

# Controlling a Robot in Block World: Markov Decision Processes

Hal Daumé III

CS5350: Machine Learning

17 April 2008

1. What is a Markov Decision Process?
2. Solving MDPs...
  - 2.1 Exact solutions
  - 2.2 Value iteration
  - 2.3 Policy iteration
3. Summary and discussion

## What is a Markov Decision Process?

Model of an **agent** interacting with the world:

- ▶ Agent begins in some **state**
- ▶ Takes a sequence of **actions**
- ▶ Each action yields a **reward** and new **state**
- ▶ Goal is to choose **actions** to maximize sum of **rewards**

## What is wrong with this notion of a goal?

If you get a job that pays \$100k/year, what is your sum of rewards?

$$\$100k + \$100k + \$100k + \$100k + \dots = \infty$$

## Discounted rewards

**Idea:** Money now is better than money later!

1. Maybe there's inflation...
2. Maybe we die...
3. Maybe we just like short-term rewards...

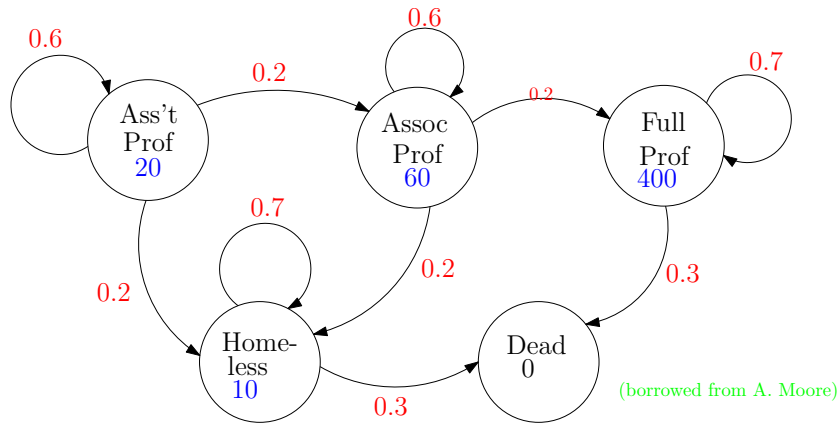
Standard assumption:

- ▶ Money  $T$  years from now is worth only  $0.9^T$  as much as now
- ▶ Or, more generally, we use  $\gamma$  discounting:

Total Reward = reward now

$$\begin{aligned}
 &+ \gamma \text{reward at time 1} \\
 &+ \gamma^2 \text{reward at time 2} \\
 &+ \gamma^3 \text{reward at time 3} \\
 &+ \dots
 \end{aligned}$$

## Discounted rewards for professors



For each state  $S$ , want to compute:

$J(S)$  = Expected reward starting in state  $S$

## Markov system (without actions)

Model the world as **states** and **transitions**:

1. States =  $S_1, S_2, \dots, S_N$
2. Rewards =  $r_1, r_2, \dots, r_N$
3. Transition matrix =  $\mathcal{P}$  with  $\mathcal{P}_{ij} = p(\text{next} = S_j \mid \text{this} = S_i)$
4. Discount factor  $\gamma$

Markov process:

1. Begin in state  $i$
2. Received reward  $r_i$
3. Randomly move to state  $j$  with probability  $\mathcal{P}_{ij}$
4. Go to (1) but discount remaining rewards by  $\gamma$

Goal: Compute  $J_i^*$  = Expected reward starting in state  $i$  for all  $i$

## Solving the system

Recall:

- ▶ Transition matrix =  $\mathcal{P}$  with  $\mathcal{P}_{ij} = p(\text{next} = S_j \mid \text{this} = S_i)$
- ▶ Goal: Compute  $J_i^*$  = Expected reward starting in state  $i$  for all  $i$

Can we compute  $J^*$ ?

$$\begin{aligned} J_i^* &= r_i + \gamma(\text{Expected reward starting in next state}) \\ &= r_i + \gamma\left(\sum_j \mathcal{P}_{ij} \text{Expected reward starting in state } j\right) \\ &= r_i + \gamma \sum_j \mathcal{P}_{ij} J_j^* \end{aligned}$$

In vector notation:

$$\mathbf{J}^* = \mathbf{R} + \gamma \mathcal{P} \mathbf{J}^* \implies \mathbf{J}^* = (\mathbf{I} - \gamma \mathcal{P})^{-1} \mathbf{R}$$

## An alternative solution: Value iteration

Idea: **Do everything on a step-by-step recursive basis**

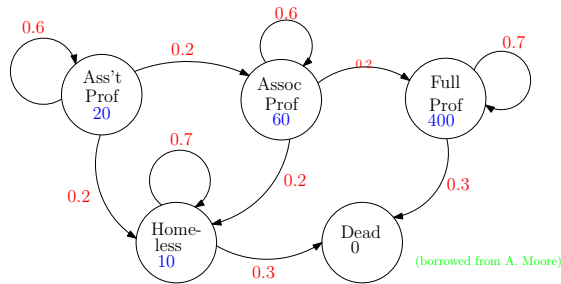
**Define:**

- $J_i^1$  = Expected reward starting in state  $i$  in 1 step
- $J_i^2$  = Expected reward starting in state  $i$  in 2 steps
- $J_i^t$  = Expected reward starting in state  $i$  in  $t$  steps

**Compute:**

$$\begin{aligned} J_i^1 &= r_i \\ J_i^2 &= r_i + \gamma \sum_j \mathcal{P}_{ij} J_j^1 \\ J_i^t &= r_i + \gamma \sum_j \mathcal{P}_{ij} J_j^{t-1} \end{aligned}$$

## Running value iteration



(borrowed from A. Moore)

t	$J^t(\text{asst})$	$J^t(\text{assc})$	$J^t(\text{full})$	$J^t(\text{hl})$	$J^t(\text{dead})$
1	20	60	400	10	0
2	$20 + .5(26)$ $= 33$	$60 + .5(118)$ $= 119$	$400 + .5(280)$ $= 540$	$10 + .5(7)$ $= 13.5$	$0 + .5(0)$ $= 0$
3	43.15	151.05	589	14.725	0
4	49.5225	165.565	606.15	15.1537	0
5	52.9286	171.7999	612.1525	15.3038	0

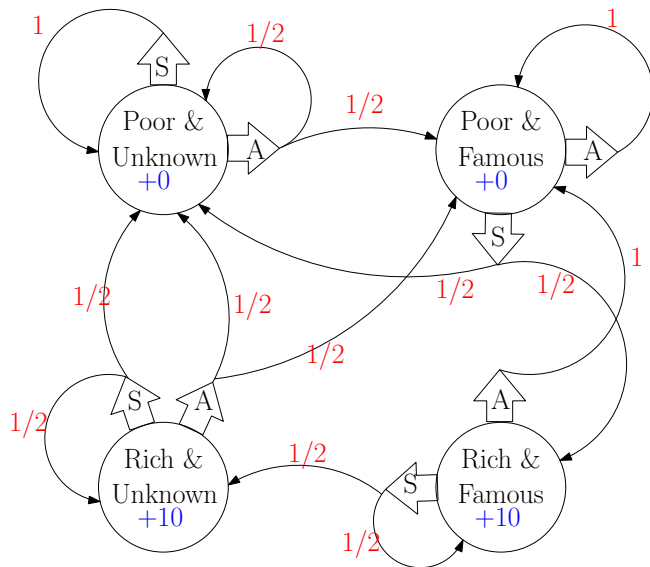
## Running value iteration

1. Compute  $J_i^1 = r_i$  for each state  $i$
2. Compute  $J_i^2 = r_i + \gamma \sum_j \mathcal{P}_{ij} J_j^1$  for each state  $j$
3. ...
4. Compute  $J_i^t = r_i + \gamma \sum_j \mathcal{P}_{ij} J_j^{t-1}$  for each state  $j$

**Important:** As  $t \rightarrow \infty$ ,  $J_i^t \rightarrow J_i^*$

- ▶ Stop when  $J$ s cease to change
- ▶ Much faster than inversion when  $\mathcal{P}$  is sparse

## A Markov Decision Process



## Markov system with actions

Model the world as **states** and **transitions**:

1. States =  $S_1, S_2, \dots, S_N$
2. Rewards =  $r_1, r_2, \dots, r_N$
3. A set of actions  $a_1, a_2, \dots, a_M$
4. Transition matrix =  $\mathcal{P}$  with  $\mathcal{P}_{ij}^k = p(\text{next} = S_j \mid \text{this} = S_i \text{ and I take action } k)$
5. Discount factor  $\gamma$

Markov process:

1. Begin in state  $i$
2. Received reward  $r_i$
3. Choose action  $k$
4. Randomly move to state  $j$  with probability  $\mathcal{P}_{ij}^k$
5. Go to (1) but discount remaining rewards by  $\gamma$

## Our goal: a Policy

A **policy** maps states to actions.

- ▶ Given a policy  $\pi$ , our MDP turns in to a Markov System
- ▶ We know how to solve Markov Systems
- ▶ How many policies are there?
- ▶ How do we find a good policy?

Important: for any MDP, there exists an optimal policy, usually denoted  $\pi^*$

What happens with the following two policies on the previous example? Policy 1 = “advertise only if famous”; Policy 2 = “always advertise”

## Value iteration for MDPs

**Define:**

$J_i^*$  = Expected future reward starting in state  $i$  and following  $\pi^*$

Following value iteration idea, define:

$J_i^t$  = Maximum possible expected reward starting in state  $i$  and living for  $t$  time steps

Note:  $J_i^1 = r_i$  for all  $i$

Why would  $J^*$  be useful for computing an optimal policy?

## Value iteration for MDPs

**Recall** that for non-decision processes, we had:

$$J_i^t = r_i + \gamma \sum_j \mathcal{P}_{ij} J_j^{t-1}$$

For processes with actions, we have:

$$J_i^t = r_i + \max_k \gamma \sum_j \mathcal{P}_{ij}^k J_j^{t-1}$$

- ▶ Above is known as “Bellman’s Equation”
- ▶  $J$  values can be computed exactly as before
- ▶ Also known as ... **Dynamic Programming**

## From Bellman to Optimal Policies

The optimal action in state  $i$  is given by:

$$\arg \max_k r_i + \gamma \sum_j \mathcal{P}_{ij}^k J_j^*$$

## Alternative Perspective: Policy Iteration

Work directly with policies rather than values.

Learn a sequence of policies  $\pi^{(1)}, \pi^{(2)}, \dots, \pi^{(\mathcal{L})}$ :

$\pi^{(\ell)}(i) =$  action chosen in state  $i$

1. Initialize  $\pi^{(1)}$
2. For  $\ell = 1 \dots \mathcal{L} - 1$ :
  - 2.1 Assume we follow  $\pi^{(\ell)}$  and compute  $J_i^{(\ell)*}$  using (non-MDP) value iteration
  - 2.2 Compute  $\pi^{(\ell+1)}(i) = \arg \max_k r_i + \gamma \sum_j \mathcal{P}_{ij}^k J_j^{(\ell)*}$
3. Return  $\pi^{(\mathcal{L})}$

Does this converge? Optimally?

## Value iteration versus policy iteration

Many actions?	Policy iteration
Few actions & acyclic?	Value iteration
Given okay policy?	Policy iteration
Few actions & cyclic?	Ehh...

What's the biggest problem with both of these methods?

When state-space is large, we can approximate the value functions using any regression algorithm.

But we're no longer (guaranteed) to be optimal

## Where are we?

Given a discrete state/action set, and known transition probabilities, can compute optimal policies (in two ways).

What happens if:

- ▶ the world isn't discrete?  
⇒ state splitting methods
- ▶ our actions aren't discrete?  
⇒ regress on actions
- ▶ we don't know the state space?  
⇒ simultaneous localization and mapping
- ▶ we can't even tell what state we're in?  
⇒ partially observable MDPs – hard
- ▶ the transition probabilities are unknown?  
⇒ reinforcement learning!