# Markov Decision Processes and Reinforcement Learning

## 15-381 lecture notes, 13 April 2000

David Cohn <www.cs.cmu.edu/~cohn>

[based heavily on notes by Andrew W. Moore]

*Chapter 20 of Russell & Norvig*

After this lecture, you should:

- be able to describe the components of a Markov decision process (MDP)
- given a system, either describe how it can be coded as an MDP, or explain why it *can't* be coded as an MDP
- concisely define "optimal value function"
- describe three ways in which you might solve an MDP

# The value of being in state S

- Last lecture, we searched for optimal configurations
  - assumed we could easily (if expensively) compute *eval(s)*
- What is

  *eval(in-an-airplane-50-feet-above-the-runway-headed-north-at-253-knots)*?

  *eval(quit-school-and-join-my-friend's-silicon-valley-startup)*?

  *eval(three-twos-and-an-ace-against-Greg-playing-poker)*?

  *eval(set-out-on-three-hour-tour-in-Minnow)*?

- Evaluation depends on what happens next
  - similar to game tree search, but
    - not necessarily an "end" to process
    - not assume "world/adversary" acts optimally against you

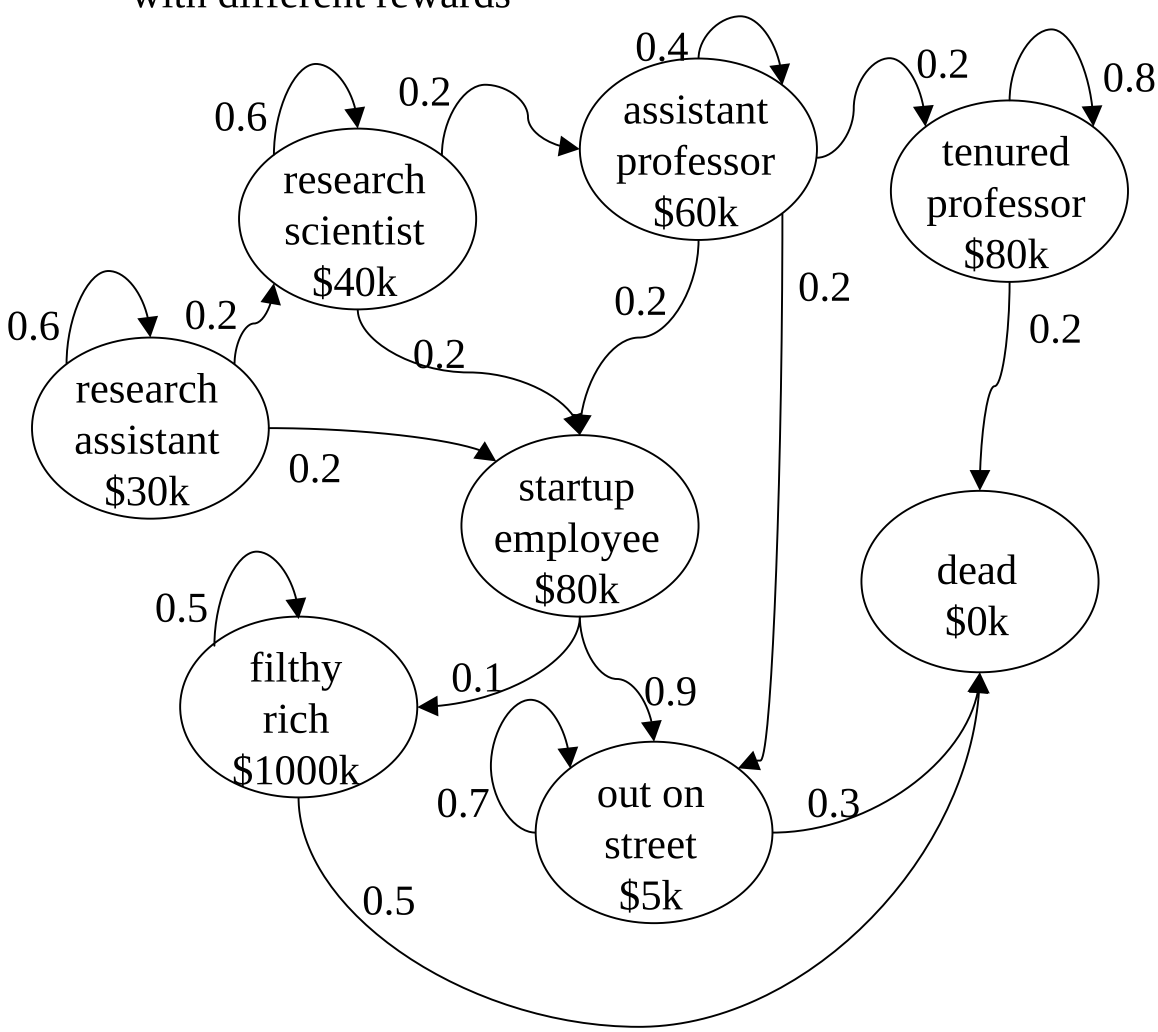
# Discounted rewards

- As a research assistant, get paid \$30k/year
  - how much is it worth to stay in this job all your life?
  - \$30k+\$30k+\$30k+\$30k+... = infinity!
- Discounting: a reward in the future is not worth as much as that same reward right now
  - justification:
    - inflation
    - impatience
    - chance that we'll get hit by an asteroid
  - example: \$10k next year is worth only 90% as much as \$10k right now
- Assuming payment *n* years in the future is worth $0.9^n$ of payment right now, what is **future discounted sum of rewards** for being a research assistant?
  - given discount factor of γ, compute

$$
\begin{aligned}
eval(ra) = {} & reward(now) \\
& + \gamma \cdot reward(now+1) \\
& + \gamma^2 \cdot reward(now+2) \\
& + \gamma^3 \cdot reward(now+3) \\
& + \ldots \text{ infinite sum}
\end{aligned}
$$

# In a more complex world

- Previous example assumes that you remain in same state all your life
  - but states change (probabilistically!), putting you in states with different rewards



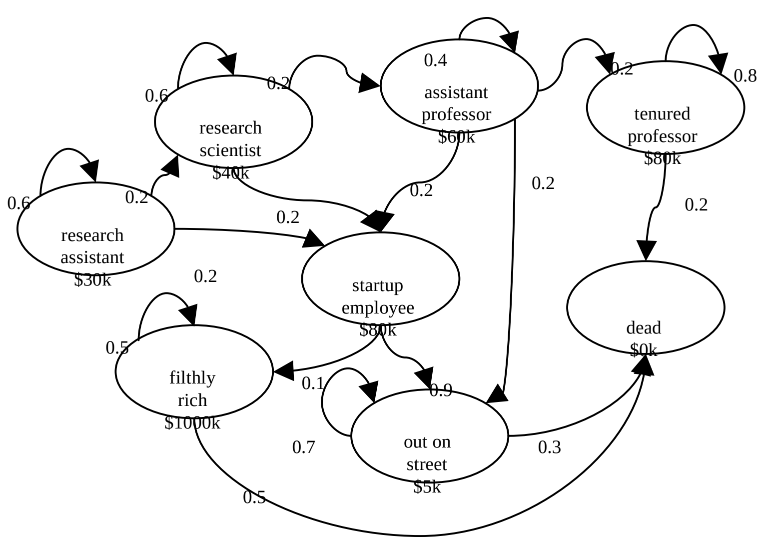

- ***Now*, what's the value of being a research assistant?**

# Markov Processes

- A Markov process has
  - a set of states $S$, e.g. $S = \{ra, rs, ap, tp, se, fr, os, d\}$
  - a transition probability matrix P, such that $p_{ij} = Prob(next=s_j|this=s_i)$

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & & & \\ \vdots & & \ddots & \\ p_{N1} & & & p_{NN} \end{pmatrix}$$

  - a reward $r$ associated with each state, e.g. $\{r_{ra}, r_{rs}, r_{ap}, r_{tp}, r_{se}, r_{fr}, r_{os}, r_d\}$
  - a discount factor $0<\gamma<1$
- At each time step (assume we're in state $s_i$):
  - get reward $r_i$
  - randomly move to state $s'$ according to $p_{s's}$



# Solving a Markov process

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

*$J^*(s_i) = r_i + \gamma \cdot$ (expected future rewards starting in next state)*

*$J^*(s_i) = r_i + \gamma \cdot \Sigma_j [p_{ji} \cdot J^*(s_j)]$*

- Write in vector notation:

$$J^* = \begin{pmatrix} J^*(s_1) \\ J^*(s_2) \\ \vdots \\ J^*(s_N) \end{pmatrix}, R = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{pmatrix}, P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & & & \\ \vdots & & \ddots & \\ p_{N1} & & & p_{NN} \end{pmatrix}$$

- In vector form, $J^*_t = R + \gamma \cdot PJ^*_{t+1}$
  - estimate value using dynamic programming!

# Dynamic programming

- Given reward $R$, transition probabilities $P$, and discount $\gamma$

$$J^* = \begin{pmatrix} J^*(s_1) \\ J^*(s_2) \\ \vdots \\ J^*(s_N) \end{pmatrix}, R = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{pmatrix}, P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & & & \\ \vdots & & \ddots & \\ p_{N1} & & & p_{NN} \end{pmatrix}$$

  what's our expected discounted sum of future rewards from state $s_i$
  - if we die immediately?
  - if we only live one time step?
  - if we live two time steps?
  - n time steps?
  - forever?
- Can we find a closed form solution for this value?
- What are the advantages/disadvantages of this approach?

# Value iteration - another way to solve a Markov System

- Bellman's equation: $J^* = R + \gamma \cdot PJ^*$
  - converges to a fixed point
  - why not simply try iterating until get an answer that's "good enough"?

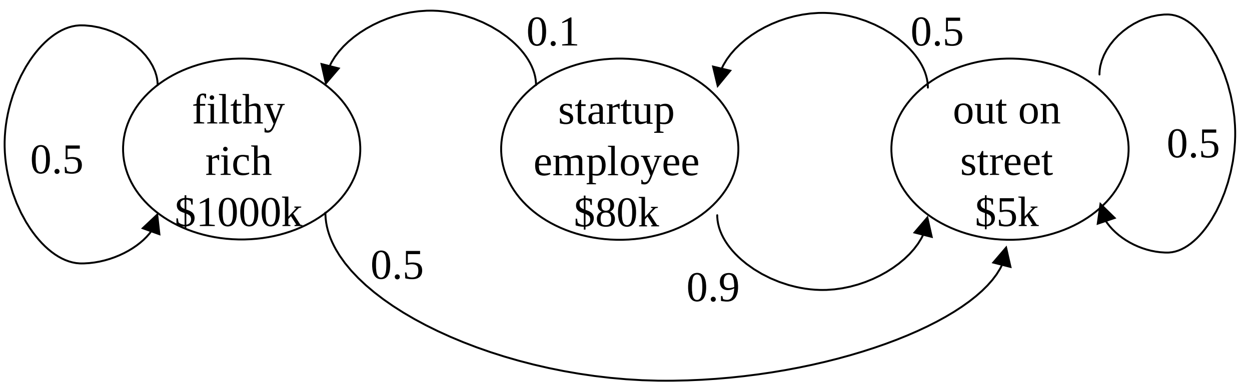
- Value iteration
  - set $J(s_i) = r_i$ for all states in system
    *(reward after zero time steps)*
  - pick a state $s_i$ at random
  - examine all $s_j$ such that $p_{ji}$ is nonzero
  - set $J(s_i) := r_i + \gamma \cdot \Sigma_j [p_{ji} \cdot J^*(s_j)]$
  - pick another $s_i$ and repeat

- Bellman equation guarantees these "backups" will converge to a fixed point where $J = J^*$
  - if all states are backed up infinitely often (ergodicity)

- How to pick $s_i$?
  - uniformly (same as dynamic programming)
  - randomly
  - by walking through Markov process
    - but must ensure ergodicity!

# Value iteration

- Okay - let’s try some value iteration!
- The idyllic world of a Silicon Valley entrepreneur



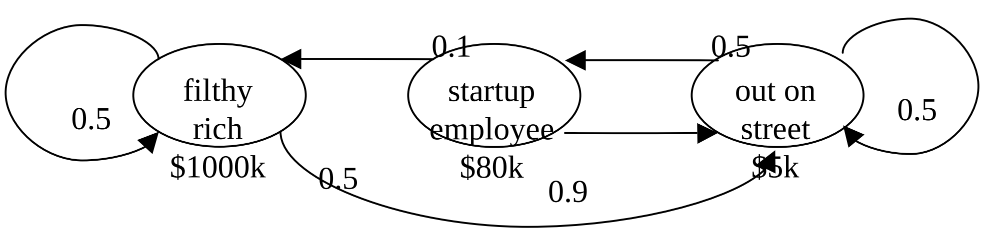

| k | Jk(rich) | Jk(startup) | Jk(street) |
|---|---|---|---|
| 0 | | | |
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |

# Markov decision processes

- But wait! You're not just a passive observer in this process - you actually get to take actions which affect your outcome!
  - actions affect probability of transitioning between states
  - $p_{jia}$ = probability of going from state $i$ to state $j$ given that you take action $a$
- Example:
  - three actions available: *slack-off*, *work-like-hell*, *buy-porsche*
  - not all actions available in all states

| **s=startup** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
|---|---|---|---|---|
| a=slack | 0.01 | 0 | 0.99 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| | | | | |
| **s=rich** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0.9 | 0 | 0.1 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| a=buy porsche | 0.1 | 0 | 0.1 | 0.8 |
| | | | | |
| **s=street** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0 | 0.6 | 0.4 | 0 |
| a=work | 0 | 0.6 | 0.4 | 0 |

- No longer makes sense to talk about "value of state"



# Markov decision processes (MDPs)

- A Markov decision process has
  - a set of states *S*
  - a set of actions *A*
  - a transition probability matrix *P*, such that
    $p_{ija} = Prob(next{=}s_j | this{=}s_i, action{=}a)$
  - a reward *r* associated with each state
  - a discount factor $0<\gamma<1$

| **s=startup** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
|---|---|---|---|---|
| a=slack | 0.01 | 0 | 0.99 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| | | | | |
| **s=rich** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0.9 | 0 | 0.1 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| a=buy porsche | 0.1 | 0 | 0.1 | 0.8 |
| | | | | |
| **s=street** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0 | 0.6 | 0.4 | 0 |
| a=work | 0 | 0.6 | 0.4 | 0 |

- ***Value*** of state depends on what action we take in that state
- A ***policy*** is a mapping from states to actions
  - example policy:

| | |
|---|---|
| **s=startup** | **work** |
| **s=rich** | **buy porsche** |
| **s=street** | **work** |

# Some important questions about MDPs

- How many policies are possible for our entrepreneur?
- How would we compute the value of a policy?
- How would we find the ***best*** policy?
  - review: what do we mean by "best"?

| **s=startup** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
|---|---|---|---|---|
| a=slack | 0.01 | 0 | 0.99 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| | | | | |
| **s=rich** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0.9 | 0 | 0.1 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| a=buy porsche | 0.1 | 0 | 0.1 | 0.8 |
| | | | | |
| **s=street** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0 | 0.6 | 0.4 | 0 |
| a=work | 0 | 0.6 | 0.4 | 0 |

- Factoid: for every MDP, there exists an ***optimal policy***
  - an optimal policy is one such that, for every possible start state, thre is no better option than to follow the policy
  - question: can there be more than one optimal policy?

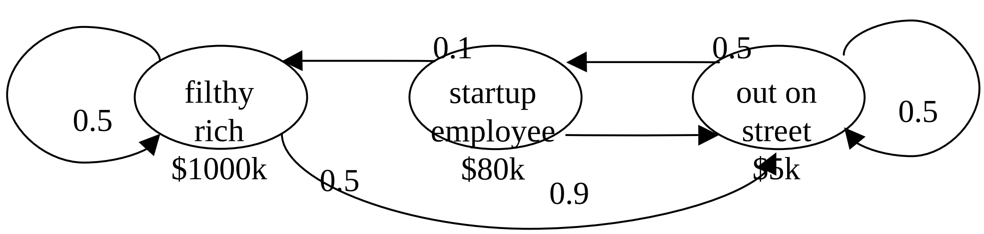
- So, how to *find* an optimal policy?
- Idea #1: Evaluate all possible policies; pick the best
  - how many policies did we say there were?
  - how long will it take to evaluate each one?

# The optimal value function

- Define $J^*(s_i)$ to be expected discounted sum of future rewards starting from $s_i$ - *assuming that we use the optimal policy*

- *Start with finite horizon - what's the best we can do if we're going to live only 1 time step?*
  - *if we're going to live only two time steps?*

| **s=startup** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
|---|---|---|---|---|
| a=slack | 0.01 | 0 | 0.99 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| | | | | |
| **s=rich** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0.9 | 0 | 0.1 | 0 |
| a=work | 0.1 | 0 | 0.9 | 0 |
| a=buy porsche | 0.1 | 0 | 0.1 | 0.8 |
| | | | | |
| **s=street** | **p(rich)** | **p(startup)** | **p(street)** | **p(dead)** |
| a=slack | 0 | 0.6 | 0.4 | 0 |
| a=work | 0 | 0.6 | 0.4 | 0 |



# Computing the optimal value function

- Define $J^k(s_i)$ = maximum possible expected future sum of rewards I can get by starting in state $s_i$ and taking $k$ steps
- Note that $J^0(s_i) = r_i$

- *Let's compute $J^k(s_i)$ for our entrepreneur!*

| k | Jk(rich) | Jk(startup) | Jk(street) | Jk(dead) |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |

- Bellman's equation for MDPs

$$J^{n+1}(s_i) = \max_k \left[ r_i + \gamma \cdot \sum_j p_{ji}^k J^n(s_j) \right]$$

- Can we find a closed form solution for this?
  - why or why not?

# Computing the optimal value function

- "Max" ruins our chance of finding a closed form solution - so what do we do?
  - perform value iteration - just like with (non-decision) Markov processes

$$J^{n+1}(s_i) = \max_{\mathrm{k}} \left[ r_i + \gamma \cdot \sum_j p_{ji}^k J^n(s_j) \right]$$

- Bellman's equation converges to a fixed point, so
  - compute $J^1(s_i)$ for all $i$
  - compute $J^2(s_i)$ for all $i$
  - compute $J^3(s_i)$ for all $i$
  - *...until converged*
- *Here, "convergence" means* $|J^{n+1}(s_i) - J^n(s_i)| < \varepsilon$

# *Un*-discounted MDPs

- Discounting is a means of keeping rewards from going to infinity - but sometimes it's not necessary
  - chess: ***absorbing state*** at end of MDP gives zero reward until end of game, then +/-1 reward
  - landing an airplane: penalty of -1 for every time step (fuel used), reward of +1..+5 for landing, penalty of -1000000 for crashing
- Can use undiscounted form of Bellman equation

$$J^{n+1}(s_i) = \max_{\mathrm{k}}\left[ r_i + \sum_j p_{ji}^k J^n(s_j) \right]$$

# Policy iteration - another way to learn optimal policies for MDPs

- Denote a policy as $\Pi$, where $\Pi(s_i)$ = action selected in state $s_i$
- Denote $\Pi^0$ as any random policy
- Denote $J^0(s_i)$ = long term reward starting at $s_i$ and following policy $\Pi^0$
  - since policy is fixed, can compute in closed form, or quickly using updated
- Define a new, *greedy* policy $\Pi^1$ in terms of $J^0(s_i)$:

$$\Pi^1(s_i) = \arg\max_a \left[ r_i + \gamma \cdot \sum_j p_{ji}^a J^0(s_j) \right]$$

- Then define a new $J^1(s_i)$ in terms of $\Pi^1(s_i)$
- Then define a new $\Pi^2(s_i)$ in terms of $J^1(s_i)$...
- ...until $\Pi^{t+1} = \Pi^t$. This policy will be optimal

# Policy vs. value iteration - which is better?

- Depends... (doesn't everything?)
  - lots of actions: choose policy iteration
  - already have a decent policy: choose policy iteration
  - few actions, acyclic: choose value iteration
- Can try for the best of both worlds: *modified policy iteration*
  - do a few steps of value iteration to get "approximate" values
  - update policy greedily based on that

- Yet another approach: linear programming

# Applications of MDPs

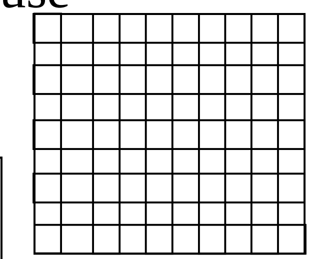
- Huge numbers of applications
  - robot control, path planning
  - travel planning
  - elevator scheduling
  - stochastic job shop scheduling
  - bank customer retention
  - aircraft control
  - network switching and routing
  - game playing
  - taking over the known universe

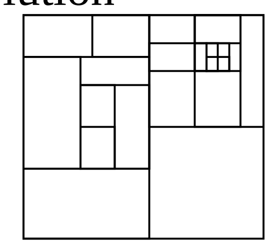
- Anything where we can model system as
  - states, actions, rewards

- Things that are tricky to model with MDPs
  - time-varying rewards, probabilities
  - things with very many states or actions
  - things with hidden state (POMDPs)

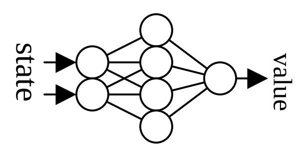
# One big problem: we can’t represent big problems

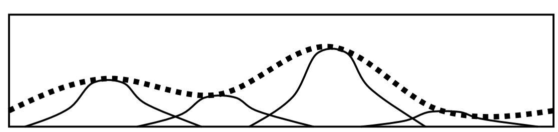
- Many (most?) interesting problems have continuous state spaces, infinite numbers of states
- How to deal? Instead of a lookup table, use
  - discretization
    - fixed vs. variable resolution





  - function approximation
    - neural networks
    - splines
    - basis functions



  - memory-based methods
    - nearest neighbor, locally-weighted regression...

- Disadvantage: all convergence guarantees disappear
- Advantage: things *usually* work (with a bit of help)

# For more information:

- Survey on MDPs and reinforcement learning in general, by Kaelbling, Littman & Moore at *http://www.jair.org/abstracts/kaelbling96a.html*

- A few RL repositories, with pointers to people, papers, sample domains, applicatio  ns, workshops, data...
- *http://web.cps.msu.edu/rlr/*
  *http://www-anw.cs.umass.edu/~mharmon/rltutorial*