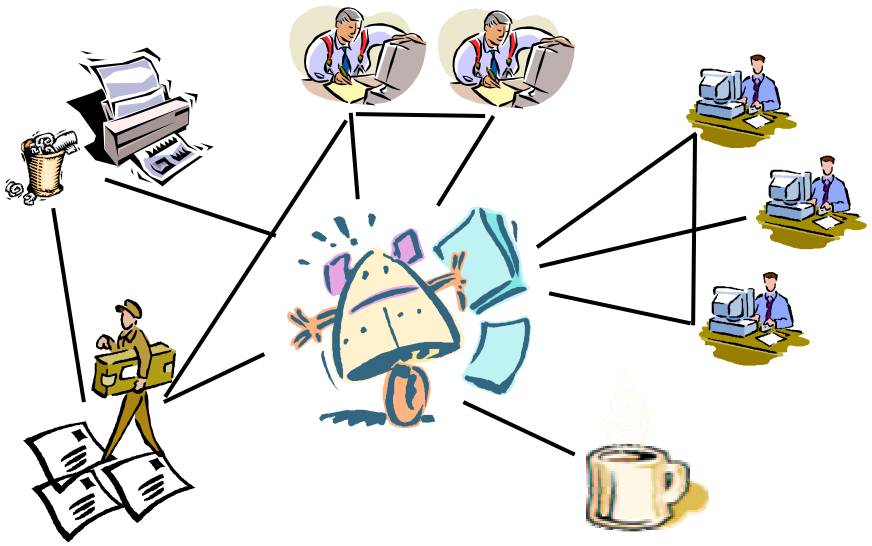
Logical Representations and Computational Methods for Markov Decision Processes

Craig Boutilier
Department of Computer Science
University of Toronto

Planning in Artificial Intelligence

- Planning has a long history in AI
  - strong interaction with logic-based knowledge representation and reasoning schemes
- Basic planning problem:
  - Given: start state, goal conditions, actions
  - Find: sequence of actions leading from start to goal
  - Typically: states correspond to possible worlds; actions and goals specified using a logical formalism (e.g., STRIPS, situation calculus, temporal logic, etc.)
- Specialized algorithms, planning as theorem proving, etc. often exploit logical structure of problem is various ways to solve effectively

A Planning Problem



Difficulties for the Classical Model

- Uncertainty
  - in action effects
  - in knowledge of system state
  - a *"sequence of actions that guarantees goal achievement"* often does not exist
- Multiple, competing objectives
- Ongoing processes
  - lack of well-defined termination criteria

Some Specific Difficulties

- **Maintenance goals:** *"keep lab tidy"*
  - goal is never achieved once and for all
  - can't be treated as a safety constraint
- **Preempted/Multiple goals:** *"coffee vs. mail"*
  - must address tradeoffs: priorities, risk, etc.
- **Anticipation of Exogenous Events**
  - e.g., wait in the mailroom at 10:00 AM
  - on-going processes *driven* by exogenous events
- Similar concerns: logistics, process planning, medical decision making, etc.

Markov Decision Processes

- Classical planning models:
  - logical rep'n s of deterministic transition systems
  - goal-based objectives
  - plans as sequences
- Markov decision processes generalize this view
  - controllable, stochastic transition system
  - general objective functions (rewards) that allow tradeoffs with transition probabilities to be made
  - more general solution concepts (policies)

Logical Representations of MDPs

- MDPs provide a nice conceptual model
- Classical representations and solution methods tend to rely on state-space enumeration
  - combinatorial explosion if state given by set of possible worlds/logical interpretations/variable assts
  - Bellman's *curse of dimensionality*
- Recent work has looked at extending AI-style representational and computational methods to MDPs
  - we'll look at some of these (with a special emphasis on "logical" methods)

Course Overview

- Lecture 1
  - motivation
  - introduction to MDPs: classical model and algorithms
- Lecture 2
  - AI/planning-style representations
  - probabilistic STRIPs; dynamic Bayesian networks; decision trees and BDDs; situation calculus
  - some simple ways to exploit logical structure: abstraction and decomposition

Course Overview (con't)

- Lecture 3
  - decision-theoretic regression
  - propositional view as variable elimination
  - exploiting decision tree/BDD structure
  - approximation
  - first-order DTR with situation calculus
- Lecture 4
  - linear function approximation
  - exploiting logical structure of basis functions
  - discovering basis functions

Course Overview (con't)

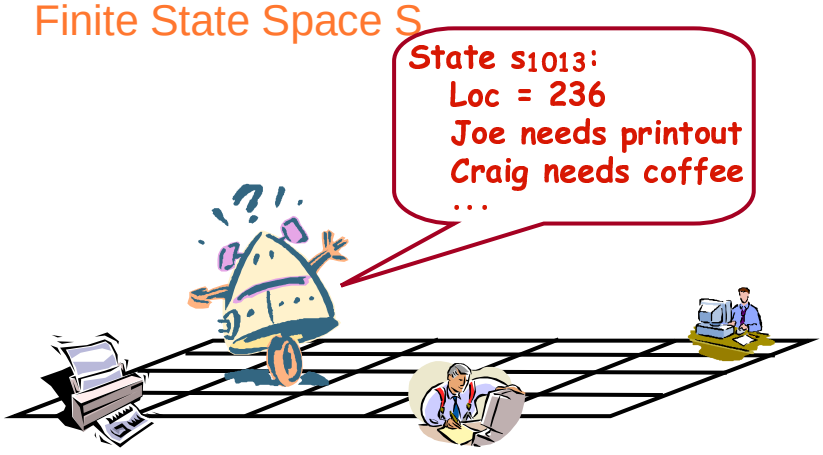
- Lecture 5
  - temporal logic for specifying non-Markovian dynamics
  - model minimization
  - wrap up; further topics

Markov Decision Processes

- An MDP has four components, S, A, R, Pr:
  - (finite) state set S (|S| = n)
  - (finite) action set A (|A| = m)
  - transition function Pr(s,a,t)
    - each Pr(s,a,-) is a distribution over S
    - represented by set of n x n stochastic matrices
  - bounded, real-valued reward function R(s)
    - represented by an n-vector
    - can be generalized to include action costs: R(s,a)
    - can be stochastic (but replacable by expectation)
- Model easily generalizable to countable or continuous state and action spaces

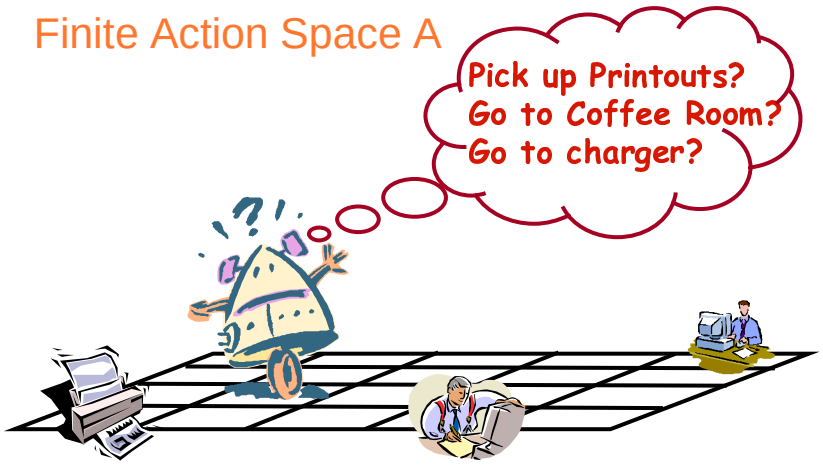
System Dynamics

Finite State Space S



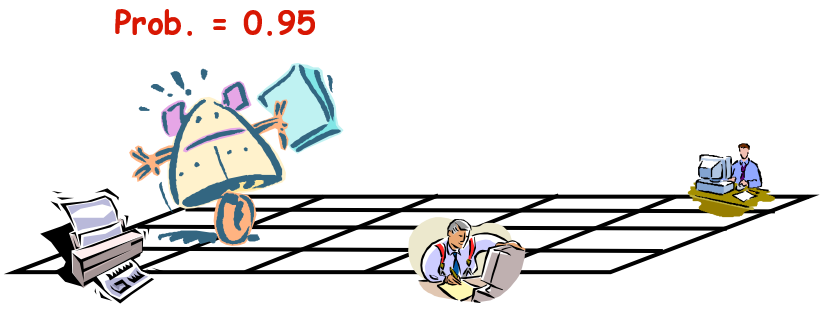
System Dynamics

Finite Action Space A



System Dynamics

Transition Probabilities: $Pr(s_i, a, s_j)$

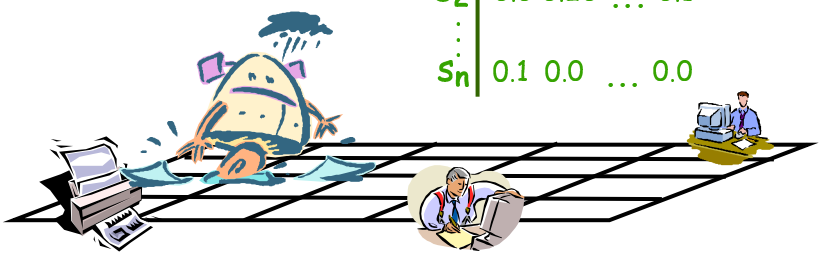


System Dynamics

Transition Probabilities: $Pr(s_i, a, s_k)$

Prob. = 0.05

	s_1	s_2	...	s_n
s_1	0.9	0.05	...	0.0
s_2	0.0	0.20	...	0.1
⋮				
s_n	0.1	0.0	...	0.0

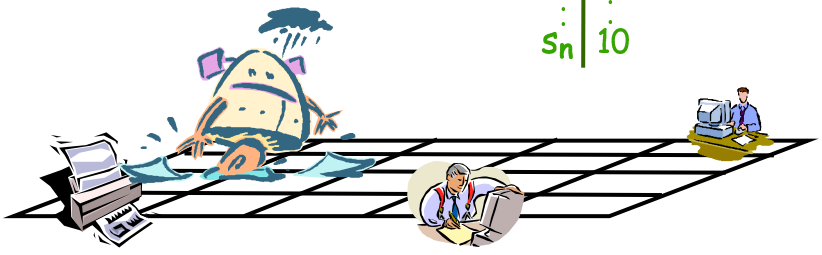


Reward Process

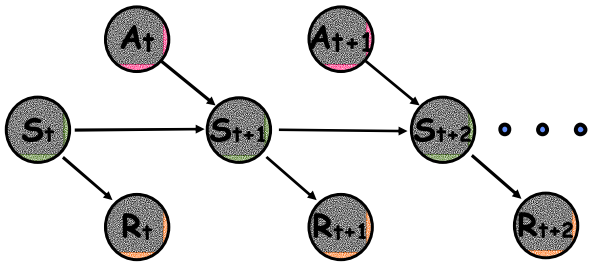
Reward Function: $R(s_i)$
- action costs possible

Reward = -10

	R
s_1	12
s_2	0.5
⋮	⋮
s_n	10



Graphical View of MDP



Assumptions

- Markovian dynamics (history independence)
  - $Pr(S^{t+1}|A^t,S^t,A^{t-1},S^{t-1},..., S^0) = Pr(S^{t+1}|A^t,S^t)$
- Markovian reward process
  - $Pr(R^t|A^t,S^t,A^{t-1},S^{t-1},..., S^0) = Pr(R^t|A^t,S^t)$
- Stationary dynamics and reward
  - $Pr(S^{t+1}|A^t,S^t) = Pr(S^{t'+1}|A^{t'},S^{t'})$ for all t, t'
- **Full observability**
  - though we can't predict what state we will reach when we execute an action, once it is realized, we know what it is

Policies

- Nonstationary policy
  - π:S x T → A
  - π(s,t) is action to do at state s with t-stages-to-go
- Stationary policy
  - π:S → A
  - π(s) is action to do at state s (regardless of time)
  - analogous to reactive or universal plan
- These assume or have these properties:
  - full observability
  - history-independence
  - deterministic action choice

Value of a Policy

- How good is a policy π? How do we measure "accumulated" reward?
- **Value function** V: S →$\mathbb{R}$ associates value with each state (sometimes S x T)
- V_π(s) denotes value of policy at state s
  - how good is it to be at state s? depends on immediate reward, but also what you achieve subsequently
  - expected accumulated reward over horizon of interest
  - note V_π(s) ≠ R(s); it measures *utility*

Value of a Policy (con't)

- Common formulations of value:
  - Finite horizon n: total expected reward given π
  - Infinite horizon discounted: discounting keeps total bounded
  - Infinite horizon, average reward per time step

Finite Horizon Problems

- Utility (value) depends on stage-to-go
  - hence so should policy: nonstationary π(s,k)
- $V_\pi^k(s)$ is k-stage-to-go value function for π

$$V_\pi^k(s) = E\left[\sum_{t=0}^{k} R^t \mid \pi, s\right]$$

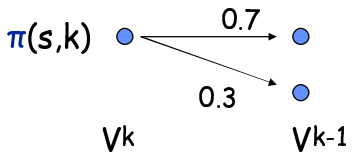
- Here R^t is a random variable denoting reward received at stage t

Successive Approximation

- Successive approximation algorithm used to compute $V_\pi^k(s)$ by dynamic programming

(a) $V_\pi^0(s) = R(s), \quad \forall s$

(b) $V_\pi^k(s) = R(s) + \sum_{s'} \Pr(s, \pi(s,k), s') \cdot V_\pi^{k-1}(s')$



Successive Approximation

- Let $P_{\pi,k}$ be matrix constructed from rows of action chosen by policy
- In matrix form:

  $V^k = R + P_{\pi,k} V^{k-1}$

- Notes:
  - π requires T n-vectors for policy representation
  - V_π^k requires an n-vector for representation
  - Markov property is critical in this formulation since value at s is defined independent of how s was reached

Value Iteration (Bellman 1957)

- Markov property allows exploitation of DP principle for optimal policy construction
  - no need to enumerate $|A|^{Tn}$ possible policies
- Value Iteration

Bellman backup

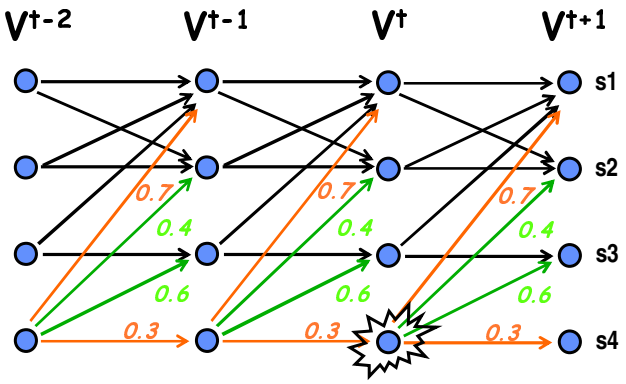
$$V^0(s) = R(s), \quad \forall s$$

$$V^k(s) = R(s) + \max_a \sum_{s'} \Pr(s,a,s') \cdot V^{k-1}(s')$$

$$\pi^*(s,k) = \arg\max_a \sum_{s'} \Pr(s,a,s') \cdot V^{k-1}(s')$$

V^k is optimal k-stage-to-go value function

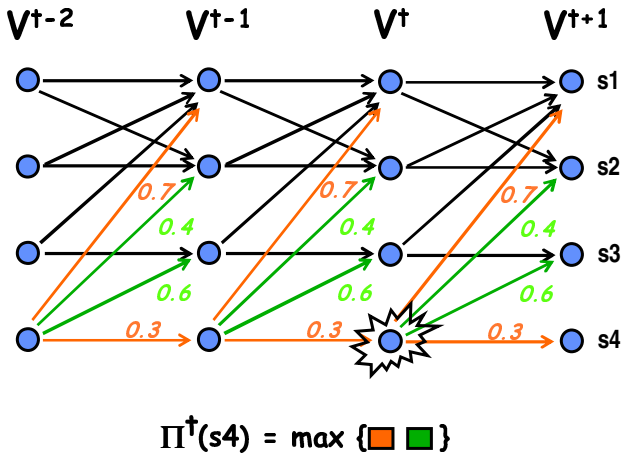
Value Iteration



$V^t(s4) = R(s4) + \max \{ 0.7\ V^{t+1}(s1) + 0.3\ V^{t+1}(s4)$

$0.4\ V^{t+1}(s2) + 0.6\ V^{t+1}(s3) \}$

Value Iteration



Value Iteration

- Note how DP is used
  - optimal soln to k-1 stage problem can be used without modification as part of optimal soln to k-stage problem
- Because of finite horizon, policy nonstationary
- In practice, Bellman backup computed using:

$$Q^k(a,s) = R(s) + \sum_{s'} \Pr(s,a,s') \cdot V^{k-1}(s'), \quad \forall a$$

$$V^k(s) = \max_a Q^k(a,s)$$

Complexity

- T iterations
- At each iteration |A| computations of n x n matrix times n-vector: O(|A|n^3)
- Total O(T|A|n^3)
- Can exploit sparsity of matrix: O(T|A|n^2)

Summary

- Resulting policy is optimal

$$V^k_{\pi^*}(s) \geq V^k_{\pi}(s), \quad \forall \pi, s, k$$

  - convince yourself of this; convince that nonMarkovian, randomized policies not necessary
- Note: optimal value function is unique, but optimal policy is not

Discounted Infinite Horizon MDPs

- Total reward problematic (usually)
  - many or all policies have infinite expected reward
  - some MDPs (e.g., zero-cost absorbing states) OK
- "Trick": introduce discount factor $0 \leq \beta < 1$
  - future rewards discounted by β per time step

$$V_\pi^k(s) = E\left[\sum_{t=0}^{\infty} \beta^t R^t \mid \pi, s\right]$$

- Note: $$V_\pi(s) \leq E\left[\sum_{t=0}^{\infty} \beta^t R^{\max}\right] = \frac{1}{1-\beta} R^{\max}$$
- Motivation: economic? failure prob? convenience?

Some Notes

- Optimal policy maximizes value at each state
- Optimal policies guaranteed to exist (Howard60)
- Can restrict attention to stationary policies
  - why change action at state s at new time t?
- We define $V^*(s) = V_\pi(s)$ for some optimal π

Value Equations (Howard 1960)

- Value equation for fixed policy value

$$V_\pi(s) = R(s) + \beta \sum_{s'} \mathbf{Pr}(s, \pi(s), s') \cdot V_\pi(s')$$

- Bellman equation for optimal value function

$$V^*(s) = R(s) + \beta \max_a \sum_{s'} \mathbf{Pr}(s, a, s') \cdot V^*(s')$$

Backup Operators

- We can think of the fixed policy equation and the Bellman equation as operators in a vector space
  - e.g., $L^a(V) = V' = R + \beta P^a V$
  - V_π is unique fixed point of policy backup operator L_π
  - V^* is unique fixed point of Bellman backup L^*
- We can compute V_π easily: **policy evaluation**
  - simple linear system with n variables, n constraints
  - solve $V = R + \beta PV$
- Cannot do this for optimal policy
  - max operator makes things nonlinear

Value Iteration

- Can compute optimal policy using value iteration, just like FH problems (just include discount term)

$$V^k(s) = R(s) + \beta \max_a \sum_{s'} \Pr(s, a, s') \cdot V^{k-1}(s')$$

  - no need to store argmax at each stage (stationary)

Convergence

- L(V) is a contraction mapping in R^n
  - $|| LV - LV' || \leq \beta || V - V' ||$
- When to stop value iteration? when $||V^k - V^{k-1}|| \leq \varepsilon$
  - $||V^{k+1} - V^k|| \leq \beta ||V^k - V^{k-1}||$
  - this ensures $||V^k - V^*|| \leq \varepsilon\beta / 1-\beta$
- Convergence is assured
  - any guess V: $|| V^* - L^*V || = ||L^*V^* - L^*V || \leq \beta || V^* - V ||$
  - so fixed point theorems ensure convergence

How to Act

- Given V* (or approximation), use *greedy* policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} \Pr(s, a, s') \cdot V^*(s')$$

  - if V within ε of V*, then V(π) within 2ε of V*
- There exists an ε s.t. optimal policy is returned
  - even if value estimate is off, greedy policy is optimal
  - proving you are optimal can be difficult (methods like *action elimination* can be used)

Policy Iteration

- Given fixed policy, can compute its value exactly:

$$V_\pi(s) = R(s) + \beta \sum_{s'} \Pr(s, \pi(s), s') \cdot V_\pi(s')$$

- Policy iteration exploits this

1. Choose a random policy π
2. Loop:
   (a) Evaluate V_π
   (b) For each s in S, set $\pi'(s) = \arg\max_a \sum_{s'} \Pr(s, a, s') \cdot V_\pi(s')$
   (c) Replace π with π′
Until no improving action possible at any state

Policy Iteration Notes

- Convergence assured (Howard)
  - intuitively: no local maxima in value space, and each policy must improve value; since finite number of policies, will converge to optimal policy
- Very flexible algorithm
  - need only improve policy at one state (not each state)
- Gives exact value of optimal policy
- Generally converges much faster than VI
  - each iteration more complex, but fewer iterations
  - quadratic rather than linear rate of convergence

Modified Policy Iteration

- MPI a flexible alternative to VI and PI
- Run PI, but don't solve linear system to evaluate policy; instead do several iterations of successive approximation to evaluate policy
- You can run SA until near convergence
  - but in practice, you often only need a few backups to get estimate of V(π) to allow improvement in π
  - quite efficient in practice
  - choosing number of SA steps a practical issue

Asynchronous Value Iteration

- Needn't do full backups of VF when running VI
- **Gauss-Siedel**: Start with V^k .Once you compute $V^{k+1}(s)$, you replace $V^k(s)$ before proceeding to the next state (assume some ordering of states)
  - tends to converge much more quickly
  - note: V^k no longer k-stage-to-go VF
- AVI: set some V^0; Choose random state s and do a Bellman backup at that state alone to produce V^1; Choose random state s…
  - if each state backed up frequently enough, convergence assured
  - useful for online algorithms (reinforcement learning)

Some Remarks on Search Trees

- Analogy of Value Iteration to decision trees
  - decision tree (expectimax search) is really value iteration with computation focussed on reachable states
- Real-time Dynamic Programming (RTDP)
  - simply real-time search applied to MDPs
  - can exploit heuristic estimates of value function
  - can bound search depth using discount factor
  - can cache/learn values
  - can use pruning techniques

References

- M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley, 1994.
- D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, 1987.
- R. Bellman, *Dynamic Programming*, Princeton, 1957.
- R. Howard, *Dynamic Programming and Markov Processes*, MIT Press, 1960.
- C. Boutilier, T. Dean, S. Hanks, Decision Theoretic Planning: Structural Assumptions and Computational Leverage, *Journal of Artif. Intelligence Research* 11:1-94, 1999.
- A. Barto, S. Bradke, S. Singh, Learning to Act using Real-Time Dynamic Programming, *Artif. Intelligence* 72(1-2):81-138, 1995.