Introduction to Machine Learning

CS307 --- Fall 2022

Markov Decision Processes

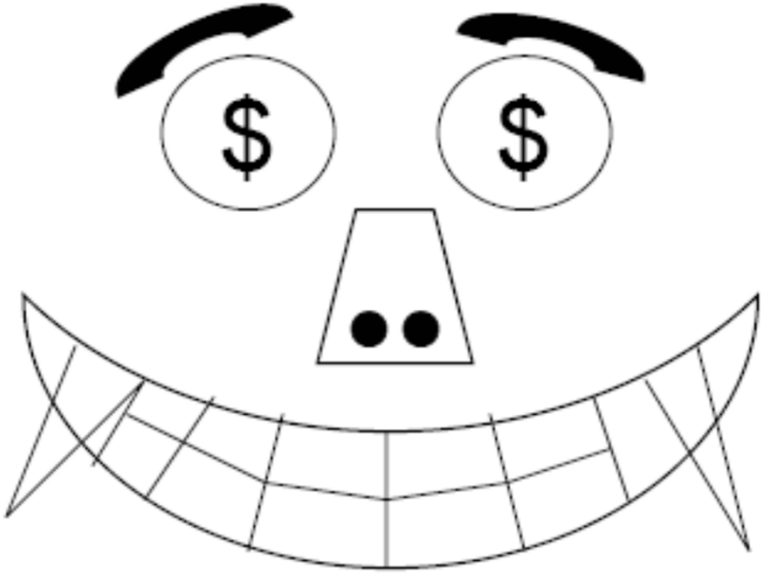
Reading: Sections 16.1-16.6, 17.1-17.4, R&N

Rewards

An assistant professor gets paid, say, 20K per year.

How much, in total, will the A.P. earn in his life?

20 + 20 + 20 + 20 + 20 + … = Infinity



What's wrong with this argument?

Horizon Problem

- The problem is that we did not put any limit on the "future", so this sum can be infinite.

- This definition is useless unless we consider a finite time horizon.

- But, in general, we don't have a good way to define such a time horizon.

Discounted Rewards

"A reward (payment) in the future is not worth quite as much as a reward now."

- Because of chance of obliteration
- Because of inflation

Example:

Being promised $10,000 next year is worth only 90% as much as receiving $10,000 right now.

Assuming payment *n* years in future is worth only $(0.9)^n$ of payment now, what is the A.P.'s Discounted Sum of Future Rewards?

Discount Factors

People in economics and probabilistic decision making do this all the time.

The discounted sum of future rewards 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) +

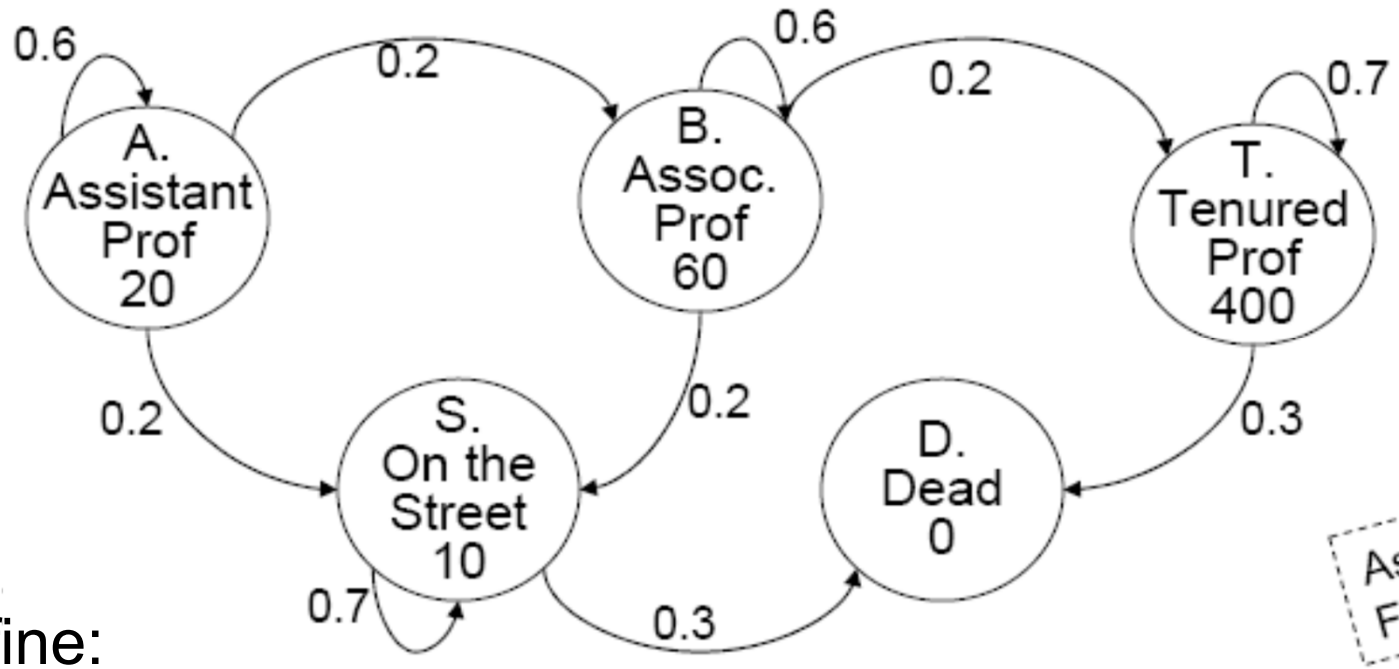
:

: (infinite sum)

Discounting

- Always converges if $\gamma < 1$ and the reward function, R(.), is bounded
- γ close to 0 → instant gratification, don't pay attention to future reward
- γ close to 1 → extremely conservative, consider profits/losses no matter how far in the future
- The resulting model is the *discounted reward*
- Prefers expedient solutions (models impatience)
- Compensates for uncertainty in available time (models mortality)

The Academic Life



Assume Discount Factor $\gamma = 0.9$

Define:

J_A = expected discounted future rewards starting in state A

J_B = expected discounted future rewards starting in state B

J_T = expected discounted future rewards starting in state T

J_S = expected discounted future rewards starting in state S

J_D = expected discounted future rewards starting in state D

How do we compute J_A, J_B, J_T, J_S, J_D ?

A Markov System with Rewards

- Has a set of states $\{ S_1 \; S_2 \; \cdot\cdot \; S_N \}$
- Has a transition probability matrix

$$P = \begin{pmatrix} P_{11} & P_{12} & \cdot\cdot & P_{1N} \\ P_{21} & \cdot. & & \\ : & & & \\ P_{N1} & \cdot\cdot & & P_{NN} \end{pmatrix} \qquad P_{ij} = \text{Prob}(\text{NextState} = S_j \mid \text{ThisState} = S_i)$$

- Each state has a reward. $\{ r_1 \; r_2 \; \cdot\cdot \; r_N \}$
- There's a discount factor γ. $0 < \gamma < 1$

On each time step:

1. Call current state S_i
2. Receive reward r_i
3. Randomly move to another state S_j with probabilty P_{ij}
4. All future rewards are discounted by γ

Solving a Markov System

Write $J^*(S_i)$ = expected discounted sum of future rewards starting in state S_i

$J^*(S_i) = r_i + \gamma$(expected future rewards starting from the next state)

$= r_i + \gamma(P_{i1}J^*(S_1) + P_{i2}J^*(S_2) + \cdots + P_{iN}J^*(S_N))$

Using vector notation, write

$$J = \begin{pmatrix} J^*(S_1) \\ J^*(S_2) \\ \vdots \\ J^*(S_N) \end{pmatrix} \quad R = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{pmatrix} \quad P = \begin{pmatrix} P_{11} & P_{12} & \cdot\cdot & P_{1N} \\ P_{21} & \cdot. & & \\ \vdots & & & \\ P_{N1} & P_{N2} & \cdot\cdot & P_{NN} \end{pmatrix}$$

Question: can you write a closed form expression for J in terms of R, P and γ ?

Solving a Markov System with Matrix Inversion

- Upside: You get an exact answer
- Downside: If you have 100,000 states you have to solve a system of 100,000 equations with 100,000 variables.

Value Iteration: another way to solve a M.S.

Define

$J^1(S_i)$ = expected discounted sum of rewards over the next 1 time step

$J^2(S_i)$ = expected discounted sum of rewards over the next 2 time steps

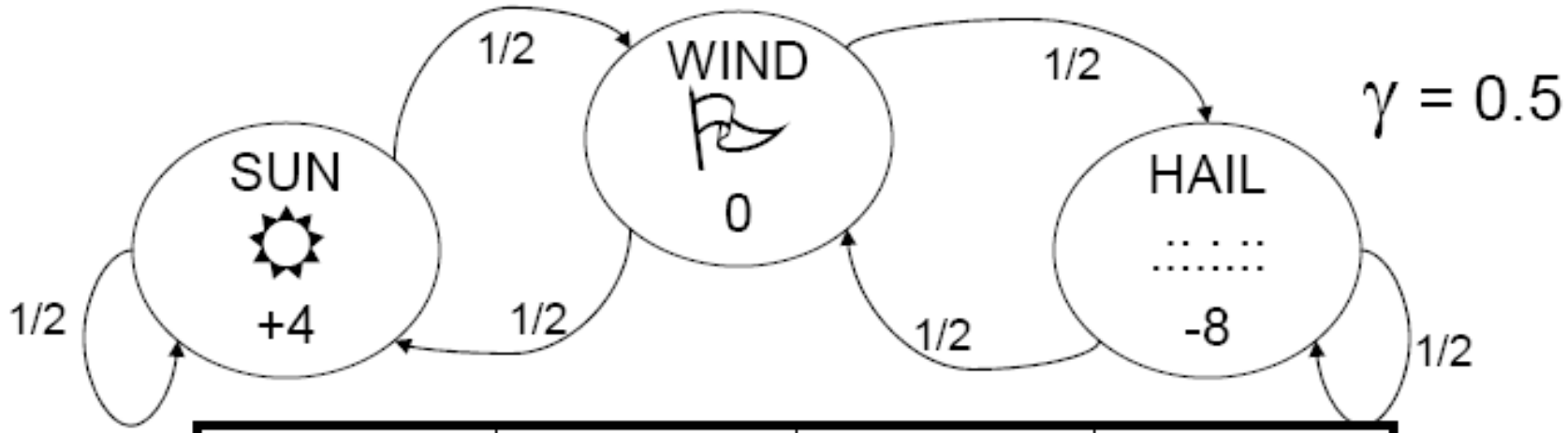
$J^3(S_i)$ = expected discounted sum of rewards over the next 3 time steps

:

$J^k(S_i)$ = expected discounted sum of rewards over the next *k* time steps

$J^1(S_i)$ =	(what?)
$J^2(S_i)$ =	(what?)
.	
$J^{k+1}(S_i)$ =	(what?)

Let's do Value Iteration



k	J^k(SUN)	J^k(WIND)	J^k(HAIL)
1			
2			
3			
4			
5			

Value Iteration for Solving Markov Systems

- Compute $J^1(S_i)$ for each *j*
- Compute $J^2(S_i)$ for each *j*

  :

- Compute $J^k(S_i)$ for each *j*

  As $k \rightarrow \infty$ $J^k(S_i) \rightarrow J^*(S_i)$ Why?

When to stop?

When $\max_i | J^{k+1}(S_i) - J^k(S_i) | < \xi$

This is faster than matrix inversion

but only if the transition matrix is sparse

Different Types of Markov Models

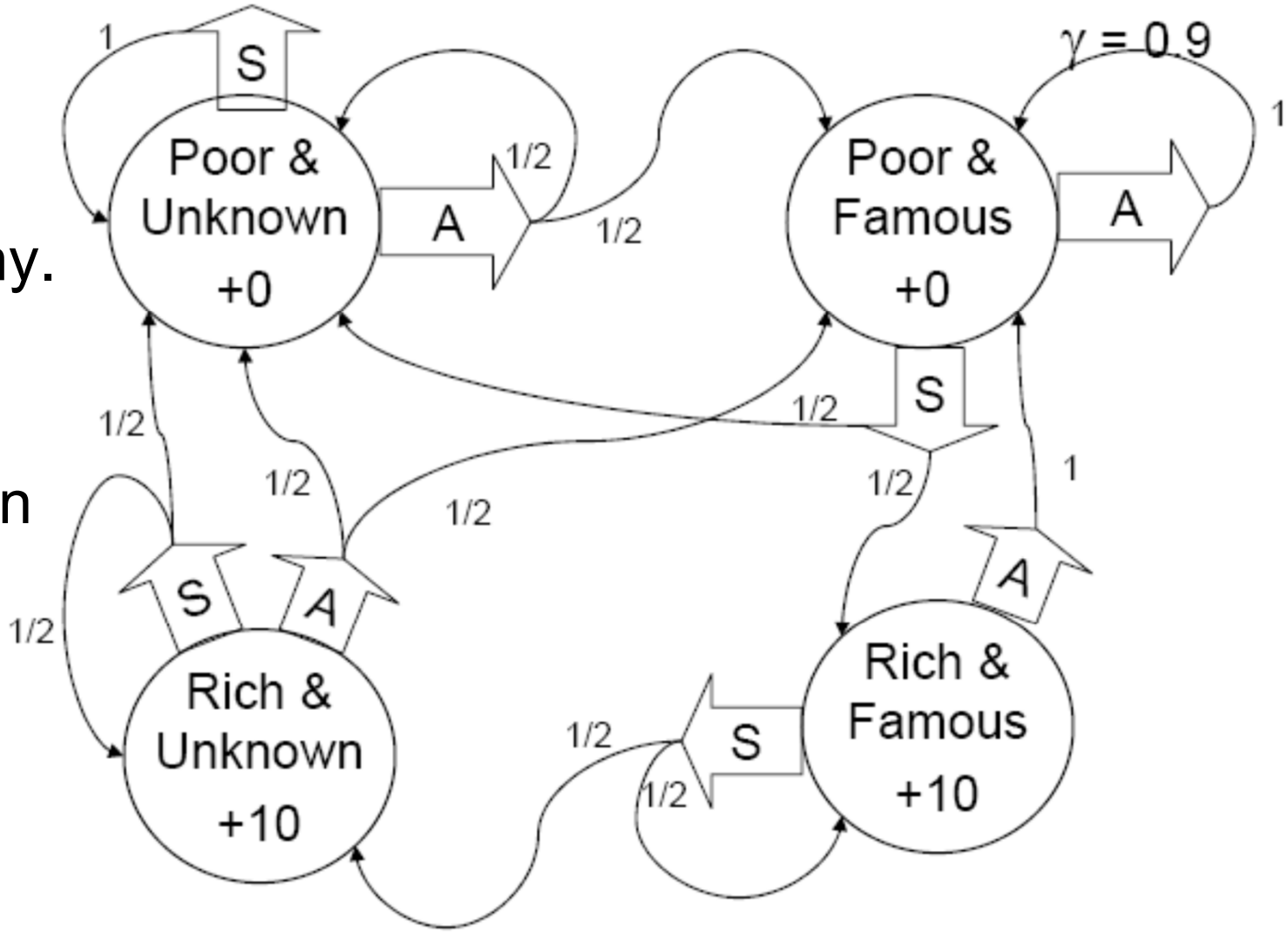
State	No Control	Control
Fully observable	**Markov Systems**	**MDPs**
Hidden	**HMMs**	**POMDPs**

Markov Decision Processes: An Example

- I run a startup company
- I can choose to either save money or spend money on advertising
- If I advertise, I may become famous (50% prob.) but will spend money so I may become poor
- If I save money, I may become rich (50% prob.), but I may also become unknown because I don't advertise
- What should I do?

A Markov Decision Process

You run a startup company. In every state you must choose between Saving money or Advertising.



Markov Decision Processes

An MDP has…

- A set of states $\{s_1 \cdots s_N\}$
- A set of actions $\{a_1 \cdots a_M\}$
- A set of rewards $\{r_1 \cdots r_N\}$ (one for each state)
- A transition probability function

$$P_{ij}^{k} = \text{Prob}\left(\text{Next} = j \middle| \text{This} = i \text{ and I use action } k\right)$$

On each time step:

1. Call current state S_i
2. Receive reward r_i
3. Choose action $\in \{a_1 \cdots a_M\}$
4. If you choose action a_k you move to state S_j with probability P_{ij}^{k}
5. All future rewards are discounted by γ

What we are looking for: *Policy*

A policy is a mapping from states to actions.

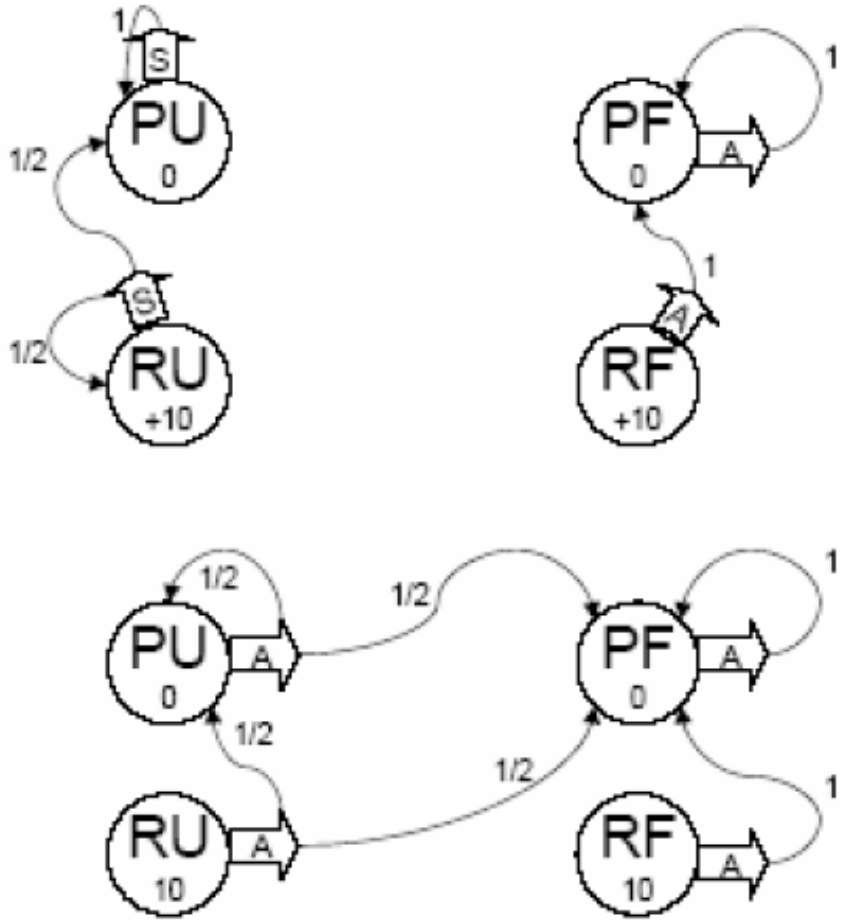
Examples

Policy Number 1:

STATE →	ACTION
PU	S
PF	A
RU	S
RF	A

Policy Number 2:

STATE →	ACTION
PU	A
PF	A
RU	A
RF	A



- How many policies are there?
- Which one is the best policy and how to compute it?

Optimal Decision Policy

- *Policy* π = Mapping from states to actions $\pi(s) = a$
  → Which action should I take in each state?

- Intuition: π encodes the best action that we can take from any state to maximize future rewards

- *Optimal Policy* = The policy π^* that maximizes the expected utility $J(s)$ of the sequence of states generated by π^*, starting at s

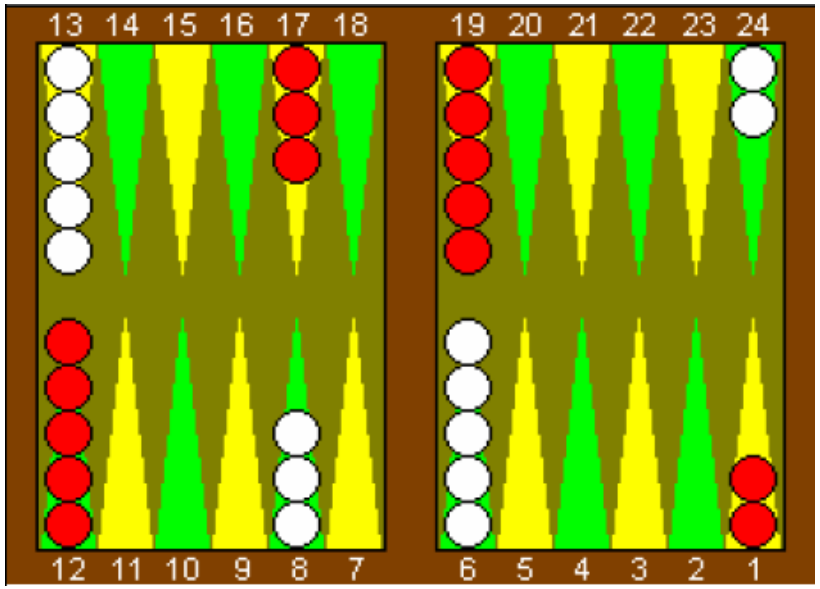
Example: Finance and Business

- States: Status of the company (cash reserves, inventory, etc.)
- Actions: Business decisions (advertise, acquire other companies, roll out product, etc.)
- Uncertainty due all the external uncontrollable factors (economy, shortages, consumer confidence)
- Optimal policy: The policy for making business decisions that maximizes the expected future profits

Example: Robotics

- States are 2-D positions
- Actions are commanded motions (turn by x degrees, move y meters)
- Uncertainty comes from the fact that the mechanism is not perfect (slippage, etc.) and does not execute the commands exactly
- Reward when avoiding forbidden regions (for example)
- Optimal policy: The policy that minimizes the cost to the goal

Example: Games



- States: Number of white and black checkers at each location
- Note: Number of states is huge, on the order 10^{20} states.
- Branching factor prevents direct search
- Actions: Set of legal moves from any state
- Uncertainty comes from the roll of the dice
- Reward computed from number of checkers in the goal quadrant
- Optimal policy: The one that maximizes the probability of winning the game

Key Result

- For every MDP, there exists an optimal policy
- There is no better option (in terms of expected sum of rewards) than to follow this policy, but there could be multiple optimal policies
- How to compute the optimal policy?

Computing the Optimal Policy

First idea:

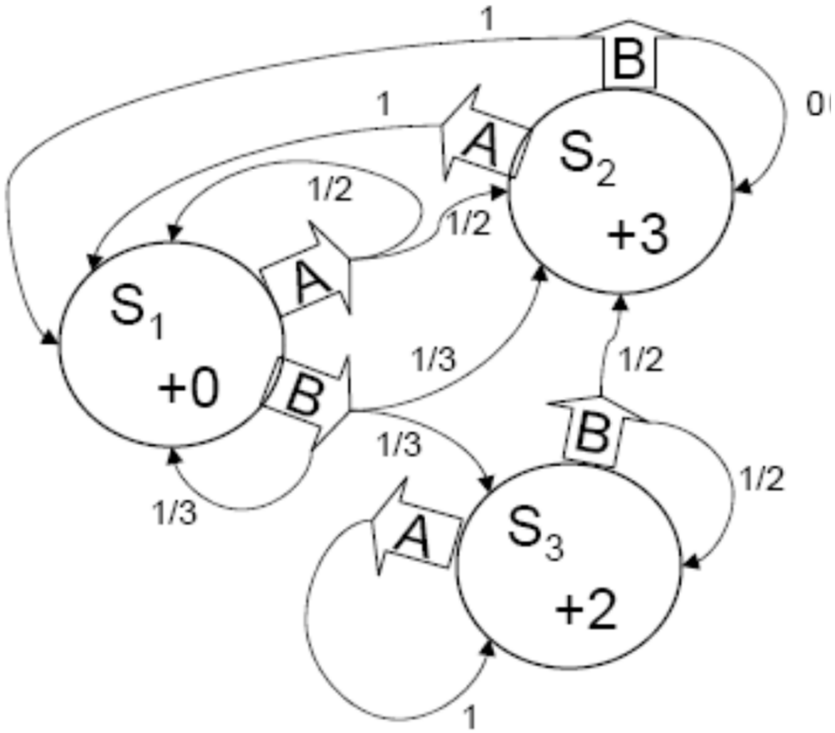
Run through all possible policies.

Select the best.

What's the problem?

Optimal Value Function

Define $J^*(S_i)$ = Expected Discounted Future Rewards, starting from state S_i, assuming we use the optimal policy



Question

What (by inspection) is an optimal policy for that MDP? (assume $\gamma = 0.9$)

What is $J^*(S_1)$?

What is $J^*(S_2)$?

What is $J^*(S_3)$?

Computing the Optimal Value Function with Value Iteration

Define

$J^k(S_i)$ = Maximum possible expected sum of discounted rewards I can get if I start at state S_i and I live for k time steps.

Note that $J^1(S_i) = r_i$

Bellman's Equation

$$J^{n+1}(S_i) = \max_k \left[r_i + \gamma \sum_{j=1}^{N} P_{ij}^k J^n(S_j) \right]$$

Value Iteration for solving MDPs

- Compute $J^1(S_i)$ for all *i*
- Compute $J^2(S_i)$ for all *i*
- :
- Compute $J^n(S_i)$ for all *i*

….. until converged

converged when $\max_i \left| J^{n+1}(S_i) - J^n(S_i) \right| < \xi$

Finding the Optimal Policy

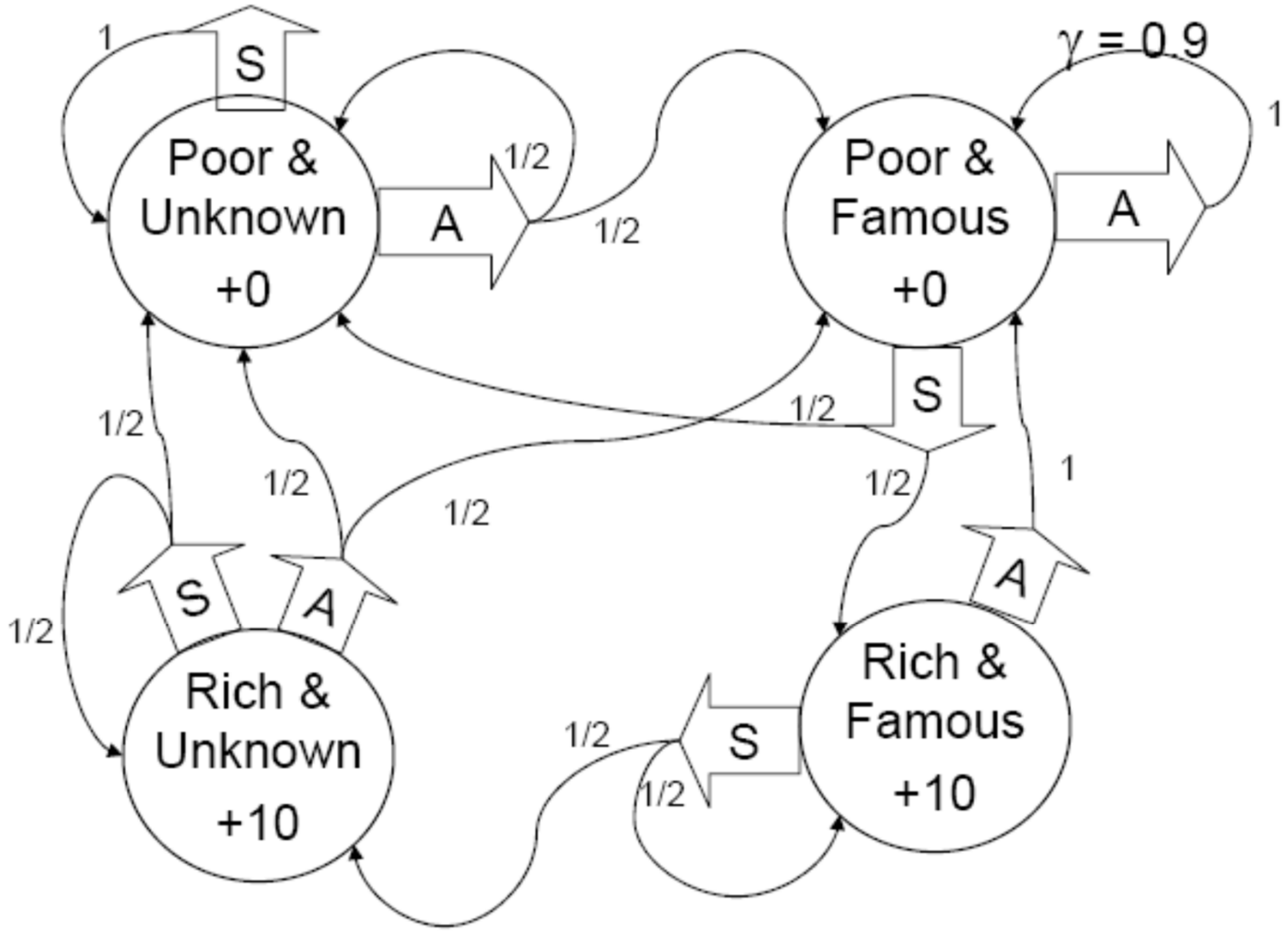
1. Compute $J^*(S_i)$ for all *i* using Value Iteration

2. Define the best action in state S_i as

$$\arg\max_k \left[r_i + \gamma \sum_j P_{ij}^k J^*(S_j) \right]$$

What's the optimal policy in our example?

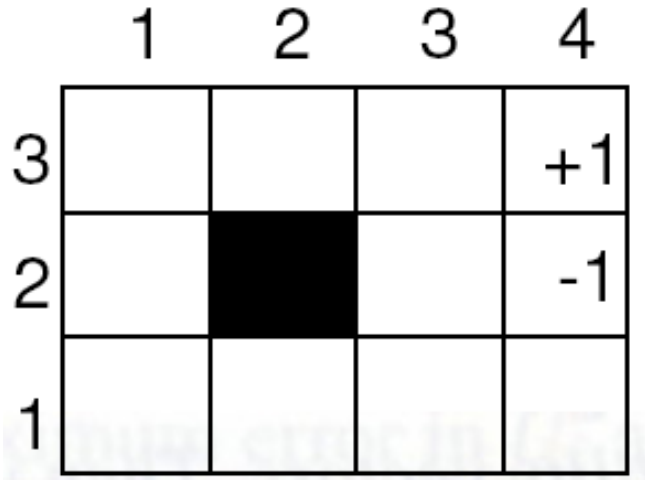
Computing $J^k(S_i)$ for our example

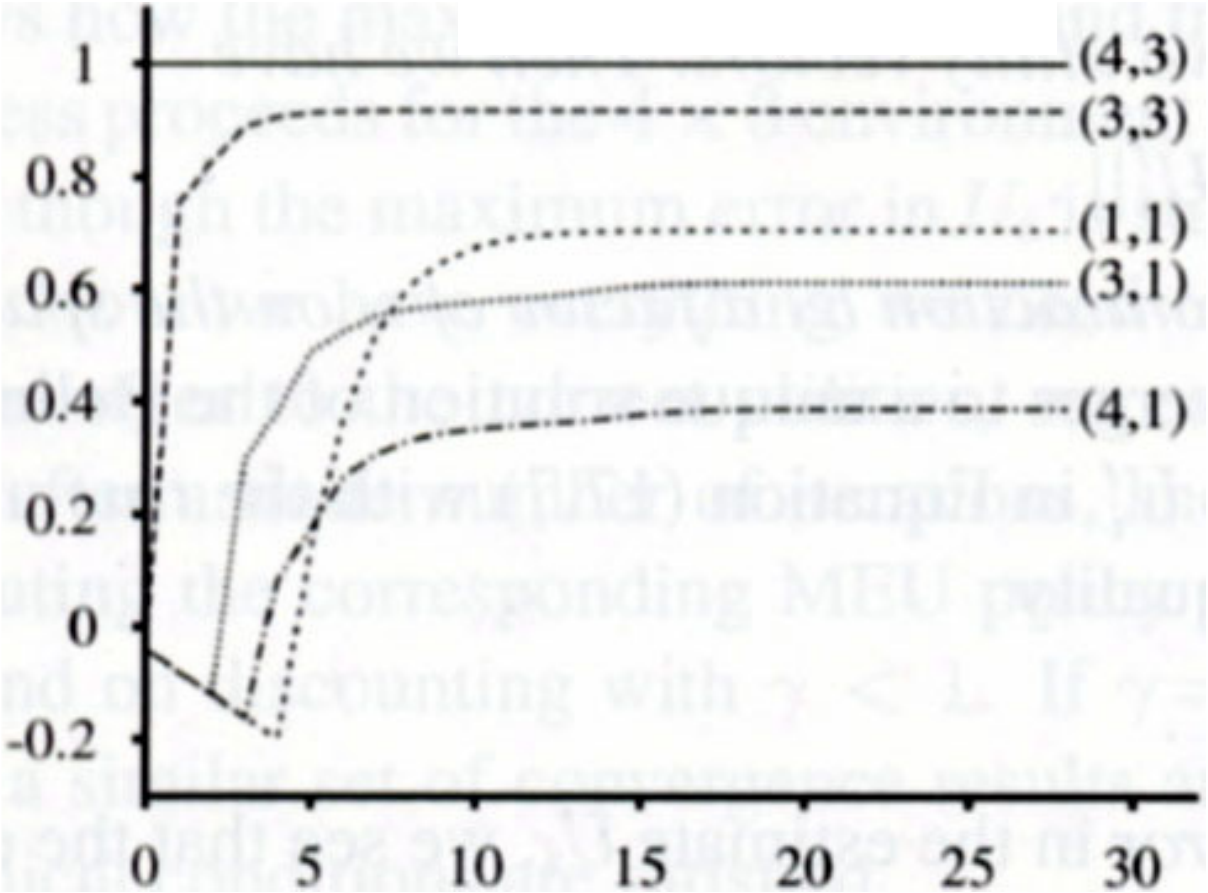


Compute $J^k(S_i)$ for our example

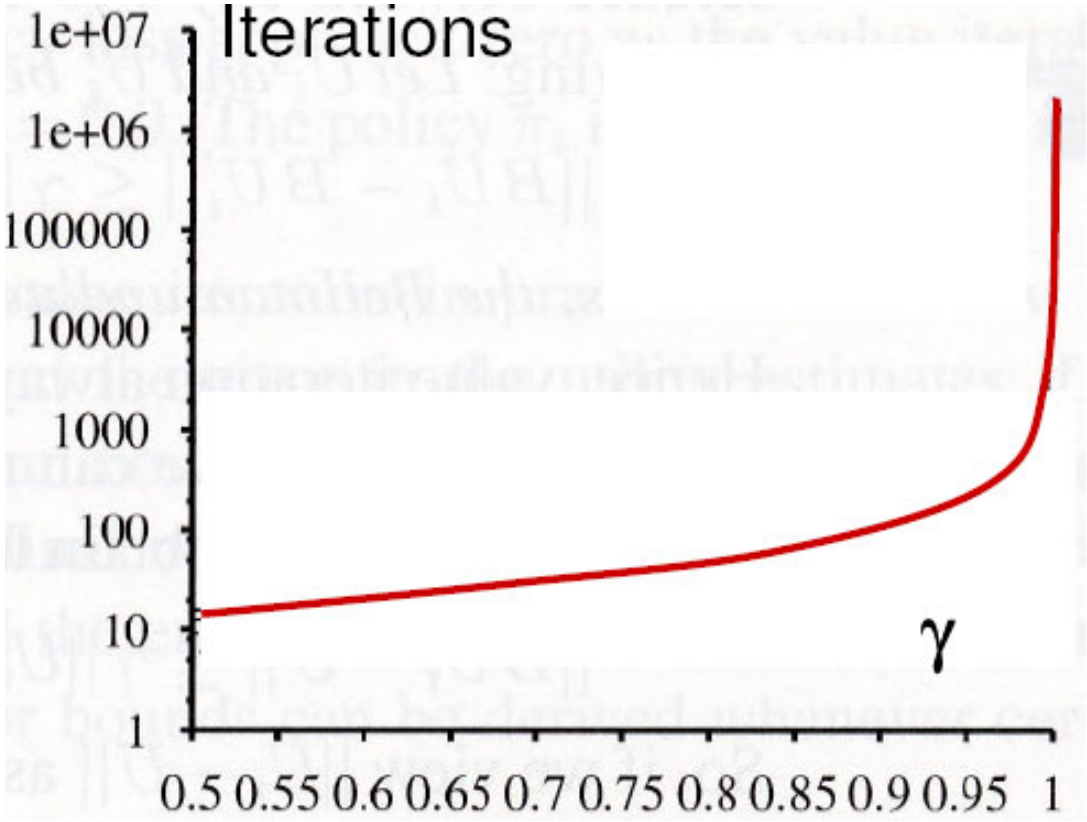
k	J^k(PU)	J^k(PF)	J^k(RU)	J^k(RF)
1				
2				
3				
4				
5				
6				

Maze Example





Key Convergence Results



- Exponentially fast convergence
- Slower convergence as γ increases

So far …

- Definition of discounted sum of rewards to measure utility
- Definition of Markov Decision Processes (MDP)
- Assumes observable states and uncertain action outcomes
- Optimal policy is the choice of actions that result in the maximum expected rewards in the future
- Bellman equation for general formulation of optimal policy in MDP
- Value iteration technique for computing the optimal policy
- Next: other approaches for optimal policy computation

Another Solution: Policy Iteration

- Start with a randomly chosen policy π_o
- Iterate until convergence:
  1. Compute $J_k(s)$ for every state *s* using π_k
  2. Update the policy by choosing the best action given the expected discounted rewards computed at step *k*:

$$\pi_{k+1}(S_i) = \arg\max_a \left(\sum_{s_j} P_{ij}^a J_k(S_j)\right)$$

The sequence of policies $\pi_o, \pi_1, \ldots, \pi_k, \ldots$ converges to π^*

Evaluating a Policy

Compute $J_k(s)$ for every state s using π_k

$$J_k(S_i) = r_i + \gamma \sum_{S_j} P_{ij}^{\pi_k(S_i)} J_k(S_j)$$

Linear set of equations can be solved in $O(|S|^3)$

May be too expensive for $|S|$ large, instead use simplified update rule:

$$J_k(S_i) \leftarrow r_i + \gamma \sum_{S_j} P_{ij}^{\pi_k(S_i)} J_{k-1}(S_j)$$

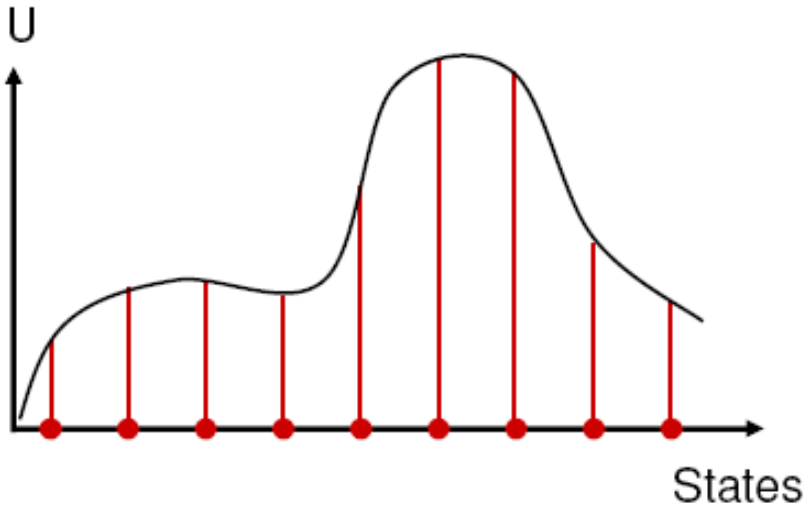
(*modified policy iteration*)

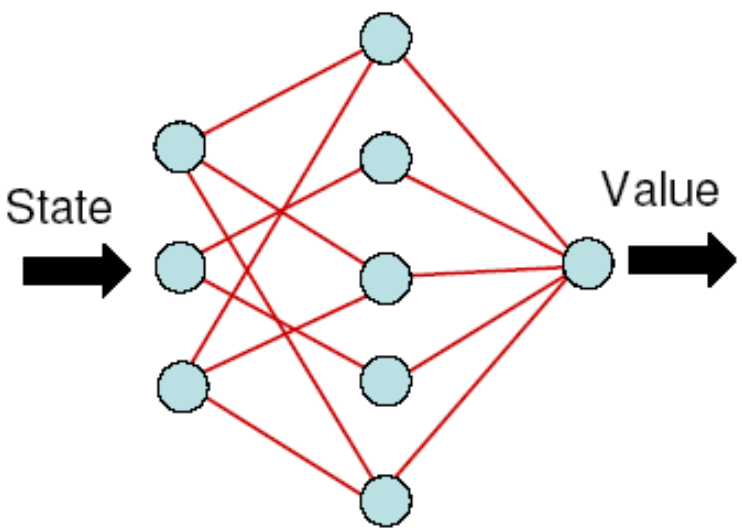
Limitations

- We need to represent the values (and policy) for every state in principle
- In real problems, the number of states may be very large
- Leads to intractably large tables (checker-like problem with N cells and M pieces $N(N\text{-}1)(N\text{-}2)..(N\text{-}M)$ states!)
- Need to find a compact way of representing the states
- Solutions:
  - Interpolation
  - Memory-based representations

State s	Value U
s_1	
s_2	
.....	
s_{100000}	
s_{100001}	

Function Approximation



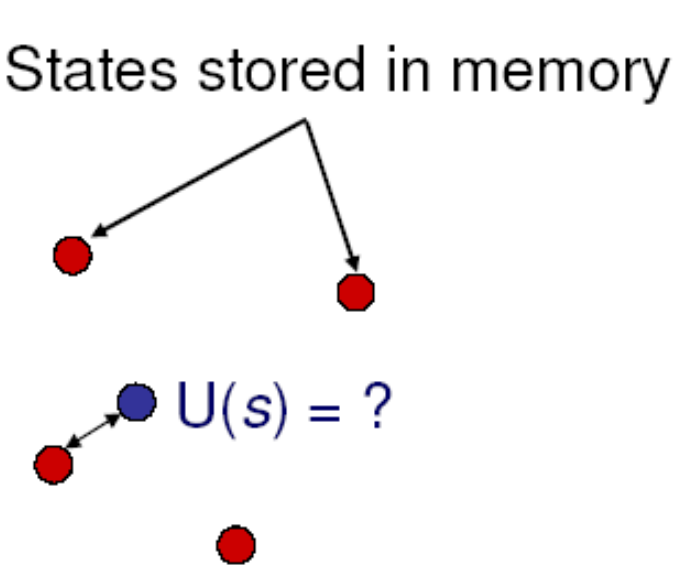


Polynomials/Splines approximation:
Represent U(*s*) by a polynomial function that can be represented by a small number of parameters.

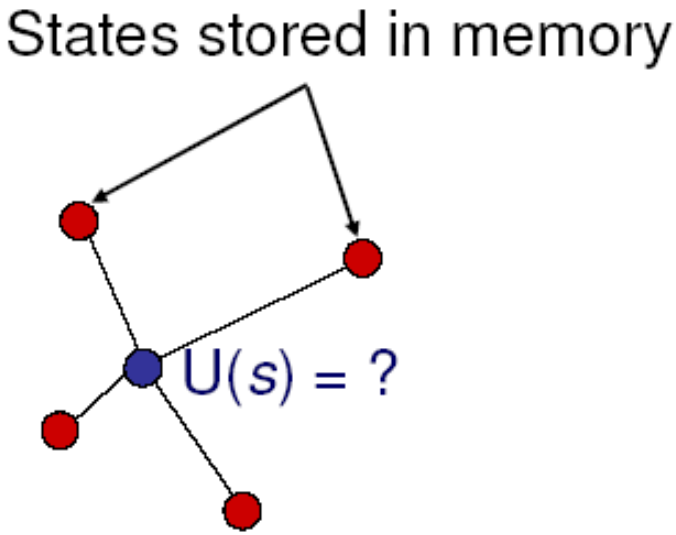
Economic models, control
Operations Research
Channel routing, Radio therapy

Neural Nets:
Represent U(*s*) implicitly by a neural net (function interpolation).
Elevator scheduling, Cell phones, Backgammon, etc.

Memory-Based Techniques



Replace U(*s*) by
U(closest neighbor to *s*)



Replace U(*s*) by weighted average of U(*K* closest neighbor to *s*)

Applications of MDPs

This extends the basic search algorithms to the case of probabilistic next states.

Many important problems are MDPs….

- … Robot path planning
- … Travel route planning
- … Elevator scheduling
- … Bank customer retention
- … Autonomous aircraft navigation
- … Manufacturing processes
- … Network switching & routing

Summary

- Definition of discounted sum of rewards to measure utility
- Definition of Markov Decision Processes (MDPs)
- Assumes observable states and uncertain action outcomes
- Optimal policy = choice of action that results in the maximum expected rewards in the future
- Bellman equation for general formulation of optimal policy in MDP
- *Value iteration* technique for computing the optimal policy
- *Policy iteration* technique for computing the optimal policy