Lecture 10 - Planning under Uncertainty (II)

Jesse Hoey
School of Computer Science
University of Waterloo

July 4, 2022

Readings: Poole & Mackworth (2nd ed.)Chapter 9.5

Agents as Processes

Agents carry out actions:

- forever infinite horizon
- until some stopping criteria is met indefinite horizon
- finite and fixed number of steps finite horizon

Decision-theoretic Planning

What should an agent do when

- it gets rewards (and punishments) and tries to maximize its rewards received
- actions can be noisy; the outcome of an action can't be fully predicted
- there is a model that specifies the probabilistic outcome of actions
- the world is fully observable : the current state is always fully in evidence

for the various planning horizons?

World State

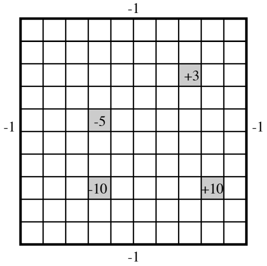
- The world state is the information such that if you knew the world state, no information about the past is relevant to the future. Markovian assumption .
- Let S_i, A_i be the state,action at time i

$$P(S_{t+1}|S_0, A_0, \ldots, S_t, A_t) = P(S_{t+1}|S_t, A_t)$$

$P(s'|s, a)$ is the probability that the agent will be in state s' immediately after doing action a in state s.

- The dynamics is stationary if the distribution is the same for each time point.

Example: Simple Grid World



Grid World Model

- Actions: up, down, left, right.
- 100 states corresponding to the positions of the robot.
- Robot goes in the commanded direction with probability 0.7, and one of the other directions with probability 0.1.
- If it crashes into an outside wall, it remains in its current position and has a reward of -1.
- Four special rewarding states; the agent gets the reward when leaving the state.

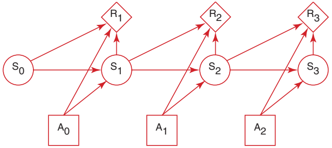
Planning Horizons

The planning horizon is how far ahead the planner looks to make a decision.

- The robot gets flung to one of the corners at random after leaving a positive ($+10$ or $+3$) reward state.
  - the process never halts
  - infinite horizon
- The robot gets $+10$ or $+3$ entering the state, then it stays there getting no reward. These are absorbing states.
  - The robot will eventually reach the absorbing state.
  - indefinite horizon

Decision Processes

- A Markov decision process augments a Markov chain with actions and values (information arcs not shown).



Markov Decision Processes

For an MDP you specify:

- set S of states.
- set A of actions.
- $P(S_{t+1}|S_t, A_t)$ specifies the dynamics.
- $R(S_t, A_t, S_{t+1})$ specifies the reward. The agent gets a reward at each time step (rather than just a final reward). $R(s, a, s')$ is the expected reward received when the agent is in state s, does action a and ends up in state s'.

Information Availability

What information is available when the agent decides what to do?

- fully-observable MDP the agent gets to observe S_t when deciding on action A_t.
- partially-observable MDP (POMDP) the agent has some noisy sensor of the state. It needs to remember its sensing and acting history. It can do this by maintaining a sufficiently complex belief state.

Rewards and Values

Suppose the agent receives the sequence of rewards $r_1, r_2, r_3, r_4, \ldots$. What value should be assigned?

- total reward $V = \sum_{i=1}^{\infty} r_i$
- average reward $V = \lim_{n\to\infty} (r_1 + \cdots + r_n)/n$
- discounted reward $V = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$
  γ is the discount factor $0 \leq \gamma \leq 1$.

Policies

- A stationary policy is a function:

  $$\pi : S \rightarrow A$$

  Given a state s, $\pi(s)$ specifies what action the agent who is following π will do.
- An optimal policy is one with maximum expected discounted reward.
- For a fully-observable MDP with stationary dynamics and rewards with infinite or indefinite horizon, there is always an optimal stationary policy.

Value of a Policy

- $Q^{\pi}(s,a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.
- $V^{\pi}(s)$, where s is a state, is the expected value of following policy π in state s.
- Q^{π} and V^{π} can be defined mutually recursively:

$$\begin{aligned} Q^{\pi}(s,a) &= \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V^{\pi}(s')\right) \\ V^{\pi}(s) &= Q^{\pi}(s,\pi(s)) \end{aligned}$$

Value of the Optimal Policy

- $Q^{*}(s,a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $\pi^{*}(s)$ is the optimal action to take in state s
- $V^{*}(s)$, where s is a state, is the expected value of following the optimal policy in state s.
- Q^{*} and V^{*} can be defined mutually recursively:

$$\begin{aligned} Q^{*}(s,a) &= \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V^{*}(s')\right) \\ V^{*}(s) &= \max_{a} Q^{*}(s,a) \\ \pi^{*}(s) &= argmax_a Q^{*}(s,a) \end{aligned}$$

Value Iteration

- The t-step lookahead value function, V^{t} is the expected value with t steps to go
- Idea: Given an estimate of the t-step lookahead value function, determine the $t+1$-step lookahead value function.

Value Iteration

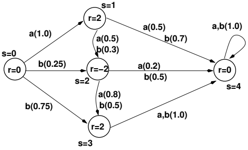
- Set V^0 arbitrarily, $t = 1$
- Compute Q^t, V^t from V^{t-1}.

$$Q^t(s,a) = \left[R(s) + \gamma \sum_{s'} Pr(s'|s,a) V^{t-1}(s')\right]$$

$$V^t(s) = \max_{a} Q^t(s,a)$$

- The policy with t stages to go is simply the actions that maximizes this $\pi^t(s) = \arg\max_a \left[R(s) + \gamma \sum_{s'} Pr(s'|s,a) V^{t-1}(s')\right]$
- This is dynamic programming
- This converges exponentially fast (in t) to the optimal value function.
- Convergence when $||V^t(s) - V^{t-1}(s)|| < \epsilon\frac{(1-\gamma)}{\gamma}$ ensures V^t is within ϵ of optimal ($||X|| = max\{|x|, x \in X\}$)

State space graph (NOT a DBN):



This same graph, represented as a decision network, would have the following factors, where the $(row, col) = (i, j)$ entry in each probability table is $P(S' = j|S = i, A)$

$$P(S'|S, A = a) = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.8 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$P(S'|S, A = b) = \begin{bmatrix} 0.0 & 0.0 & 0.25 & 0.75 & 0.0 \\ 0.0 & 0.0 & 0.3 & 0.0 & 0.7 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \quad R(S) = \begin{bmatrix} 0 \\ 2 \\ -2 \\ 2 \\ 0 \end{bmatrix}$$

first iteration , using $\gamma = 0.9$

$$V^0(s') = R(s')$$
$$Q^1(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a)V^0(s')$$
$$= \begin{bmatrix} 1.8 & 1.1 & -0.56 & 2.0 & 0 \\ 0.9 & 1.46 & -1.1 & 2.0 & 0 \end{bmatrix}$$
$$V^1(s) = max_a(Q^1(s, a))$$
$$= \begin{bmatrix} 1.8 & 1.46 & -0.56 & 2.0 & 0 \end{bmatrix}$$
$$\pi^1(s) = \begin{bmatrix} a & b & a & a & a \end{bmatrix}$$

second iteration

$$Q^2(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a)V^1(s')$$
$$= \begin{bmatrix} 1.31 & 1.75 & -0.56 & 2.0 & 0 \\ 1.22 & 1.85 & -1.1 & 2.0 & 0 \end{bmatrix}$$
$$V^2(s) = max_a(Q^2(s, a))$$
$$= \begin{bmatrix} 1.31 & 1.84 & -0.56 & 2.0 & 0 \end{bmatrix}$$
$$\pi^2(s) = \begin{bmatrix} a & b & a & a & a \end{bmatrix}$$

on convergence, optimal value function is

$$V^*(s) = \begin{bmatrix} 1.66 & 1.85 & -0.56 & 2.0 & 0 \end{bmatrix}$$

policy is

$$\pi^*(s) = \begin{bmatrix} a & b & a & a & a \end{bmatrix}$$

Asynchronous Value Iteration

- You don't need to sweep through all the states, but can update the value function for each state individually.
- This converges to the optimal value function, if each state and action is visited infinitely often in the limit.
- You can either store $V[s]$ or $Q[s,a]$.

Asynchronous VI: storing $V[s]$

- Repeat forever:
  - Select state s;
  - $V[s] \leftarrow \max_a \sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V[s']);$

Asynchronous VI: storing $Q[s,a]$

- Repeat forever:
  - Select state s, action a;
  - $Q[s,a] \leftarrow \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q[s',a']\right);$

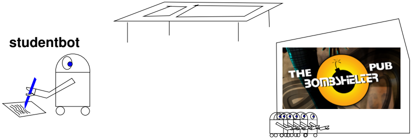
Markov Decision Processes: Factored State

- Represent $S = \{X_1, X_2, \ldots, X_n\}$
- X_i are random variables
- for each X_i, and each action $a \in A$, we have $P(X_i'|S,A)$
- Reward $R(X_1, X_2, \ldots, X_N)$ may be additive:

$$R(X_1, X_2, \ldots, X_N) = \sum_i R(X_i)$$

- Value iteration proceeds as usual but can do one variable at a time (e.g. variable elimination)

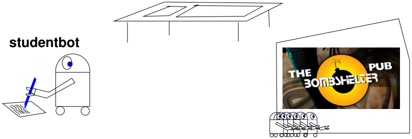
Example: studentbot



state variables (3x2x4x2=48 states):

- tired: studentbot is tired (no/a bit/very)
- passtest: studentbot passes test (no/yes)
- knows: studentbot's state of knowledge (nothing/a bit/a lot/everything)
- goodtime: studentbot has a good time (no/yes)

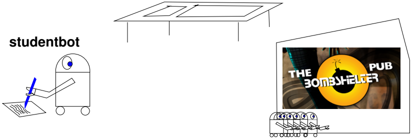
Example: studentbot



studentbot actions:

- study: studentbot's knowledge increases, studentbot gets tired
- sleep: studentbot gets less tired
- party: studentbot has a good time if he's not tired, but gets tired and loses knowledge
- take test: studentbot takes a test (can take test anytime)

Example: studentbot



studentbot rewards:

- +20 if studentbot passes the test
- +2 if studentbot has a good time

basic tradeoff: short term vs. long-term rewards

Studentbot

State-based:

$$P(s'|s,a) = [48 \times 48]$$

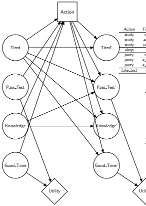
$$R(s) = [48 \times 1]$$

Studentbot

As a dynamic decision network:

			Knowledge'			
Action	Tired	Knowledge	nothing	a_bit	a_lot	everything
study	no	nothing	0.5	0.5	0	0
	no	a_bit	0	0.5	0.5	0
	no	a_lot	0.0	0.0	0.5	0.5
	no	everything	0		0	1.0
	a_bit	nothing	0.5	0.5	0	0
	a_bit	a_bit	0	0.5	0.5	0.0
	a_bit	a_lot	0	0	0.5	0.5
	a_bit	everything	0	0	0	1.0
	very	nothing	1.0	0	0	0
	very	a_bit	0	1.0	0.0	0.0
	very	a_lot	0	0	1.0	0.0
	very	everything	0	0	0	1.0
party	–	nothing	1.0	0.0	0	0
	no	a_bit	0.5	0.5	0.0	0.0
	no	a_lot	0.0	0.5	0.5	0.0
	no	everything	0	0	0.5	0.5
sleep	–	nothing	1.0	0.0	0	0
	–	a_bit	0.0	1.0	0.0	0.0
	–	a_lot	0.0	0.0	1.0	0.0
	–	everything	0	0.0	0.0	1.0
take_test	–	–	1.0	0.0	0	0

		Good_Time'	
Action	Tired	no	yes
party	no	0.0	1.0
party	yes	1.0	0.0
other	–	1.0	0.0

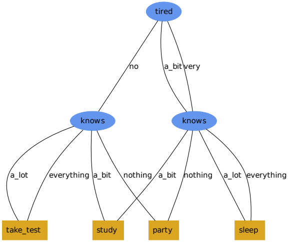


		Tired'		
Action	Tired	no	a_bit	very
study	no	0.8	0.2	0.0
study	abit	0.0	0.8	0.2
study	very	0.0	0.0	1.0
sleep	–	1	0	0
party	no	0	1	0
party	a_bit	0	0	1
party	a_lot	0	0	1
take_test	–	0	0	1

			PassTest'	
Action	Tired	Knowledge	no	yes
take_test	no	nothing	1	0
	no	a_bit	0.9	0.1
	no	a_lot	0.7	0.3
	no	everything	0.1	0.9
	a_bit	nothing	1	0
	a_bit	a_bit	0.9	0.1
	a_bit	a_lot	0.7	0.3
	a_bit	everything	0.1	0.9
	very	–	1	0
else	–	–	1	0

Good_Time	Pass_Test	U(Good_Time, Pass_Test)
yes	yes	22
yes	no	2
no	yes	20
no	no	0

Studentbot Policy



Partially Observable Markov Decision Processes (POMDPs)

A POMDP is like an MDP, but

some variables are not observed. It is

a tuple $\langle S, A, T, R, O, \Omega \rangle$

S: finite set of unobservable states

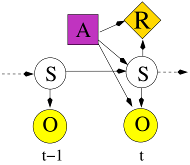
A: finite set of agent actions

$T : S \times A \to S$ transition function

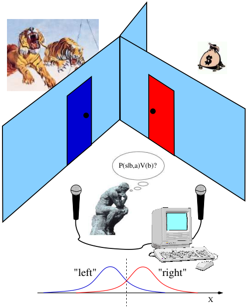
$R : S \times A \to \mathcal{R}$ reward function

O: set of observations

$\Omega : S \times A \to O$ observation function



e.g. 1-D Tiger problem



The material after this is optional

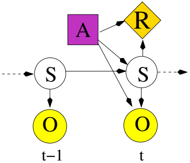
Partially Observable Markov Decision Processes (POMDPs)

A POMDP is like an MDP, but

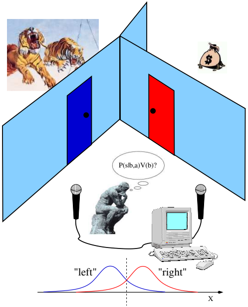
some variables are not observed. It is

a tuple $\langle S, A, T, R, O, \Omega \rangle$

- S: finite set of unobservable states
- A: finite set of agent actions
- $T : S \times A \rightarrow S$ transition function
- $R : S \times A \rightarrow \mathcal{R}$ reward function
- O: set of observations
- $\Omega : S \times A \rightarrow O$ observation function



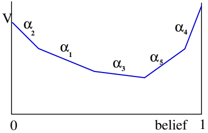
e.g. 1-D Tiger problem

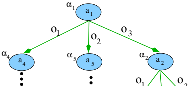


Value Functions and Conditional Plans

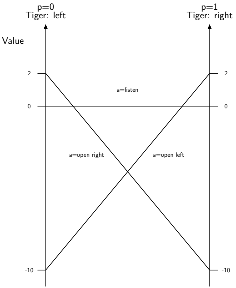
$$V^{k+1}(b) = \max_a R^a(b) + \gamma \sum_o Pr(o|b,a) V^k(b_o^a)$$

$V(b)$ can be represented with a piecewise linear function over the belief space - pieces are called α vectors

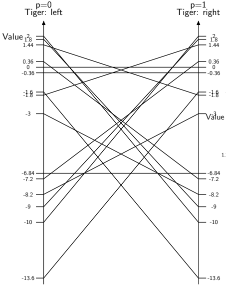




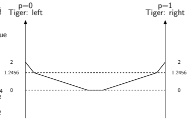
e.g. Tiger problem, after zero iterations



e.g. Tiger problem, after one iteration



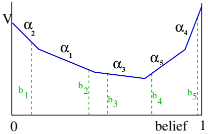
all generated α vectors



optimal value function

Point-based Value Iteration

1. Generate belief samples to make belief set belief set $\mathcal{B}$



2. compute forward-propagated belief states

$$b_o^a(s') = \sum_{s \in S} T(s'|a,s)\Omega(o|s',a)b(s) \quad \forall b \in \mathcal{B}$$

Point-Based Value Iteration II

1. start with one alpha vector: $\alpha_0 = R(s,a)$
2. repeat until converged:
   2.1 for each belief sample, b:

$$\Gamma_b^a = R(s,a) + \sum_{s' \in \mathcal{S}} \sum_{o \in \mathcal{O}} T(s'|a,s)\Omega(o|s',a) \arg\max_{\alpha_j} \alpha_j(s') \cdot b_o^a(s') \ \forall \ a \in \mathcal{A}, b \in \mathcal{B}$$

   2.2 Maximize over actions at each b:

$$\alpha^\dagger = \bigcup_{b \in \mathcal{B}} \{\arg\max_{\Gamma_b^a} (\Gamma_b^a \cdot b_j)\}$$

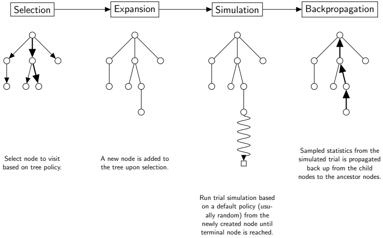
Policies

Policy: maps beliefs states into actions $\pi(b(s)) \rightarrow a$

Two ways to compute a policy

1. Backwards search
   - Dynamic programming (Variable Elimination)
   - in MDP:
     $Q_t(s,a) = R(s,a) + \gamma \sum_{s'} Pr(s'|s,a) \max_{a'} Q_{t-1}(s',a')$
   - in POMDP: $Q_t(b(s),a)$
   - Point-based backups make this efficient
2. Forwards search : Monte Carlo Tree Search (MCTS)
   - Expand the search tree
   - Expand more deeply in promising directions
   - Ensure exploration using e.g. UCB

MCTS



Forward Monte-Carlo Search for POMDPs

procedure GetValue($b(s)$)

for each action-observation pair a,o:

$b^a_o(s') \leftarrow$ propagate the full belief state forwards

for each action and observation (using stochastic simulation)

if $b^a_o(s')$ not at a leaf:

evaluate recursively by further growing the tree:

$V^a_o \leftarrow GetValue(b^a_o(s'))$

else:

create a new leaf for a,o

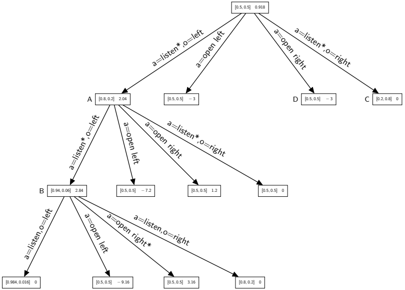
do a series of single-belief point rollouts

(e.g. propagate a single belief forward stochastically gathering reward until termination condition is met),

use the total returned value as V^a_o.

return $R(b(s)) + \max_a \{\gamma \sum_o P(o|b(s),a) \sum_{s'} V^a_o b^a_o(s'))\}$

e.g. Tiger problem, two steps expanded



Next:

- Reinforcement Learning Poole & Mackworth (2nd ed.)Chapter 12.1,12.3-12.9
- Deep Reinforcement Learning