrix Inverse

$\boldsymbol{V}_\pi$ and $\boldsymbol{R}$ are n-dimensional column vector (one element for each state)

$\boldsymbol{T}$ is an nxn matrix s.t. $\mathrm{T}(\mathrm{i}, \mathrm{j}) = \mathrm{T}(\mathrm{s}_\mathrm{i}, \pi(s_i), s_j)$

$$V_\pi = R + \beta \mathrm{T} V_\pi$$

$$\Downarrow$$

$$(I - \beta \mathrm{T}) V_\pi = R$$

$$\Downarrow$$

$$V_\pi = (I - \beta \mathrm{T})^{-1} R$$

Computing an Optimal Value Function

- **Bellman equation** for optimal value function

$$V^*(s) = R(s) + \beta \max_a \sum_{s'} T(s,a,s') \cdot V^*(s')$$

  - Bellman proved this is always true for an optimal value function

- How can we compute the optimal value function?
  - The MAX operator makes the system non-linear, so the problem is more difficult than policy evaluation

- Notice that the optimal value function is a **fixed-point** of the Bellman Backup operator B (i.e. B[V*]=V*)
  - B takes a value function as input and returns a new value function

$$B[V](s) = R(s) + \beta \max_a \sum_{s'} T(s,a,s') \cdot V(s')$$

Value Iteration

- Can compute optimal policy using value iteration based on Bellman backups, just like finite-horizon problems (but include discount term)

$$V^0(s) = 0$$

$$V^k(s) = R(s) + \beta \max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

- Will it converge to optimal value function as k gets large?
  - Yes. $\lim_{k \to \infty} V^k = V^*$
- When should we stop in practice?

Convergence: Contraction Property

- B[V] is a **contraction operator** on value functions

- That is, operator B satisfies:
  For any V and V', **|| B[V] – B[V'] || ≤ β || V – V' ||**

- Here ||V|| is the max-norm, which returns the maximum element of the vector.
  - E.g. ||(0.1 100 5 6)|| = 100

- So applying a Bellman backup to any two value functions causes them to get closer together in the max-norm sense!

Convergence

- Using the contraction property we can prove convergence of value iteration.
- Proof:
  1. For any V: $\| V^* - B[V] \| = \| B[V^*] - B[V] \| \leq \beta \| V^* - V \|$
  2. So applying Bellman backup to any value function V brings us closer to V* by a constant factor β in max-norm sense
  3. This means that $\|V^k - V^*\| \leq \beta^k \| V^* - V^0 \|$
  4. Thus $\lim_{k \to \infty} \left\| V^* - V^k \right\| = 0$

Stopping Condition

- Want to stop when we can guarantee the value function is near optimal.
- Key property:

If $\|V^k - V^{k-1}\| \leq \varepsilon$ then $\|V^k - V^*\| \leq \varepsilon\beta/(1-\beta)$

You'll show this in your homework

- Continue iteration until $\|V^k - V^{k-1}\| \leq \varepsilon$
  - Select small enough ε for desired error guarantee

How to Act

- Given a V^k from value iteration that closely approximates V*, what should we use as our policy?
- Use *greedy* policy: (one step lookahead)

$$greedy[V^k](s) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^k(s')$$

- Note that the value of greedy policy may not be equal to V^k
- Let V^G be the value of the greedy policy? How close is V^G to V*?
  - I.e. how close is our greedy policy to optimal after k iterations

How to Act

- Given a V^k that closely approximates V*, what should we use as our policy?
- Use *greedy* policy: (one step lookahead)

$$greedy[V^k](s) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^k(s')$$

- This selects the action that looks best if we assume that we get value V^k in one step

- How good is this policy?

Value of Greedy Policy

$$greedy[V^k](s) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^k(s')$$

- Define V_g to be the value of this greedy policy
  - This is likely not the same as V^k (convince yourself of this)
- **Property:** If $||V^k - V^*|| \leq \lambda$ then $||V_g - V^*|| \leq 2\lambda\beta/(1-\beta)$
  - Thus, V_g is not too far from optimal if V^k is close to optimal
- Set stopping condition so that V_g has desired accuracy
- Furthermore, there is a finite ε s.t. greedy policy is optimal
  - That is, even if value estimate is off, greedy policy is optimal once it is close enough. Why?

Optimization via Policy Iteration

- Recall, given policy, can compute its value exactly:

$$V_\pi(s) = R(s) + \beta \sum_{s'} T(s, \pi(s), s') \cdot V_\pi(s')$$

- Policy iteration exploits this: iterates steps of policy evaluation and policy improvement

1. Choose a random policy π
2. Loop:
   (a) Evaluate V_π
   (b) For each s in S, set $\pi'(s) = \arg\max_a \sum_{s'} T(s, a, s') \cdot V_\pi(s')$
   (c) Replace π with π'

Until no improving action possible at any state

Policy improvement

Policy Iteration: Convergence

- Policy improvement guarantees that π' is no worse than π. Further if π is not optimal then π' is strictly better in at least one state.
  - Local improvements lead to global improvement!
  - For proof sketch see http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node42.html
  - I'll walk you through a proof in a HW problem
- Convergence assured
  - No local maxima in value space (i.e. an optimal policy exists)
  - Since finite number of policies and each step improves value, then must converge to optimal
- Gives exact value of optimal policy

Policy Iteration Complexity

- Each iteration runs in polynomial time in the number of states and actions
- There are at most $|A|^n$ policies and PI never repeats a policy
  - So at most an exponential number of iteations
  - Not a very good complexity bound
- Empirically O(n) iterations are required
  - **Challenge:** try to generate an MDP that requires more than that n iterations
- Still no polynomial bound on the number of PI iterations (open problem)!
  - But maybe not anymore …..

Value Iteration vs. Policy Iteration

- Which is faster? VI or PI
  - It depends on the problem
- VI takes more iterations than PI, but PI requires more time on each iteration
  - PI must perform policy evaluation on each iteration which involves solving a linear system
- VI is easier to implement since it does not require the policy evaluation step
- We will see that both algorithms will serve as inspiration for more advanced algorithms

Recap: things you should know

- What is an MDP?
- What is a policy?
  - Stationary and non-stationary
- What is a value function?
  - Finite-horizon and infinite horizon
- How to evaluate policies?
  - Finite-horizon and infinite horizon
  - Time/space complexity?
- How to optimize policies?
  - Finite-horizon and infinite horizon
  - Time/space complexity?
  - Why they are correct?