Markov Decision Processes

CS 486/686: Introduction to Artificial Intelligence

Outline

- Markov Chains
- Discounted Rewards
- Markov Decision Processes
  - Value Iteration
  - Policy Iteration

Markov Chains

- Simplified version of snakes and ladders
- 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 7
- Winner is the one who gets to 11 first

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

Markov Chain

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

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

Markov Chain

- 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 in 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***

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

Markov Chain: Formal Representation

- State space S={0,1,2,3,4,5,6,7,8,9,10,11}
- Transition probability matrix P

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

P_{ij}=Prob(Next=s_j| This=s_i)

Discounted Rewards

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

30+30+30+30+…=

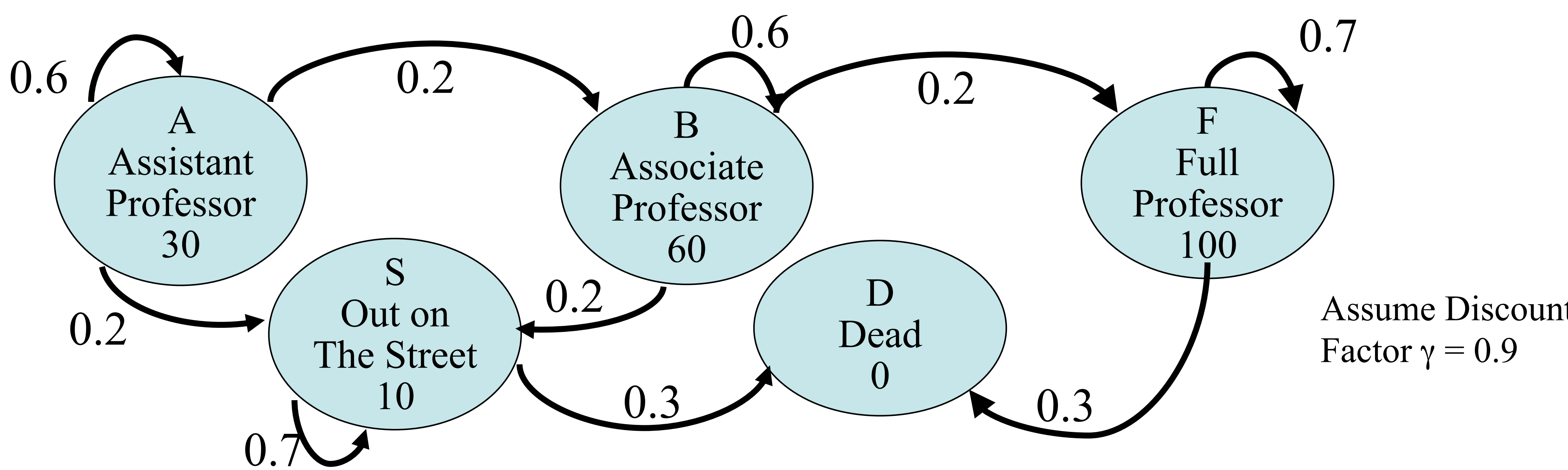
Discounted Rewards

- A reward in the future is not worth quite as much as a reward now
  - Because of chance of inflation
  - Because of chance of obliteration
- Example:
  - Being promised $10000 next year is worth only 90% as much as receiving $10000 now
- Assuming payment n years in the future is worth only $(0.9)^n$ of payment now, what is the assistant professor’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



- U_A=Expected discounted future rewards starting in state A
- U_B=Expected discounted future rewards starting in state B
- U_F=Expected discounted future rewards starting in state F
- U_S=Expected discounted future rewards starting in state S
- U_D=Expected discounted future rewards starting in state D

Markov System of Rewards

- Set of states S={s1,s2,...,sn}
- Each state has a reward {r1,r2,...,rn}
- Discount factor γ, $0<\gamma<1$
- Transition probability matrix, P

$$P = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{bmatrix}$$

P_{ij} = Prob(Next = s_j | This = s_i)

On each step:

•Assume state is s_i

•Get reward r_i

•Randomly move to state s_j with probability P_{ij}

•All future rewards are discounted by γ

Solving a Markov Process

- Write $U^*(s_i)$ = expected discounted sum of future rewards starting at state s_i
  - $U^*(s_i)=r_i+\gamma(P_{i1}U^*(s_i)+P_{i2}U^*(s_2)+\ldots+P_{in}U^*(s_n))$

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

Closed form: $U=(I-\gamma P)^{-1}R$

Solving a Markov System using Matrix Inversion

- **Upside**:
  - You get an exact number!
- **Downside**:
  - If you have *n* states you are solving an *n* by *n* 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 over next 2 time steps
  - $U^3(s_i)$=Expected discounted sum of rewards over next 3 time steps
  - ...
  - $U^k(s_i)$=Expected discounted sum of rewards over next k time steps

Value Iteration

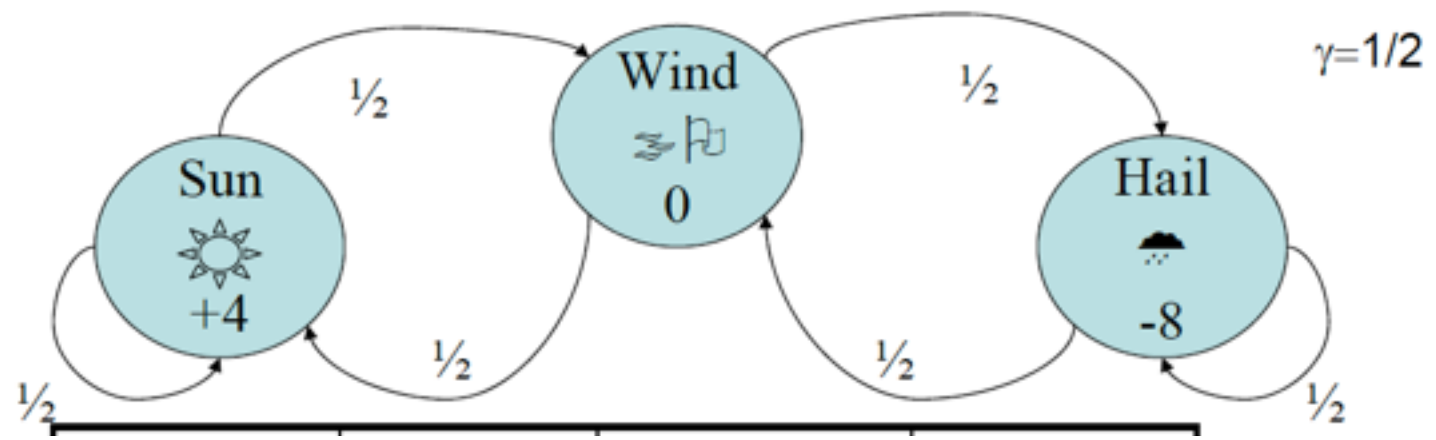
- Define
  - $U^1(s_i)$=Expected discounted sum of rewards over next 1 time step
  - $U^2(s_i)$=Expected discounted sum of rewards over next 2 time steps
  - $U^3(s_i)$=Expected discounted sum of rewards over next 3 time steps
  - ...
  - $U^k(s_i)$=Expected discounted sum of rewards over next k time steps

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

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

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

Example: Value Iteration



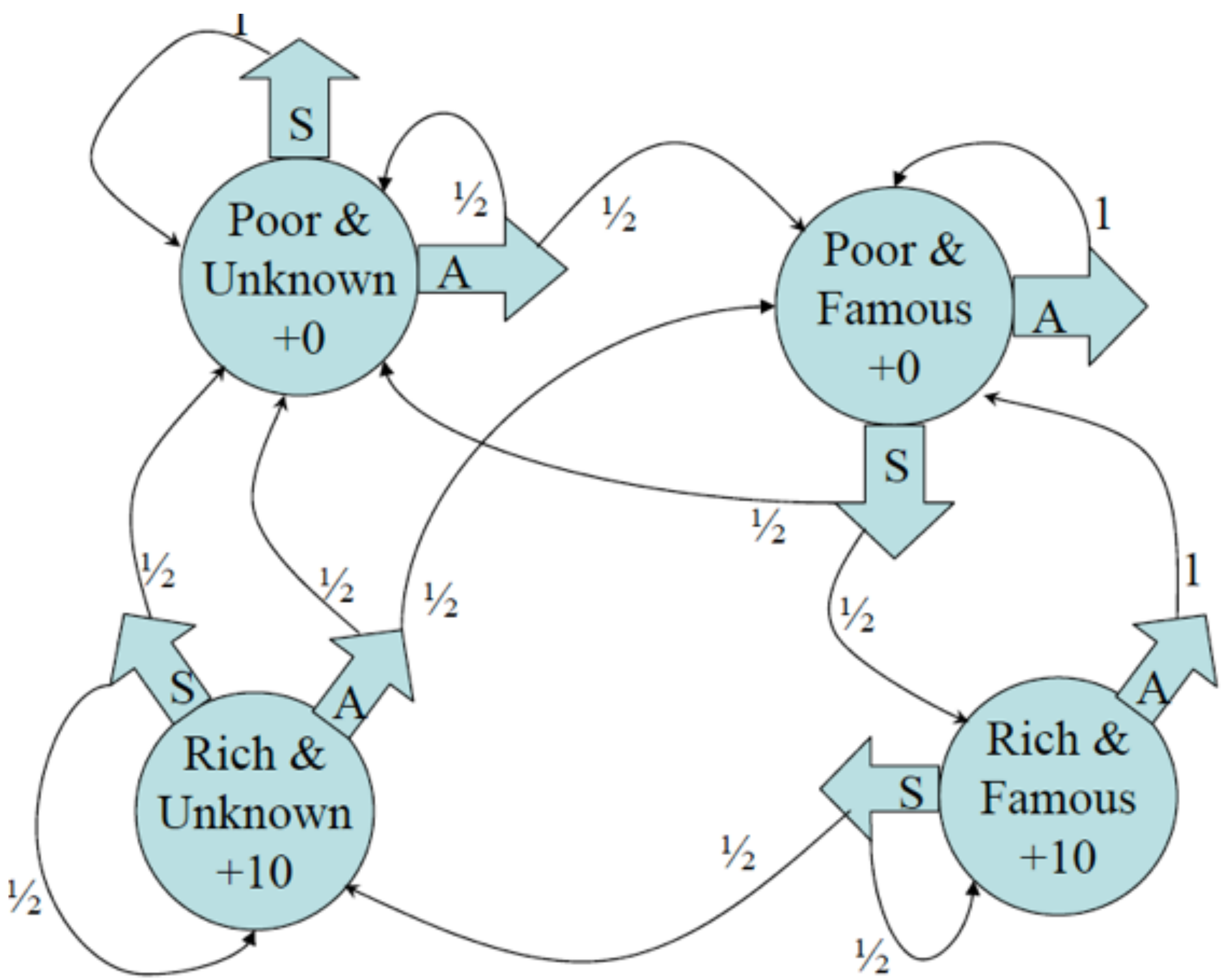
k	U^k(sun)	U^k(wind)	U^k(hail)
1			
2			
3			
4			
5			

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 $k \to \infty$, $U^k(s_i) \to U^*(s_i)$
- When to stop?
  - max $|U^{k+1}(s_i)-U^k(s_i)|<\varepsilon$

- This is often faster than matrix inversion

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 Process

- Set of states $S=\{s_1,s_2,...,s_n\}$
- Each state has a reward $\{r_1,r_2,...,r_n\}$
- **Set of actions $\{a_1,...,a_m\}$**
- Discount factor γ, $0<\gamma<1$
- Transition probability function , P

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

On each step:
- Assume state is s_i
- Get reward r_i
- Choose action a_k
- Randomly move to state s_j with probability P_{ij}^k
- 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 certain death tomorrow
- Goal is to maximize its long term reward
  - Do this by finding a **policy** that has high return

Policies

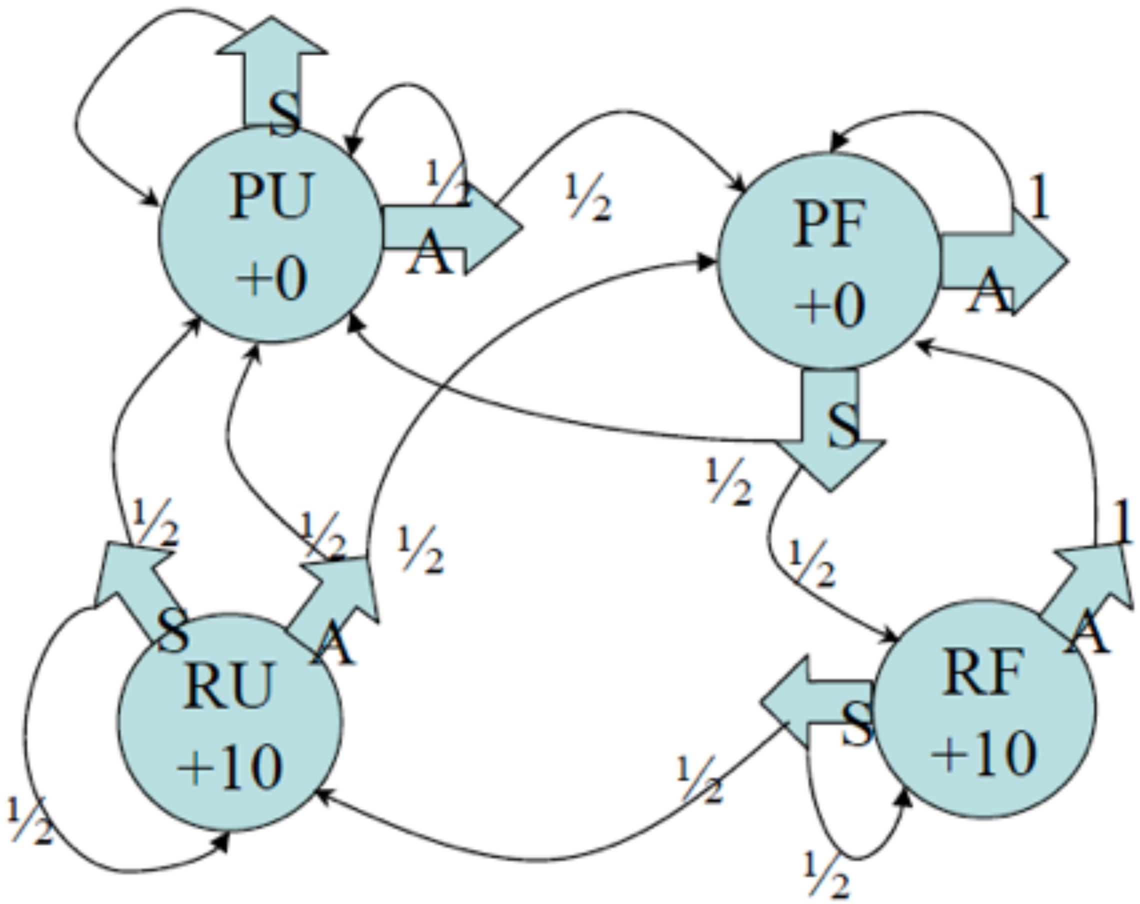
- A policy is a mapping from states to actions

Policy 1

PU	S
PF	A
RU	S
RF	A

Policy 2

PU	A
PF	A
RU	A
RF	A



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 that to follow the policy

Our goal: To find this policy!

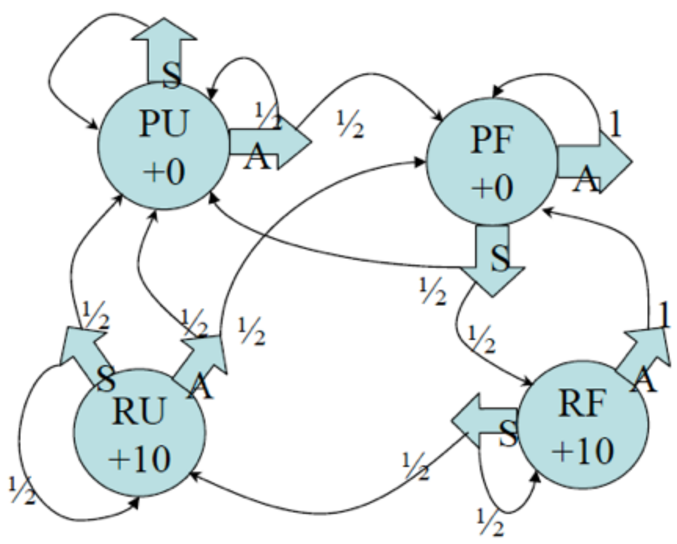
Finding the Optimal Policy

- Naive approach:
  - Run through all possible policies and select the best

Optimal Value Function

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

Example



$\gamma = 0.9$

t	$V^t(PU)$	$V^t(PF)$	$V^t(RU)$	$V^t(RF)$
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

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

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

aka Dynamic Programming

Finding the Optimal Policy

- Compute $V^*(s_i)$ for all i using value iteration
- Define the best action in state s_i as

$$\text{argmax}_k[r_i+\gamma\sum_j P_{ij}^k V^*(s_j)]$$

Policy Iteration

- There are other ways of finding the optimal policy
- Policy Iteration
  - Alternates between two steps
    - **Policy evaluation**: Given π, compute $V_i = V^{\pi}$
    - **Policy improvement**: Calculate a new π_{i+1} using 1-step lookahead

Policy Iteration Algorithm

- Start with random policy π
- Repeat until you stop changing the policy
  - Compute long term reward for each s_i, using π
  - For each state s_i

If

$$\max_k \left[r_i + \gamma \sum_j P_{i,j}^k V^*(s_j) \right] > r_i + \gamma \sum_j P_{i,j}^{\pi(s_i)} V^*(s_j)$$

Then

$$\pi(s_i) \leftarrow \arg\max_k \left[r_i + \gamma \sum_j P_{i,j}^k V^*(s_j) \right]$$

Summary

- MDPs describe planning tasks in stochastic worlds
- Goal of the agent is to maximize its expected return
- Value functions estimate the expected return
- In finite MDPs there is a unique optimal policy
  - Dynamic programing 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
- Partially Observable MDPs (POMDPs)
  - Set of states
  - Set of actions
  - Each state has a reward
  - Transition probability function $P(s_t|a_{t-1},s_{t-1})$
  - **Set of observations $O=\{o_1,\ldots,o_k\}$**
  - **Observation model $P(o_t|s_t)$**

POMDPs

- 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 a mapping from belief states to actions

POMDPs

- Decision cycle of an agent
  - Given current b, execute action $a=\pi^*(b)$
  - Receive observation o
  - Update current belief state
    - $b'(s')=\alpha O(o|s')\Sigma sP(s'|a,s)b(s)$
- Possible to write a POMDP as an MDP by summing over all actual states s' that an agent might reach
  - $P(b'|a,b)=\Sigma_o P(b'|o,a,b)\Sigma_{s'} O(o|s')\Sigma_s P(s'|a,s)b(s)$

POMDPs

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