Markov Decision Processes (MDPs)

CS 486/686

Introduction to AI

University of Waterloo

Outline

- Sequential Decision Processes
  - Markov chains
    - Highlight Markov property
  - Discounted rewards
    - Value iteration
  - Markov Decision Processes

  - Reading: R&N 17.1-17.4

Markov chains

- Simplified version of snakes and ladders

11	10	9	8	7	6
0	1	2	3	4	5

- Start at state 0, roll dice, and move the number of positions indicated on the dice. If you land on square 4 you teleport to square 4
- Winner is the one who gets to 11 first

Example by D Precup

Markov chains

11	10	9	8	7	6
0	1	2	3	4	5

- Discrete clock pacing interaction of agent with the environment, t=0,1,2,...
- Agent can be in one of a set of states S={0,1,...,11}
- Initial state is $s_0=0$
- If an agent is in state s_t at time t, the state at time s_{t+1} is determined ***only by the role of the dice at time t***

Example by D Precup

Markov chains

11	10	9	8	7	6
0	1	2	3	4	5

- The probability of the next state s_{t+1} does not depend on how the agent got to the current state s_t (**Markov Property**)
- Example: Assume at time t, agent is at state 2
  - $P(s_{t+1}=3|s_t)=1/6$
  - $P(s_{t+1}=7|s_t)=1/3$
  - $P(s_{t+1}=5|s_t)=1/6$, $P(s_{t+1}=6|s_t)=1/6$, $P(s_{t+1}=8|s_t)=1/6$
  - Game is completely described by the ***probability distribution of the next state given the current state***

Example by D Precup

Markov Chain

- Formal representation
  - S = {0,1,2,3,4,5,6,7,8,9,10,11}

$$P = \begin{bmatrix}
0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & 0 \\
0 & 0 & 1/6 & 1/6 & 0 & 1/6 & 1/6 & 1/3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/6 & 0 & 1/6 & 1/6 & 1/3 & 1/6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 1/3 & 1/6 & 1/6 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 1/6 & 1/6 & 1/3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 1/6 & 1/2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/6 & 2/3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 5/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Transition probability matrix

$$P_{ij} = \text{Prob}(\text{Next}=s_j \mid \text{This}=s_i)$$

Making sequential decisions

- Markov chains
  - To highlight the Markov property
- Discounted rewards
  - Value iteration
- Markov decision processes

Discounted Rewards

- An assistant professor gets paid, say, 30K per year
- How much, in total, will the AP earn in their life?

$$30+30+30+30+\ldots=\infty$$

Discounted Rewards

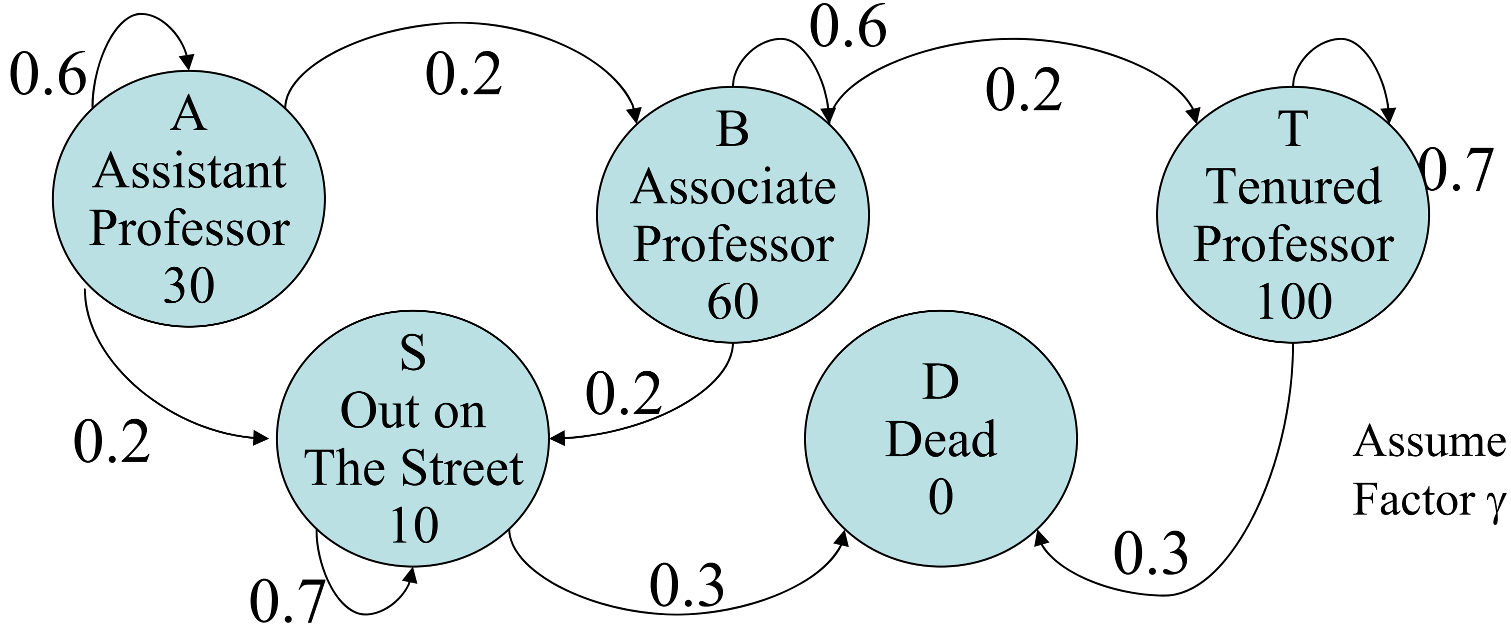
- A reward in the future is not worth quite as much as a reward now
  - Because of chance of obliteration
  - Because of chance of inflation
- **Example:**
  - Being promised $10000 next year is worth only 90% as much as receiving $10000 now
- Assuming payment n years in future is worth only $(0.9)^n$ of payment now, what is the AP's **Future Discounted Sum of Rewards?**

Discount Factors

- Used in economics and probabilistic decision-making all the time

- Discounted sum of future awards using discount factor γ is
  - Reward now + γ (reward in 1 time step) + γ^2(reward in 2 time steps) + γ^3(reward in 3 time steps)+...

The Academic Life



Assume Discount Factor $\gamma = 0.9$

- Define
  - U_A = Expected discounted future rewards starting in state A
  - U_B = Expected discounted future rewards starting in state B
  - U_T = Expected discounted future rewards starting in state T
  - U_S = Expected discounted future rewards starting in state S
  - U_D = Expected discounted future rewards starting in state D

  How do we compare U_A, U_B, U_T, U_S, U_D ?

A Markov System of Rewards...

- Has a set of states $\{s_1, s_2, ... s_n\}$
- Has a transition probability matrix

$$P = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{bmatrix} \quad P_{ij} = \mathrm{Prob}(\mathrm{Next} = s_j \mid \mathrm{This} = s_i)$$

- Each state has a reward $\{r_1, r_2, ... r_n\}$
- There's a discount factor γ, $0 < \gamma < 1$
- ON EACH STEP...
  0. Assume your state is s_i
  1. You get given reward r_i
  2. You randomly move to another state
     $P(\mathrm{NextState} = s_j \mid \mathrm{This} = s_i) = P_{ij}$
  3. All future rewards are discounted by γ

Solving a Markov Matrix

- Write $U^*(s_i)$ = expected discounted sum of future rewards starting in state s_i
- $U^*(s_i) = r_i + \gamma$ x (expected future rewards starting from your next state)

$$= r_i + \gamma\,(P_{i1}U^*(s_1) + P_{i2}\,U^*(s_2) + \ldots + P_{iN}\,U^*(s_N))$$

Using vector notation write:

$$\bar{\mathrm{U}} = \begin{pmatrix} U^*(S_1) \\ U^*(S_2) \\ \vdots \\ U^*(S_n) \end{pmatrix} \quad \bar{\mathrm{R}} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix} \quad \bar{\mathrm{P}} = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ P_{n1} & P_{2n} & \cdots & P_{nn} \end{pmatrix}$$

There is a closed form expression for U in terms of R, P and γ.

Solving a Markov System using Matrix Inversion

- Upside: You get an exact number

- Downside:
  - If you have 100,000 states you are solving a 100,00 by 100,000 system of equations

Value Iteration

- Define:

$U^1(s_i)$ = Expected discounted sum of rewards over next 1 time step

$U^2(s_i)$ = Expected discounted sum of rewards during next 2 time step

$U^3(s_i)$ = Expected discounted sum of rewards during next 3 time step

$\vdots$

$U^k(s_i)$ = Expected discounted sum of rewards during next k time step

Value Iteration

- Define:

$U^1(s_i)$ = Expected discounted sum of rewards over next 1 time step

$U^2(s_i)$ = Expected discounted sum of rewards during next 2 time step

$U^3(s_i)$ = Expected discounted sum of rewards during next 3 time step

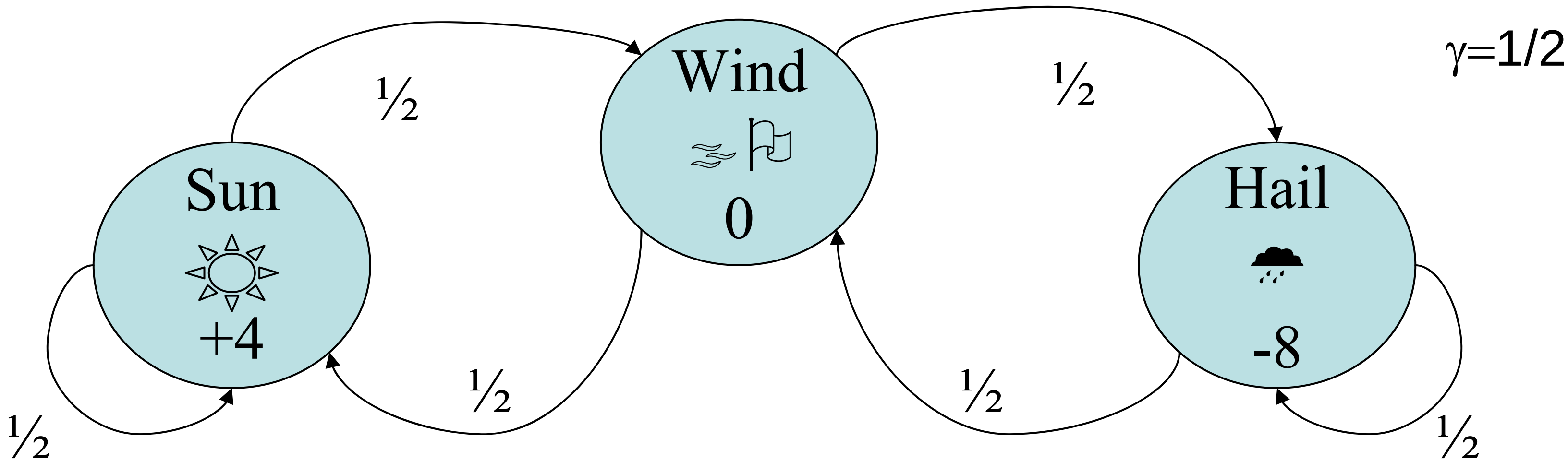
$U^k(s_i)$ = Expected discounted sum of rewards during next k time step

$$U^1(S_i)=r_i$$

$$U^2(S_i)=r_i+\gamma\Sigma_{j=1}^{n}\ p_{ij}U^1(s_j)$$

$$U^{k+1}(S_i)=r_i+\gamma\Sigma_{j=1}^{n}\ p_{ij}U^k(s_j)$$

Let's Do Value Iteration



k	U^k(sun)	U^k(wind)	U^k(hail)
1			
2			
3			
4			
5			

Value Iteration

- Compute $U^1(s_i)$ for each i
- Compute $U^2(s_i)$ for each i

- Compute $U^k(s_i)$ for each i
- As $k \rightarrow \infty$ $U^k(s_i) \rightarrow U^*(s_i)$
  - Why?
- When to stop? When:

a small number

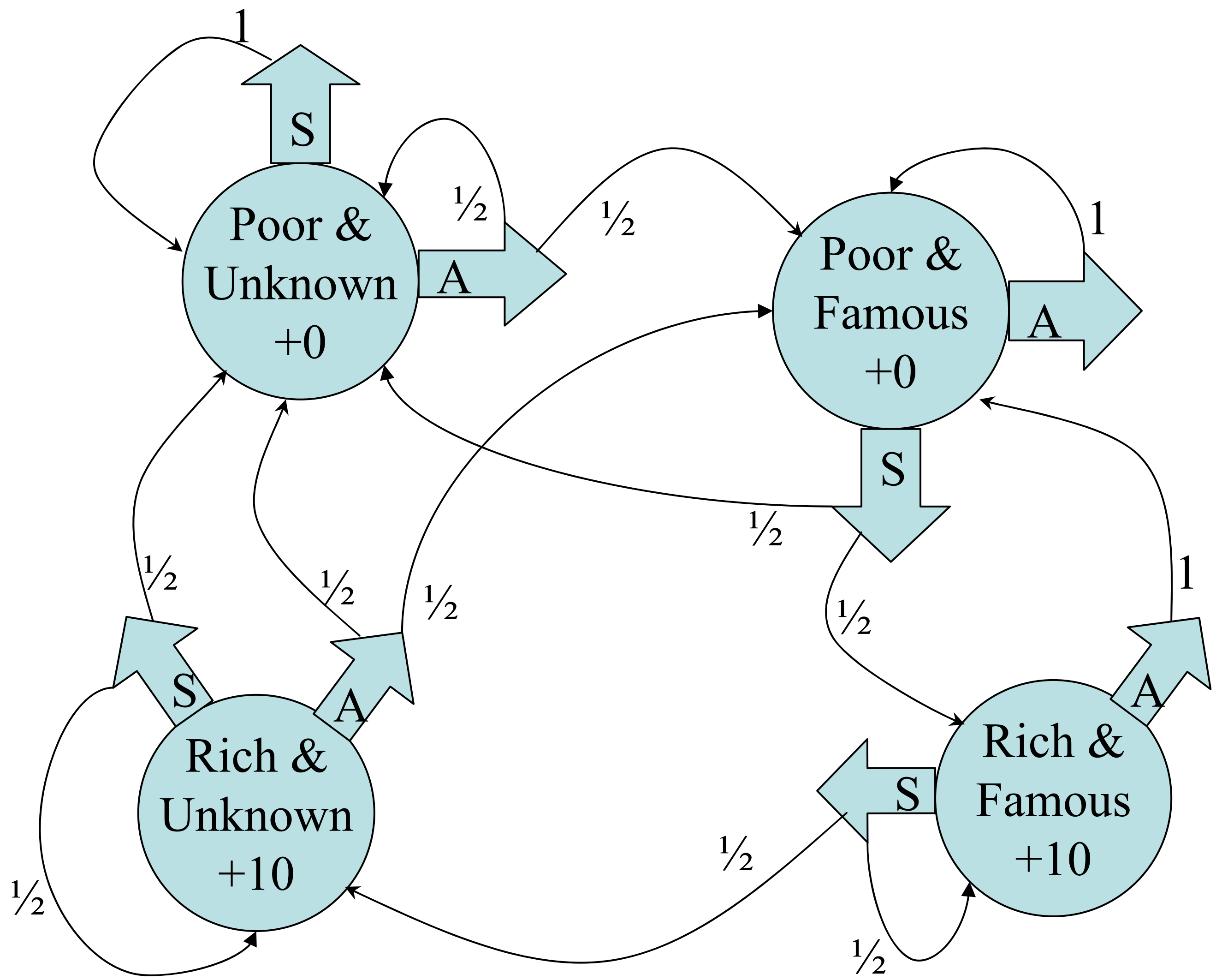
$$\max \left| U^{k+1}(s_i) - U^k(s_i) \right| < \varepsilon$$

- This is faster than matrix inversion (N^3 style) **IF** the transition matrix is sparse.

Making sequential decisions

- Markov chains
  - To highlight the Markov property
- Discounted rewards
  - Value iteration
- Markov decision processes

A Markov Decision Process



$\gamma = 0.9$

You own a company

In every state you must choose between **S**aving money or **A**dvertising

Markov Decision Processes (MDPs)

- Has a set of states $\{s_1, s_2, \ldots s_n\}$
- Has a set of actions $\{a_1, \ldots, a_m\}$
- Each state has a reward $\{r_1, r_2, \ldots r_n\}$
- Has a transition probability function

$$P_{ij}^{k} = (\text{Next}=s_j \mid \text{This}=s_i \text{ and I take action } a_k)$$

- ON EACH STEP...
  0. Assume your state is s_i
  1. You get given reward r_i
  2. Choose action a_k
  3. You will move to state s_j with probability P_{ij}^{k}
  4. All future rewards are discounted by γ

Planning in MDPs

- The goal of an agent in an MDP is to be rational
  - Maximize its expected utility
  - But maximizing immediate utility is not good enough
    - Great action now can lead to death later...
- Goal is to maximize its long term reward
  - Do this by finding a policy that has high return

Policies

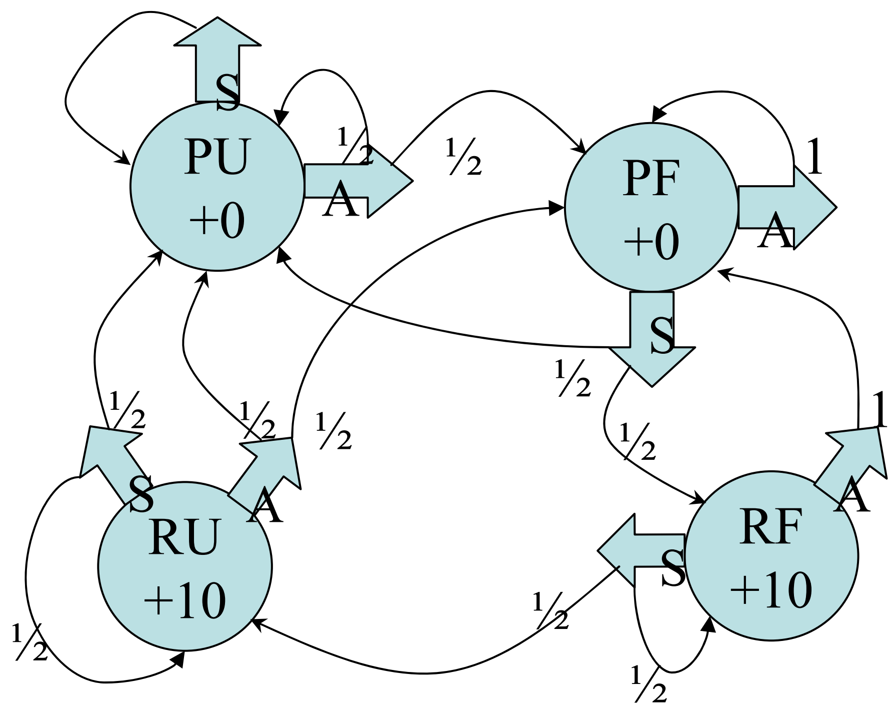
- A policy is a mapping from states to actions

Policy 1

PU	S
PF	A
RU	S
RF	A

Policy 2

PU	A
PF	A
RU	A
RF	A



How many policies?

Fact

- For every MDP there exists an optimal policy

- It is the policy such that for every possible start state there is no better option than to follow the policy

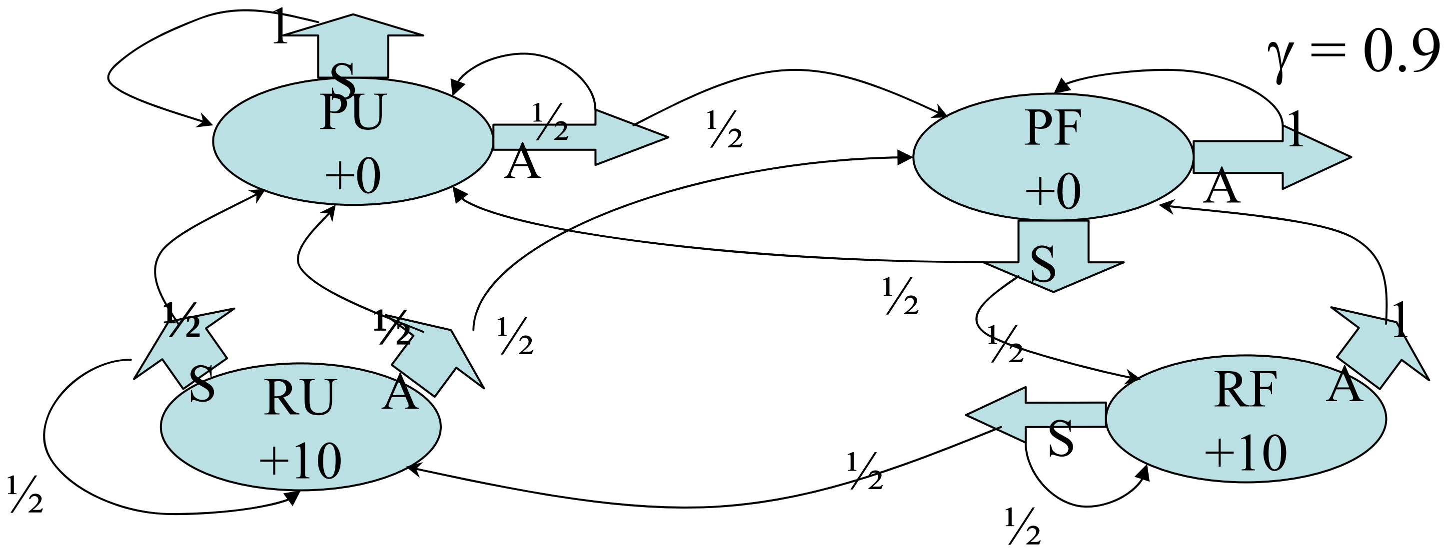
$\boxed{\text{Our goal: To find this policy!}}$

Finding the optimal policy

- First idea:
  - Simply run through all possible policies and select the best
- But we can do better!

Optimal Value Function

- Define $U^*(s_i)$ to be the expected discounted future rewards
  - Starting from state s_i, assuming we use the optimal policy
- Define $U^t(s_i)$ = maximum possible sum of discounted rewards I can get if I start at state S_i and I live for t time steps
  - Note: $U^1(s_i) = r_i$



t	$U^t(PU)$	$U^t(PF)$	$U^t(RU)$	$U^t(RF)$
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	8.55	16.53	25.08
4	4.76	12.20	18.35	28.72
5	7.63	15.07	20.40	31.18
6	10.22	17.46	22.61	33.21

Bellman's Equation

$$\boxed{U^{t+1}(s_i)=\max_k \left[r_i+\gamma\sum_{j=1}^{n} P_{ij}^{k}\, U^{t}(s_j)\right]}$$

- Now we can do Value Iteration!
  - Compute $U^1(s_i)$ for all i
  - Compute $U^2(s_i)$ for all i
  - ...
  - Compute $U^t(s_i)$ for all i
  - Until converges
    - $\max_i |U^{t+1}(s_i)-U^t(s_i)|<\varepsilon$

aka Dynamic Programming

Finding the optimal policy

- Compute $U^*(s_i)$ for all i using value iteration

- Define the best action in state s_i as

$$\arg\max_k [r_i + \gamma \Sigma_j P_{ij}^k U^*(s_j)]$$

Policy iteration

- There are other ways of finding the optimal policy
  - Policy iteration

- Alternates between two steps
  - Policy evaluation
    - Given π_i, calculate $U_i = U^{\pi}$
  - Policy improvement
    - Calculate a new π_{i+1} using one step lookahead

Policy iteration algorithm

- Start with random policy π
- Repeat
  - Compute long term reward for each s_i, using π
  - For each state s_i
    - If

$$\max_k [r_i + \gamma \Sigma_j P_{ij}^k U^*(s_j)] > r_i + \gamma \Sigma_j P_{ij}^{\pi(s_i)} U^*(s_j)$$

Then $\pi(s_i) \leftarrow \text{argmax}_k\ [r_i + \gamma \Sigma_j\ P_{ij}^k U^*(s_j)]$

  - Until you stop changing the policy

Summary

- MDP's describe planning tasks in stochastic worlds
- Goal of the agent is to maximize its expected return
- Value functions estimate the expected return
- In finite MDP there is a unique optimal policy
  - Dynamic programming can be used to find it

Summary

- Good news
  - finding optimal policy is polynomial in number of states
- Bad news
  - finding optimal policy is polynomial in number of states
- Number of states tends to be very very large
  - exponential in number of state variables
- In practice, can handle problems with up to 10 million states

Extensions

- In "real life" agents may not know what state they are in
  - Partial observability
- **P**artially **O**bservable **MDP**s
  - Has a set of states $\{s_1, s_2, \ldots s_n\}$
  - Has a set of actions $\{a_1, \ldots, a_m\}$
  - **Has set of observations $O=\{o_1, \ldots, o_k\}$**
  - Each state has a reward $\{r_1, r_2, \ldots r_n\}$
  - Has a transition probability function $P(s_t|a_{t-1}, s_{t-1})$
  - **Has observation model $P(o_t|s_t)$**
  - Has discount factor γ

POMDPs

- The agent maintains a belief state, b
  - Probability distribution over all possible states
  - b(s) is the probability assigned to state s
- Insight: optimal action depends only on agent's current belief state
  - Policy is mapping from belief states to actions

POMDPs

- Decision cycle of agent
  - Given current b, execute action $a=\pi^*(b)$
  - Receive observation o
  - Update current belief state
    - $b'(s')=\alpha\, O(o|s')\sum_s P(s'|a,s)b(s)$
    - α is a normalizing factor

- Possible to write a POMDP as an MDP by summing over all actual states s' that the agent might reach
  - $Pr(b'|a,b)=\sum_o P(b'|o,a,b)\sum_{s'} O(o|s')\sum_s P(s'|a,s)b(s)$

POMDP's

- Complications
  - Our (new) MDP has a continuous state space
  - In general, finding (approximately) optimal properties is difficult (PSPACE-hard)
  - Problems with even a few dozen states are often infeasible
    - New techniques, take advantage of structure...