Towards Learning in Probabilistic Action Selection: Markov Systems and Markov Decision Processes

Manuela Veloso

Carnegie Mellon University
Computer Science Department

Planning - Fall 2001

Remember the examples on the board.

Different Aspects of "Machine Learning"

- Supervised learning
  - Classification - concept learning
  - Learning from labeled data
  - Function approximation
- Unsupervised learning
  - Data is not labeled
  - Data needs to be *grouped, clustered*
  - We need distance metric
- Control and action model learning
  - Learning to select actions efficiently
  - Feedback: goal achievement, failure, *reward*
  - Search control learning, reinforcement learning

Search Control Learning

- Improve *search* efficiency, *plan quality*
- Learn *heuristics*

```
(if (and (goal (enjoy weather))
         (goal (save energy))
         (state (weather fair)))
(then (select action RIDE-BIKE)))
```

Learning Opportunities in Planning

- Learning to improve planning efficiency
- Learning the domain model
- Learning to improve plan quality
- Learning a universal plan

Which action model,
which planning algorithm,
which heuristic control
is the most efficient for a given task?

Reinforcement Learning

- A variety of algorithms to address:
  - learning the *model*
  - converging to the *optimal* plan.

Discounted Rewards

- "Reward" today versus future (promised) reward
- \$100K + \$100K + \$100K + . . .
  INFINITE!
- Future rewards not worth as much as current.
- Assume reality . . . : discount factor , say γ.
- \$100K + γ \$100K + γ^2 \$100K + . . .
  CONVERGES!

Markov Systems with Rewards

- Finite set of n states - vector n - s_i
- Probabilistic state matrix, P - $n \times n$ - p_{ij}
- "Goal achievement" - Reward for each state, vector n -r_i
- Discount factor - γ
- Process:
  - Start at state s_i
  - Receive immediate reward r_i
  - Move randomly to a new state according to the probability transition matrix
  - Future rewards (of next state) are discounted by γ

Solving a Markov Systems with Rewards

- $V^*(s_i)-$ expected discounted sum of future rewards starting in state s_i
-

$$V^*(s_i) = r_i + \gamma[p_{i1}V^*(s_1) + p_{i2}V^*(s_2) + \ldots p_{in}V^*(s_n)]$$

Value Iteration to Solve a Markov Systems with Rewards

- $V^1(s_i)-$ expected discounted sum of future rewards starting in state s_i *for one step*.
- $V^2(s_i)-$ expected discounted sum of future rewards starting in state s_i *for two steps*.
- ...
- $V^k(s_i)-$ expected discounted sum of future rewards starting in state s_i *for k steps*.
- As $k \rightarrow \infty V^k(s_i) \rightarrow V^*(s_i)$
- Stop when difference of $k+1$ and k values is smaller than some ϵ.

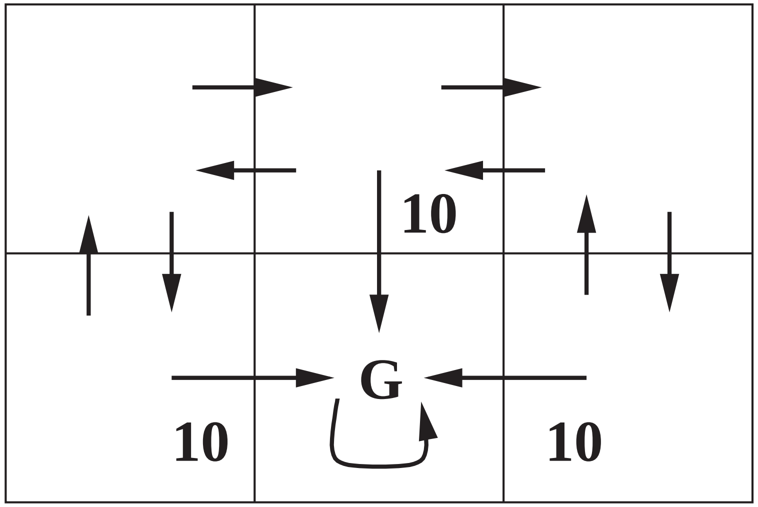
Markov Decision Processes

- Finite set of states, $s_1, \ldots, s_n$
- Finite set of actions, $a_1, \ldots, a_m$
- Probabilistic state,action transitions:

$p_{ij}^k =$ prob (next = s_j | current = s_i and take action a_k)

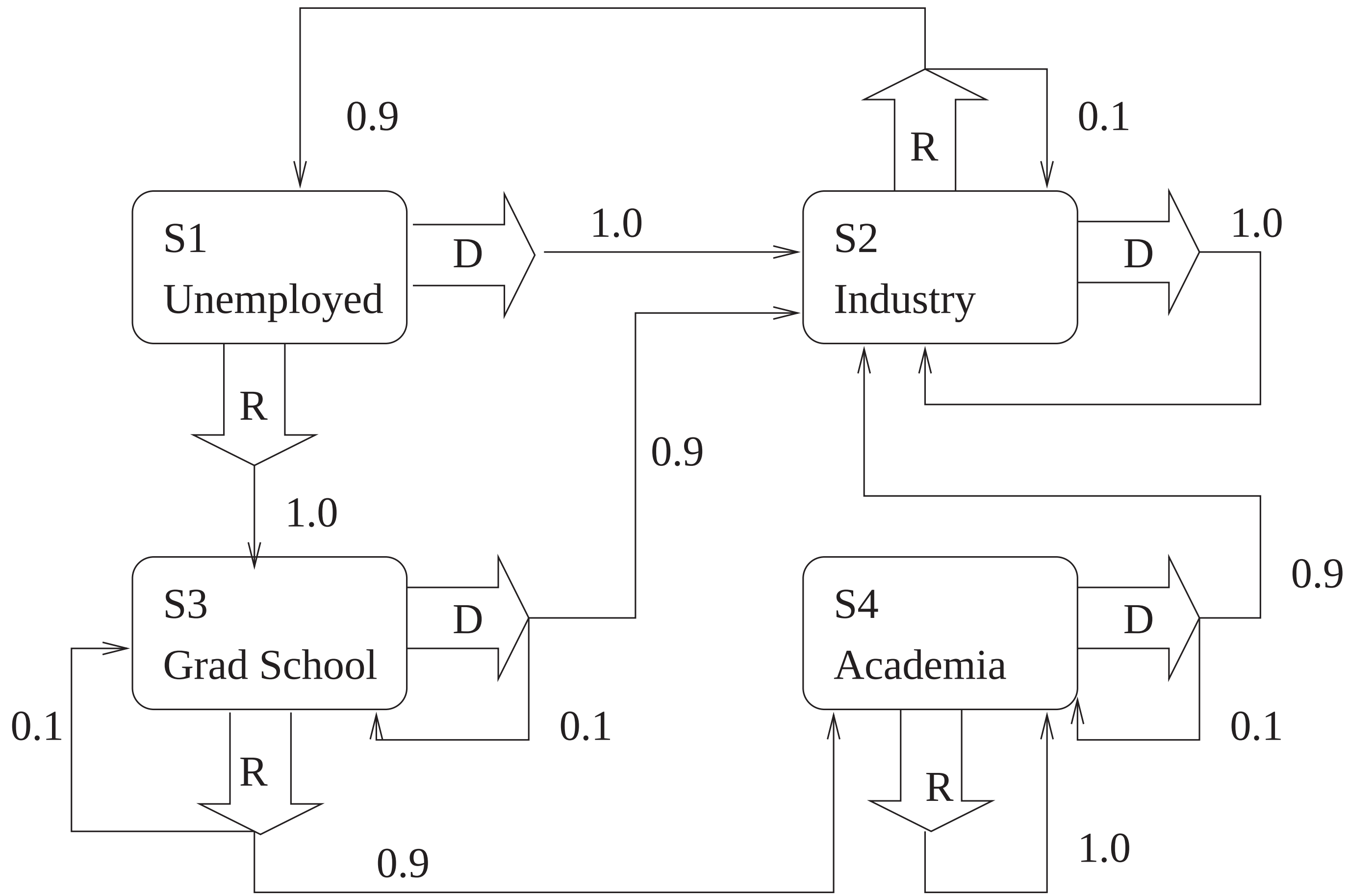
- Reward for each state, $r_1, \ldots, r_n$
- Process:
  - Start in state s_i
  - Receive immediate reward r_i
  - Choose action $a_k \in A$
  - Change to state s_j with probability p_{ij}^k.
  - Discount future rewards

Deterministic Example



(Reward on unlabelled transitions is zero.)

Nondeterministic Example



Solving an MDP

- A policy is a mapping from states to actions.
- Optimal policy - for every state, there is no other action that gets a higher sum of discounted future rewards.
- For every MDP there exists an optimal policy.
- Solving an MDP is finding an optimal policy.
- A specific policy converts an MDP into a plain Markov system with rewards.

Policy Iteration

- Start with some policy $\pi_0(s_i)$.
- Such policy transforms the MDP into a plain Markov system with rewards.
- Compute the values of the states according to current policy.
- Update policy:

$$\pi_1(s_i) = \text{argmax}_a \{r_i + \gamma \sum_j p_{ij}^a V^{\pi_0}(s_j)\}$$

- Keep computing
- Stop when $\pi_{k+1} = \pi_k$.

Value Iteration

- $V^*(s_i)$ = expected discounted future rewards, if we start from s_i and we follow the optimal policy.
- Compute V^* with value iteration:
  - $V^k(s_i)$ = maximum possible future sum of rewards starting from state s_i for k steps.
- Bellman's Equation:

$$V^{n+1}(s_i) = \max_k \{r_i + \gamma \sum_{j=1}^{N} p_{ij}^k V^n(s_j)\}$$

- Dynamic programming