Probabilistic Planning

Acting when not everything is under your control

16.410/413 Principles of Autonomy and Decision-Making

Pedro Santana (psantana@mit.edu)

October 14th, 2015.
Assignments

• Problem set 5
  – Out last Wednesday.
  – Due at midnight **this Friday**.
• Midterm on Monday, October 19th. Good luck, even though you won’t need it! ;)
• Readings
  – “Beyond Classical Search” [AIMA], Sections 4.3 and 4.4;
  – “Quantifying Uncertainty” [AIMA], Ch. 13;
  – “Making Complex Decisions” [AIMA], Ch. 17;
1. Motivation

*Where can probabilistic planning be useful?*
Science scouts

Courtesy of Andrew J. Wang
Collaborative manufacturing
Power supply restoration

Thiébaux & Cordier (2001)
How this relates to what we’ve seen?

L06: Activity Planning  
David Wang

Planning as state-space search.

L07: Activity Planning  
David Wang

Observe-Act as AND-OR search.

L08: Markov Decision Processes  
Brian Williams

Plans that optimize utility

L09: Hidden Markov Models  
Pedro Santana

Markovian hidden state inference.

L10: Adversarial Games  
Tiago Vaquero

L11: Probabilistic Planning

Computing utility with probabilistic transitions.

10/09/15 Recitation  
Enrique Fernández

P. Santana, 16.410/413 - Probabilistic Planning  
7/46
Our goal for today

How can we generate plans that optimize performance when controlling a system with stochastic transitions and hidden state?
Today’s topics

1. Motivation
2. MDP recapitulation
3. Search-based probabilistic planning
4. AO*
5. LAO*
6. Extension to hidden states
2. MDP recapitulation

*Where we will:*
- recall what we’ve learned about MDPs;
- learn that recap = recapitulation.
Elements of an MDP

$\mathcal{S}$: discrete set of states.

$\mathcal{A}$: discrete set of actions.

$T: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$, transition function

$R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, reward function.

$\gamma \in [0,1]$: discount factor

$T(s_k, a_k, s_{k+1}) = \Pr(s_{k+1}|s_k, a_k)$
Solving MDPs through VI

\[ V(s_k, a_k) = R(s_k, a_k) + \gamma \sum_{s_{k+1}} V^*(s_{k+1}) T(s_k, a_k, s_{k+1}) \]

Expected reward of executing \( a_k \) at \( s_k \)
Immediate reward
Optimal reward at \( s_{k+1} \)
Expected, discounted optimal future reward

Value iteration

\[ V^*(0)(s_k), \forall s_k \]
\[ V^*(t+1)(s_k) = \max_{a_k} R(s_k, a_k) + \gamma \sum_{s_{k+1}} V^*(t)(s_{k+1}) T(s_k, a_k, s_{k+1}) \]

After convergence

\[ \pi^*(s) = \arg \max_a V(s, a) \]

[AIMA] Section 17.2
VI and goal regression

Dynamic programming works backwards from the goals.

\[ V^*(s_k) = \max_{a_k} R(s_k, a_k) + \gamma \sum_{s_{k+1}} V^*(s_{k+1}) T(s_k, a_k, s_{k+1}) \]
How does VI scale?
How does VI scale?

VI allows one to compute the optimal policy from every state reaching the goal.

Grows linearly with $|\mathbb{S}|$.

$|\mathbb{S}|$ grows exponentially with the dimensions of $\mathbb{S}$.

What if we only care about policies starting at one (or a few) possible initial states $s_0$?

Heuristic forward search!
Searching from an initial state $s_0$

- Subset of $\mathbb{S}$ reachable from $s_0$
- Subset of $\mathbb{S}$ on the optimal path from $s_0$ to $s_{goal}$

Explored by HFS

Explored by VI

Good heuristic: $\approx$

Bad heuristic: $\approx$

10/14/2015
3. Search-based probabilistic planning

L06: Activity Planning

Planning as state-space search.

Observe-Act as AND-OR search.

L08: Markov Decision Processes

Plans that optimize utility

Computing utility with probabilistic transitions.

L10: Adversarial Games

10/09/15 Recitation
Searching on the state graph
Elements of probabilistic search

$S$: discrete set of states.

$A$: discrete set of actions.

$T: S \times A \times S \rightarrow [0,1]$, transition function

$R: S \times A \rightarrow \mathbb{R}$, reward function.

$s_0 \in S$: initial state.

$S_g \subseteq S$: (terminal) goal states.

Could we frame previously-seen shortest path problems like this? How?
(Probabilistic) AND-OR search

OR node
agent action

AND node
stochastic transition

\[ s_{k+1} = 1 \]
\[ s_{k+1} = 2 \]
\[ \ldots \]
\[ s_{k+1} = d \]

\[ T(s_k, a_1, s_{k+1} = 1) \]
\[ T(s_k, a_1, s_{k+1} = d) \]
Hypergraph representation

All nodes are OR nodes

Actions yield lists of successors annotated with probabilities

If every action has a single successor, we go back to a “standard” graph.
4. AO* (Nilson, 1982)

Like A*, but for AND-OR search
AO* in a nutshell

• *Input*: implicit AND-OR search problem

\[ < \mathcal{S}, \mathcal{A}, T, R, s_0, \mathcal{S}_g > \]

*and an admissible heuristic function* \( h: \mathcal{S} \rightarrow \mathbb{R} \).

• *Output*: *optimal* policy in the form of an *acyclic hypergraph* mapping states to actions.
  – Cyclic policies: use LAO* (which we’ll see in a bit).

• *Strategy*: incrementally build solutions forward from \( s_0 \), using \( h \) to estimate future utilities (just like A*!). The set of explored solutions form the *explicit* hypergraph \( G \), and the subset of \( G \) corresponding to the *current estimate of the best policy* is called the *greedy* hypergraph \( g \).
Admissible utility estimates

\[
V(s_k, a_k) = R(s_k, a_k) + \sum_{s_{k+1}} V^*(s_{k+1}) T(s_k, a_k, s_{k+1})
\]

- Expected reward of executing \(a_k\) at \(s_k\)
- Immediate reward
- Optimal reward at \(s_{k+1}\)
- Expected optimal future reward

\[
h(s_{k+1}) \geq V^*(s_{k+1})
\]

Admissible (“optimistic”) estimate of future reward.
Should be “easy” to compute.

\[
V_h(s_k, a_k) = R(s_k, a_k) + \sum_{s_{k+1}} h(s_{k+1}) T(s_k, a_k, s_{k+1}) \geq V(s_k, a_k)
\]
AO* example

Node in $g$, the greedy graph.

Node in $G$, the explicit graph, but not in $g$.

Known value

Heuristic estimate

$\text{s}_0$
Start

*Open nodes:* $[s_0]$

$g = \{ s_0: \text{None} \}$

$G$ starts just with just the initial state $s_0$.
**Expansion**

*Open nodes: [sₐ]*

g={sₐ: None }

1. **Choose an open node to expand → s₀**

2. **Estimate the value of the leaf nodes using the heuristic h.**
Backup

Open nodes: $[s_0]$

$g=\{s_0: \text{None} \}$

3. Backup values for the currently expanded node ($s_0$) and all its ancestors that are part of $g$ (no ancestors), recording the best value at each node.

$$V_h(s_k, a_k) = R(s_k, a_k) + \sum_{s_{k+1}} h(s_{k+1}) T(s_k, a_k, s_{k+1})$$

\[
V_h(s_0, a_1) = -1 + (-10 \times 0.7 - 11 \times 0.3) = -11.3
\]

\[
V_h(s_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6
\]
Update $g$

Open nodes: $[s_0]$

$g = \{ s_0: None \}$

4. Update $g$ and the list of open nodes (non-terminal) by selecting the best action at the nodes which got their values updated

$$V_h(s_0, a_1) = -1 + (-10 \times 0.7 - 11 \times 0.3) = -11.3$$

$$V_h(s_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6$$
4. Update $g$ and the list of open nodes (non-terminal) by selecting the best action at the nodes which got their values updated.

\[
V_h(s_0, a_1) = -1 + (-10 \times 0.7 - 11 \times 0.3) = -11.3
\]

\[
V_h(s_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6
\]
Expansion

Open nodes: \([s_1^1, s_1^2]\]

\(g = \{s_0: a1\}\)

\[R(s_0, a_1) = -1\]

\[R(s_0, a_2) = -2\]

\([-11.3, -11.6]\]

1. Choose any open node to expand \(\rightarrow s_1^2\)

2. Estimate the value of the leaf nodes using the heuristic \(h\).
3. Backup values for the currently expanded node ($s_1^2$) and all its ancestors that are part of $g$ ($s_0$), recording the best value at each node.

\[
V_h(s_1^2, a_1) = -1 + (-10 \times 0.2 - 12 \times 0.8) = -12.6 \\
V_h(s_1^2, a_2) = -2 + (-13 \times 0.5 - 14 \times 0.5) = -15.5 \\
V_h(s_0, a_1) = -1 + (-10 \times 0.7 - 12.6 \times 0.3) = -11.78 \\
V_h(s_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6
\]
Update $g$

Open nodes: $[s^1_1, s^2_1]$

$g=\{s_0: a1\}$

\[
V_h(s^2_1, a_1) = -1 + (-10 \times 0.2 - 12 \times 0.8) = -12.6
\]

\[
V_h(s^2_1, a_2) = -2 + (-13 \times 0.5 - 14 \times 0.5) = -15.5
\]

\[
V_h(s_0, a_1) = -1 + (-10 \times 0.7 - 12.6 \times 0.3) = -11.78
\]

\[
V_h(s_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6
\]

4. Update $g$ and the list of open nodes (non-terminal) by selecting the best action at the nodes which got their values updated.

From leafs to the root
Update $g$

Open nodes: $[s^3_1, s^4_1]$

$g = \{s_0: a2\}$

$R(s_0, a_1) = -1$

$R(s_0, a_2) = -2$

4. Update $g$ and the list of open nodes (non-terminal) by selecting the best action at the nodes which got their values updated.

\[
V_h(s^2_1, a_1) = -1 + (-10 \times 0.2 - 12 \times 0.8) = -12.6
\]

\[
V_h(s^2_1, a_2) = -2 + (-13 \times 0.5 - 14 \times 0.5) = -15.5
\]

\[
V_h(s_0, a_1) = -1 + (-10 \times 0.7 - 12.6 \times 0.3) = -11.78
\]

\[
V_h(s_0, a_2) = -2 + (-10 \times 0.6 - 9 \times 0.4) = -11.6
\]

From leafs to the root
Search terminates when the list of open nodes is empty, i.e., all leaf nodes in $g$ are terminal (goals).

At this point, return $g$ as the optimal policy $\pi$. 

Termination

Open nodes: $[s_1^3, s_1^4]$  
$g=\{s_0: a2\}$
AO*’s pseudocode

Input: < S, A, T, R, s₀, S_g >, heuristic h.

Output: Policy π: S → A

Explicit graph G ← s₀, g ← Best partial policy of G

while best partial policy graph g has nonterminal leafs

  m ← Expand any nonterminal leaf from g and add children to G
  Z ← set containing m and all of its predecessors that are part of g

while Z is not empty

  n ← Remove from Z a node with no descendants in Z
  Update utilities (V values) for n

  π ← Choose next best action at n

Update g with the new π

g is the graph obtained by following π from s₀

Bellman backups

Heuristics for value-to-go
5. LAO* (Hansen & Zilberstein, 2001)

What happens if we find loops in the policy?
Acyclic value updates

Policy nodes are updated no more than once.

$$V_h(s_k, a_k) = R(s_k, a_k) + \sum_{s_{k+1}} h(s_{k+1}) T(s_k, a_k, s_{k+1})$$
Loops require iteration

\[ V_h(s_k, a_k) = R(s_k, a_k) + \gamma \sum_{s_{k+1}} h(s_{k+1}) T(s_k, a_k, s_{k+1}) \]

Value iteration

\[ V^{(0)}(s_k) = h(s_k), \text{ for } s_k \text{ among the policy nodes} \]

\[ V^{(t+1)}(s_k) = \max_{a_k} R(s_k, a_k) + \gamma \sum_{s_{k+1}} V^{(t)}(s_{k+1}) T(s_k, a_k, s_{k+1}) \]

Updates go both directions
LAO* in a nutshell

• **Input**: MDP $< S, A, T, R, \gamma >$, initial state $s_0$, and an admissible heuristic $h: S \rightarrow \mathbb{R}$.

• **Output**: **optimal** policy mapping states to actions.

• **Strategy**: same as in AO*, but value updates are performed through value or policy iteration.
VI on $g$

Open nodes: $[s_1^1, s_1^2]$

$g=\{s_0; a1\}$

Perform VI on the expanded node and all of its ancestors in $g$.

Value iteration on policy nodes

$$V^{(0)}(s_k) = h(s_k), \text{ for } s_k \text{ among the policy nodes}$$

$$V^{(t+1)}(s_k) = \max_{a_k} R(s_k, a_k) + \gamma \sum_{s_{k+1}} V^{(t)}(s_{k+1}) T(s_k, a_k, s_{k+1})$$
6. Extension to hidden state

What changes if the state isn’t directly observable?

L09: Hidden Markov Models

Pedro Santana
From states to belief states

Discrete belief state

\[
\Pr(S_k | o_{1:k}) = \hat{p}_k
\]

Filtering (forward)

(see L09: HMMs)
Incorporating HMM observations

\[
\hat{p}_k = \Pr(S_k | o_{1:k})
\]

\[
\tilde{p}_k = \Pr(S_{k+1} | a_k, o_{1:k}) = T(a_k) \hat{p}_k \quad \text{(prediction)}
\]

\[
\Pr(o_{k+1} | a_k, \hat{p}_k) = \sum_{S_{k+1}} \Pr(o_{k+1} | S_{k+1}) \tilde{p}_k(S_{k+1})
\]

HMM obs. model

\[
Pr(S_{k+1} | o_{1:k}, o_{k+1} = 1)
\]

Pr(S_{k+1} | o_{1:k}, o_{k+1} = m) \quad \text{(filtering)}
Estimating belief state utility

\[ V_h(s_k, a_k) = R(s_k, a_k) + \sum_{s_{k+1}} T(s_k, a_k, s_{k+1}) h(s_{k+1}) \]

\[ V_h(\hat{p}_k, a_k) = \sum_{s_k} \hat{p}_k(s_k) R(s_k, a_k) + \sum_{o_{k+1}} \Pr(o_{k+1}|a_k, \hat{p}_k) H(\hat{p}_k, o_{k+1}) \]

\[ H(\hat{p}_k, o_{k+1}) = \sum_{s_{k+1}} \Pr(S_{k+1} = s_{k+1}|o_{1:k}, o_{k+1}) h(s_{k+1}) \]
Partially observable AO* in a nutshell

• **Input:** < \( S, A, T, R, O, S_0, h, \hat{p}_0 >, \) where \( O \) is the HMM observation model and \( \hat{p}_0 \) is the initial belief.

• **Output:** optimal policy in the form of an **acyclic hypergraph** mapping beliefs to actions.

• **Strategy:** same as in AO*, replacing: \( s_0 \) by \( \hat{p}_0 \); \( T(s_k, a_k, s_{k+1}) \) by \( \Pr(o_{k+1} | a_k, \hat{p}_k) \); \( V_h(s_k, a_k) \) by \( V_h(\hat{p}_k, a_k) \).