Probabilistic planning based on Markov decision processes

Jim Blythe
November 13th

(slides due to Craig Boutilier)
Markov Decision Processes

- Components of a fully observable MDP:
  - states $S$ ($|S| = n$)
  - actions $A$
  - transition function $Pr(s,a,t)$
    - represented by set of $n \times n$ stochastic matrices
  - reward function $R(s)$
    - represented by $n$-vector

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>...</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>0.9</td>
<td>0.05</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>$S_2$</td>
<td>0.0</td>
<td>0.20</td>
<td>...</td>
<td>0.1</td>
</tr>
<tr>
<td>$S_n$</td>
<td>0.1</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
</tbody>
</table>

$R$:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>...</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_n$</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graphical View of MDP
Policies

- Stationary policy \( \pi \) (does not depend on time)

  \begin{itemize}
  \item \( \pi: S \rightarrow A \)
  \item \( \pi(s) = \) action chosen at state \( s \)
  (universal plan)
  \end{itemize}
Value of a Policy

- How good is a policy $\pi$? How do we measure “accumulated” reward?
- **Value function** $V: S \rightarrow R$ associates value with each state
- $V(\pi, s)$ denotes value of policy $\pi$ at state $s$
  - expected accumulated reward over horizon of interest
  - note $V(\pi, s)$ is not $R(s)$; it measures utility
- **Common formulations of value:**
  - Finite horizon $n$: total expected reward given $\pi$
  - Infinite horizon discounted: discounting keeps total bounded
Finite Horizon Problems

- Utility (value) depends on stage-to-go
  - hence so should policy: nonstationary $\pi(s,k)$

- $V^k_{\pi}(s)$ is $k$-stage-to-go value function for $\pi$

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

- Here $R_t$ is a random variable denoting reward received at stage $t$
Compute V iteratively

- Compute $V^k_{\pi}(s)$ by dynamic programming:

(a) $V^0_{\pi}(s) = R(s), \forall s$

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

\[\begin{align*}
\pi(s, k) & \quad 0.7 \\
& \quad 0.3 \\
\forall k & \quad \forall k-1
\end{align*}\]
Value Iteration

- Markov property allows exploitation of DP principle for optimal policy construction
  - no need to enumerate $|A|^T_n$ possible policies

- Value Iteration

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

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

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

$V^k$ 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: implied policy

\[ \Pi^+(s_4) = \max \{ \text{[orange, green]} \} \]
This gives an optimal policy

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

- 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
- “Trick”: introduce discount factor $0 \leq \beta < 1$
  - Future rewards discounted by $\beta$ per time step

\[ V^k_\pi(s) = E \left[ \sum_{t=0}^{\infty} \beta^t R^t \mid \pi, s \right] \]
Things about policies

- Optimal policy maximizes value at each state
- Optimal policies guaranteed to exist (Howard60)
- We define $V^*(s) = V(\pi, s)$ for some optimal $\pi$. 
Value Iteration (again)

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

\[ V^k(s) = R(s) + \beta \max_a \sum_{s'} \Pr(s, a, s') \cdot V^{k-1}(s') \]
How to act based on an approximation to $V^*$

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

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

  - if $V$ within $\varepsilon$ of $V^*$, then $V(\pi)$ within $2\varepsilon$ of $V^*$

- There exists an $s$ s.t. optimal policy is returned
  - even if value estimate is off, greedy policy is optimal
Policy Iteration

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

- 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
So why not just use MDPs?

- Most AI problems are **feature-based**
  - \(|S|\) exponential in number of variables
  - Spec./Rep’n of problem in state form impractical
  - Explicit state-based DP impractical

- Require structured representations
  - exploit regularities in probabilities, rewards

- Require structured computation
  - exploit regularities in policies, value functions
  - can aid in approximation (anytime computation)
Structured representation

- States decomposable into state variables

\[ S = X_1 \times X_2 \times \ldots \times X_n \]

- Structured representations the norm in AI
  - STRIPS, Sit-Calc., Bayesian networks, etc.
  - Describe how actions affect/depend on features
  - Natural, concise, can be exploited computationally

- Same ideas can be used for MDPs
  - actions, rewards, policies, value functions, etc.
  - dynamic Bayes nets [DeanKanazawa89, BouDeaGol95]
  - decision trees and diagrams [BouDeaGol95, Hoeyetal99]
Action Representation – DBN/ADD

Pickup Printout

\[
\begin{array}{c|cc}
J & J_{(t+1)} & J_{(t+1)} \\
T & 1.0 & 0.0 \\
F & 0.0 & 1.0 \\
\end{array}
\]

\[f_J(J_t, J_{t+1})\]

\[
\begin{array}{cccc|cc}
L & E & P & P_{(t+1)} & P_{(t+1)} \\
T & T & T & 1.0 & 0.0 \\
F & T & T & 1.0 & 0.0 \\
T & F & T & 1.0 & 0.0 \\
F & F & T & 1.0 & 0.0 \\
T & T & F & 0.8 & 0.2 \\
F & T & F & 0.0 & 1.0 \\
T & F & F & 0.0 & 1.0 \\
F & F & F & 0.0 & 1.0 \\
\end{array}
\]

\[f_P(L_t, P_t, E_t, P_{t+1})\]
Action Representation – DBN/ADD

\[ Pr(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1} \mid J_t, L_t, P_t, E_t) \]

\[ = f_J(J_t, J_{t+1}) \times f_P(L_t, P_t, E_t, P_{t+1}) \]

\[ \times f_L(L_t, L_{t+1}) \times f_E(E_t, E_{t+1}) \]

- Only 28 parameters vs. 256 for matrix

- Removes global exponential dependence
Action Representation – DBN/ADD

- ADDs, decision trees, Horn rules, ... (natural)

Pickup Printout

Diagram (ADD)
Reward Representation

- Rewards represented similarly
  - save on $2n$ size of vector rep’n
Reward Representation

- Additive independent reward also very common
  - as in multiattribute utility theory
  - offers more natural and concise representation for many types of problems

![Diagram showing additive independent reward](image_url)
Key Question: Structured computation

- Given compact representation, can we solve MDP without explicit state space enumeration?
- Can we avoid \( O(|S|) \)-computations by exploiting regularities made explicit by DBNs/ADDs?
State Space Abstraction

- General method: *state aggregation*
  - group states, treat aggregate as single state

- *Abstraction* is a specific aggregation technique
  - aggregate by ignoring details (features)
  - ideally, focus on *relevant* features
Graphical view of abstraction
Decision-Theoretic Regression

- Goal regression a classical abstraction method
  - \( \text{Regr}(G,a) \) is a logical condition \( C \) under which \( a \) leads to \( G \) (aggregates \( C \) states and \( \sim C \) states)

- Decision-theoretic analog: given "logical description" of \( V_{t+1} \), produce such a description of \( V_t \) or optimal policy (e.g., using ADDs)

- Cluster together states at any point in calculation with \textit{same best action} (policy), or with \textit{same value} (VF)
Graphical view of decision-theoretic regression
Generally, $V_{t+1}$ depends on only a subset of variables (usually in a structured way).

What is value of action $a$ at time $t$ (at any $s$)?

$V_{t+1}$

$P$

$E$

20

0
Decision-theoretic refinement

- Assume VF $V_{t+1}$ is structured: what is value of doing action $a$ at time $t$?

$$V^a_t(J_t,L_t,P_t,E_t)$$
Decision-theoretic refinement

Assume VF \( V_{t+1} \) is structured: what is value of doing action \( a \) at time \( t \)?

\[
V^a_t(J_t, L_t, P_t, E_t) = R + \sum_{J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1}} \Pr^a(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1} | J_t, L_t, P_t, E_t) \ V_{t+1}(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1})
\]
Assume VF $V_{t+1}$ is structured: what is value of doing action $a$ at time $t$?

$$V^a_t(J_t, L_t, P_t, E_t)$$

$$= R^+ \sum_{J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1}} \Pr^a(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1} \mid J_t, L_t, P_t, E_t) \ V_{t+1}(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1})$$

$$= R^+ \sum_{J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1}} f_J(J_t, J_{t+1}) f_P(L_t, P_t, E_t, P_{t+1}) f_L(L_t, L_{t+1}) f_E(E_t, E_{t+1}) \ V_{t+1}(P_{t+1}, E_{t+1})$$
Decision-theoretic refinement

Assume VF $V_{t+1}$ is structured: what is value of doing action $a$ at time $t$?

\[
V^a_t(J_t, L_t, P_t, E_t) = R + \sum_{J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1}} \Pr^a(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1} \mid J_t, L_t, P_t, E_t) \ V_{t+1}(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1})
\]

\[
= R + \sum_{J_t, L_t, P_t, E_{t+1}} f_J(J_t, J_{t+1}) f_P(L_t, P_t, E_t, P_{t+1}) f_L(L_t, L_{t+1}) f_E(E_t, E_{t+1}) \ V_{t+1}(P_{t+1}, E_{t+1})
\]

\[
= R + \sum_{L_t, P_t, E_{t+1}} f_P(L_t, P_t, E_t, P_{t+1}) f_L(L_t, L_{t+1}) f_E(E_t, E_{t+1}) \ V_{t+1}(P_{t+1}, E_{t+1})
\]
Decision-theoretic refinement

- Assume VF $V_{t+1}$ is structured: what is value of doing action $a$ at time $t$?

$$V^a_t(J_t, L_t, P_t, E_t)$$

$$= R + \sum_{J_t, L_t, P_{t+1}, E_{t+1}} P_t^a(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1} | J_t, L_t, P_t, E_t) \cdot V_{t+1}(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1})$$

$$= R + \sum_{J_t, L_t, P_{t+1}, E_{t+1}} f_J(J_t, J_{t+1}) \cdot f_L(L_t, L_{t+1}) \cdot f_P(L_t, P_t, E_t, P_{t+1}) \cdot f_E(E_t, E_{t+1}) \cdot V_{t+1}(P_{t+1}, E_{t+1})$$

$$= R + \sum_{L_t, P_{t+1}, E_{t+1}} f_L(L_t, L_{t+1}) \cdot f_P(L_t, P_t, E_t, P_{t+1}) \cdot f_E(E_t, E_{t+1}) \cdot V_{t+1}(P_{t+1}, E_{t+1})$$

- $V_t(a)$ depends on subset of variables as well
  - Each component function represented as ADD
  - ADD operations allow structure to be preserved
Planning by DTR

- Standard DP algorithms can be implemented using structured DTR

- All operations exploit ADD rep’n and algorithms
  - multiplication, summation, maximization of functions
  - standard ADD packages very fast

- Several variants possible
  - MPI/VI with decision trees [BouDeaGol95,00; Bou97; BouDearden96]
  - MPI/VI with ADDs [HoeyStAubinHuBoutilier99, 00]
Structured Value Iteration

- Assume compact representation of $V_k$
  - start with $R$ at stage-to-go 0 (say)
- For each action $a$, compute $Q_{k+1}$ using variable elimination on the two-slice DBN
  - eliminate all $k$-variables, leaving only $k+1$ variables
  - use ADD operations if initial rep’n allows
- Compute $V_{k+1} = \max_a Q_{k+1}$
  - use ADD operations if initial representation allows

- Policy iteration can be approached similarly
Structured Policy and Value Function

Diagram showing a tree structure with nodes labeled as Noop, HCR, HCU, Loc, BuyC, DelC, W, R, U, Go, and GetU. The edges and values indicate transitions and costs.
Structured Policy Evaluation: Trees

- Assume a tree for $V_t$, produce $V_{t+1}$
- For each distinction $Y$ in Tree($V_t$):
  a) use 2TBN to discover conditions affecting $Y$
  b) piece together using the structure of Tree($V_t$)

- Result is a tree exactly representing $V_{t+1}$
  - dictates conditions under which leaves (values) of Tree($V_t$) are reached with fixed probability

- A decision theoretic form of regression
A Simple Action/Reward Example

Network Rep’n for Action $A$

Reward Function $R$
Example: Generation of V1

\[ V^0 = \mathbb{R} \quad \text{Step 1} \quad \text{Step 2} \quad \text{Step 3: } V^1 \]
Example: Generation of V2

V1  Step 1  Step 2
DTR: Relative Merits

- Adaptive, nonuniform, exact abstraction method
  - provides exact solution to MDP
  - much more efficient on certain problems (time/space)
  - 400 million state problems (ADDs) in a couple hrs

- Some drawbacks
  - produces piecewise constant VF
  - some problems admit no compact solution representation (though ADD overhead “minimal”)
  - approximation may be desirable or necessary
Can easily approximate DTR:

- Simple *pruning* of value function
  - Can prune trees [BouDearden96] or ADDs [StaubinHoeyBou00]

- Gives regions of *approximately same value*
A pruned value ADD
Approximate DTR: Relative Merits

- Relative merits of ADTR
  - fewer regions implies faster computation
  - can provide leverage for optimal computation
  - 30-40 billion state problems in a couple hours
  - allows fine-grained control of time vs. solution quality with dynamic (a posteriori) error bounds
  - technical challenges: variable ordering, convergence, fixed vs. adaptive tolerance, etc.

- Some drawbacks
  - (still) produces piecewise constant VF
  - doesn’t exploit additive structure of VF at all
Summary

- MDPs produce optimal solutions and can exploit dynamic programming, BUT state space hard to control

- We can make the standard MDP algorithms use structured representations of actions and reward, by backing up the structure through the policy and value functions.

- Often, we still need to make approximations.