Agents as Processes

Lecture 10 - Planning under Uncertainty (II)

Jesse Hoey
School of Computer Science
University of Waterloo
March 26, 2019

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

Decision-theoretic Planning

Agents carry out actions:
- forever \textit{infinite horizon}
- until some stopping criteria is met \textit{indefinite horizon}
- finite and fixed number of steps \textit{finite horizon}

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. \textbf{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 \textit{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 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

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

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 $s'$.

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 $V = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$
  - $\gamma$ is the discount factor $0 \leq \gamma \leq 1$.

A stationary policy is a function:

$$\pi : S \rightarrow A$$

Given a state $s$, $\pi(s)$ specifies what action the agent who is following $\pi$ will do.

- An optimal policy is one with maximum expected discounted reward.
- For a fully-observables 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 $\pi$.
- $V^\pi(s)$, where $s$ is a state, is the expected value of following policy $\pi$ in state $s$.
- $Q^\pi$ and $V^\pi$ can be defined mutually recursively:

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

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

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: Simple Example

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 $P(S' = j | S = i, A)$

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

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

$R(S) = \begin{bmatrix}
0 \\
2 \\
-2 \\
2 \\
0
\end{bmatrix}$
Value Iteration: Simple Example

first iteration, using $\gamma = 0.9$

\[ V^0(s') = R(s') \]
\[ Q^1(s, a) = R(s) + \sum_{s'} P(s'|s, a) V^0(s') \]
\[ V^1(s) = \max_a Q^1(s, a) \]
\[ \pi^1(s) = [a \ b \ a \ a \ a] \]

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

Asynchronous Value Iteration

You don’t need to sweep through all the states, but can update the value functions for each state individually.

This converges to the optimal value functions, 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 $a$;
- $Q[s, a] \leftarrow \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma \max_{a'} Q[s', a'])$;

Markov Decision Processes: Factored State

Represent $S = \{X_1, X_2, \ldots, X_n\}$
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)
- **knows**: studentbot’s state of knowledge (nothing/a bit/a lot/everything)
- **goodtime**: studentbot has a good time (no/yes)

**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)

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

basic tradeoff: short term vs. long-term rewards

**State-based:**

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

\[ R(s) = [48 \times 1] \]
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
- \( R : S \times A \rightarrow \mathbb{R} \) reward function
- \( O \): set of observations
- \( \Omega : S \times A \rightarrow O \) observation function

Value Functions and Conditional Plans
e.g. 1-D Tiger problem

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

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

e.g. Tiger problem, after zero iterations

Point-based Value Iteration

1. Generate belief samples to make belief set belief set \( B \)
2. compute forward-propagated belief states
\[ b_o^a(s') = \sum_{s \in S} T(s'|a, s)\Omega(o|s', a)b(s) \quad \forall b \in B \]
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, \( b \):
      \[
      \Gamma_b^s = R(s, a) + \sum_{s' \in S, o \in O} T(s'|a, s) \Omega(o|s', a) \arg \max_a \alpha_j(s') b^a_o(s') \quad \forall \ a \in A, \ b \in B
      \]
   2.2 Maximize over actions at each \( b \):
      \[
      \alpha^t = \bigcup_{b \in B} \{ \arg \max_{\Gamma_b^s \cdot b_j} \}
      \]

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 \( a, o \):
   \[
   b^a_o(s') \leftarrow \text{propagate the full belief state forwards}
   \]
   for each action and observation (using stochastic simulation)
   if \( b^a_o(s') \) not at a leaf:
      evaluate recursively by further growing the tree:
      \[
      V^a_o \leftarrow \text{GetValue}(b^a_o(s'))
      \]
   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** \( R(b(s)) + \max_a \{ \gamma \sum_o P(o|b(s), a) \sum_{s'} V^a_o b^a_o(s') \} \)

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

POMDPs for Emotional Interaction

Partially Observable Markov Decision Process:

\[
A: \text{smile, yell, sympathise, do nothing} \quad E: \text{happy, sad, angry, neutral}
\]

\[
E \quad R(E)
\]

<table>
<thead>
<tr>
<th>( E )</th>
<th>( \text{happy} )</th>
<th>( \text{sad} )</th>
<th>( \text{angry} )</th>
<th>( \text{neutral} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{happy}</td>
<td>0.9</td>
<td>0.04</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>\text{sad}</td>
<td>0.01</td>
<td>0.5</td>
<td>0.2</td>
<td>0.28</td>
</tr>
<tr>
<td>\text{angry}</td>
<td>0.01</td>
<td>0.2</td>
<td>0.5</td>
<td>0.28</td>
</tr>
<tr>
<td>\text{neutral}</td>
<td>0.5</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
</tr>
</tbody>
</table>

\[
E \quad R'(E)
\]

<table>
<thead>
<tr>
<th>( E )</th>
<th>( \text{happy} )</th>
<th>( \text{sad} )</th>
<th>( \text{angry} )</th>
<th>( \text{neutral} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{happy}</td>
<td>1.0</td>
<td>-0.1</td>
<td>-1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>\text{sad}</td>
<td>0.05</td>
<td>0.85</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>\text{angry}</td>
<td>0.05</td>
<td>0.05</td>
<td>0.85</td>
<td>0.05</td>
</tr>
<tr>
<td>\text{neutral}</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.85</td>
</tr>
</tbody>
</table>

\[
E \quad R(E)
\]

<table>
<thead>
<tr>
<th>( E )</th>
<th>( \text{happy} )</th>
<th>( \text{sad} )</th>
<th>( \text{angry} )</th>
<th>( \text{neutral} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{happy}</td>
<td>0.85</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>\text{sad}</td>
<td>0.05</td>
<td>0.85</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>\text{angry}</td>
<td>0.05</td>
<td>0.05</td>
<td>0.85</td>
<td>0.05</td>
</tr>
<tr>
<td>\text{neutral}</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.85</td>
</tr>
</tbody>
</table>
Partially Observable Markov Decision Process:

A: smile, yell, sympathise, do_nothing

<table>
<thead>
<tr>
<th>E</th>
<th>happy</th>
<th>sad</th>
<th>angry</th>
<th>neutral</th>
</tr>
</thead>
<tbody>
<tr>
<td>happy</td>
<td>0.99</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>sad</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.08</td>
</tr>
<tr>
<td>angry</td>
<td>0.01</td>
<td>0.2</td>
<td>0.3</td>
<td>0.48</td>
</tr>
<tr>
<td>neutral</td>
<td>0.7</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

E' = E

A: sympathise_with

<table>
<thead>
<tr>
<th>E</th>
<th>happy</th>
<th>sad</th>
<th>angry</th>
<th>neutral</th>
</tr>
</thead>
<tbody>
<tr>
<td>happy</td>
<td>0.8</td>
<td>0.14</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>sad</td>
<td>0.2</td>
<td>0.3</td>
<td>0.1</td>
<td>0.4</td>
</tr>
<tr>
<td>angry</td>
<td>0.01</td>
<td>0.1</td>
<td>0.7</td>
<td>0.18</td>
</tr>
<tr>
<td>neutral</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

E' = E

Next:

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