Planning with MDPs
(Markov Decision Processes)

Héctor Geffner
ICREA and Universitat Pompeu Fabra
Barcelona, Spain
Status of Classical Planning

• Classical planning works!!
  – Large problems solved very fast (non-optimally)

• Model simple but useful
  – Operators not primitive; can be policies themselves
  – Fast closed-loop replanning able to cope with uncertainty sometimes

• Limitations
  – Does not model Uncertainty (no probabilities)
  – Does not deal with Incomplete Information (no sensing)
  – Deals with very Simple Cost Structure (no state dependent costs)
Beyond Classical Planning: Two Strategies

1. **Develop** solver for more general models; e.g., MDPs and POMDPs
   
   +: generality
   -: complexity

2. **Extend** the scope of current 'classical' solvers

   +: efficiency
   -: generality

We will pursue first approach here . . .
Reminder: Basic State Models

- Characterized by:
  - finite and discrete state space $S$
  - an initial state $s_0 \in S$
  - a set $G \subseteq S$ of goal states
  - actions $A(s) \subseteq A$ applicable in each state $s \in S$
  - a transition function $F(a, s)$ for $s \in S$ and $a \in A(s)$
  - action costs $c(a, s) > 0$

- A **solution** is a sequence of applicable actions $a_i$, $i = 0, \ldots, n$, that maps the initial state $s_0$ into a goal state $s \in S_G$; i.e.,

$$s_{i+1} = f(a_i, s_i) \text{ and } a_i \in A(s_i) \text{ for } i = 0, \ldots, n \text{ and } s_{n+1} \in S_G$$

- **Optimal** solutions minimize total cost $\sum_{i=0}^{i=n} c(a_i, s_i)$, and can be computed by **shortest-path** or **heuristic search** algorithms . . .
MDPs are fully observable, probabilistic state models:

- a state space $S$
- a set $G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- action costs $c(a, s) > 0$

- Solutions are functions (policies) mapping states into actions
- Optimal solutions have minimum expected costs
Partially Observable MDPs (POMDPs)

POMDPs are partially observable, probabilistic state models:

- states \( s \in S \)
- actions \( A(s) \subseteq A \)
- costs \( c(a, s) > 0 \)
- transition probabilities \( P_a(s'|s) \) for \( s \in S \) and \( a \in A(s) \)
- initial belief state \( b_0 \)
- final belief states \( b_F \)
- sensor model given by probabilities \( P_a(o|s) \), \( o \in Obs \)

- **Belief states** are probability distributions over \( S \)
- **Solutions** are policies that map belief states into actions
- **Optimal** policies minimize expected cost to go from \( b_0 \) to \( b_F \)
Illustration: Navigation Problems

Consider robot that has to reach target $G$ when

1. *initial state is known and actions are deterministic*
2. *initial state is unknown and actions are deterministic*
3. *states are fully observable and actions are stochastic*
4. *states are partially observable and actions are stochastic* . . .

– How do these problems map into the models considered?
– What is the form of the solutions?
Solving State Models by Dynamic Programming

• Solutions to **wide range** of state models can be expressed in terms of solution of **Bellman equation** over **non-terminal** states $s$:

$$V(s) = \min_{a \in A(s)} Q_V(a, s)$$

where **cost-to-go** term $Q_V(a, s)$ depends on model:

- for MDPs:
  $$c(a, s) + \sum_{s' \in F(a, s)} P_a(s'|s)V(s')$$
- for Max AND/OR Graphs:
  $$c(a, s) + \max_{s' \in F(a, s)} V(s')$$
- for OR Graphs . . .

($F(a, s)$: set of successor states; for terminal states, $V(s) = V^*(s)$ assumed)

• The **greedy policy** $\pi_V(s) = \arg\min_{a \in A(s)} Q_V(a, s)$ is **optimal** when $V = V^*$ solves Bellman

• **Question:** how to get $V^*$?
Value Iteration (VI)

- **Value Iteration** finds $V^*$ by successive approximations

- Starting with an arbitrary $V$, uses Bellman equation to update $V$

  \[ V(s) := \min_{a \in A(s)} Q_V(a, s) \]

- If all states updated a sufficient number of times (and certain general conditions hold), left and right hand sides converge to $V = V^*$

- Example: ...
Value Iteration: Benefits and Limitations

- VI is a very simple and general algorithm (can solve wide range of models)

- **Problem:** VI is exhaustive; value function $V(s)$ is a vector of the size of the problem space

- In particular, it does not compete with heuristic search algorithms such as A* or IDA* for solving OR-graphs (deterministic problems) . . .

- **Question:** can VI be 'modified' to deal with larger state spaces that do not fit into memory, without giving up optimality?

- Yes, use Lower Bounds and Initial State as in Heuristic Search methods . . .
Focusing Value Iteration using LBs and $s_0$: Find and Update

- Say that a state $s$ is
  - **greedy** if reachable from $s_0$ using greedy policy $\pi_V$, and
  - **inconsistent** if $V(s) \neq \min_{a \in A(s)} Q_V(a, s)$

- Then starting with an **admissible** and **monotone** $V$, follow loop:
  - **Find an inconsistent greedy state $s$ and Update it**

- **Find-and-Update** loop delivers greedy policy that is **optimal even if some states not updated or visited at all!**

- Recent **heuristic search algorithms for MDPs**, like RTDP, LAO*, and LDFS; all implement this loop in various forms

- We will focus here on RTDP (Barto, Bradke, Singh, 95)
**Greedy Policy for Deterministic MDP**

The **Greedy policy** is a closed-loop version of greedy search.

1. **Evaluate** each action $a$ applicable in $s$

   \[ Q(a, s) = c(a, s) + h(s_a) \]
   where $s_a$ is next state

2. **Apply** action $a$ that minimizes $Q(a, s)$

3. **Observe** resulting states $s'$

4. **Exit** if $s'$ is goal, else go to 1 with $s := s'$

- Greedy policy based on $h$ can be written as $\pi_h(s) = \arg\min_{a \in A(s)} Q(a, s)$
- $\pi_h$ is **optimal** when $h = h^*$, otherwise non-optimal and may get trapped into loops
Update heuristic $h$ as you move, to make it consistent with Bellman

1. **Evaluate** each action $a$ applicable in $s$
   
   \[ Q(a, s) = c(a, s) + h(s_a) \] where $s_a$ is next state

2. **Apply** action $a$ that minimizes $Q(a, s)$

3. **Update** $V(s)$ to $Q(a, s)$

4. **Observe** resulting states $s'$

5. **Exit** if $s'$ is goal, else go to 1 with $s := s'$

- Greedy policy based on $h$ can be written as $\pi_h(s) = \arg\min_{a \in A(s)} Q(a, s)$
- $\pi_h$ is **optimal** when $h = h^*$, otherwise non-optimal and may get trapped into loops
Real Time Dynamic Programming (RTDP)

Same as LRTA* but deals with true (probabilistic) MDP

1. **Evaluate** each action $a$ applicable in $s$ as

   $$Q(a, s) = c(a, s) + \sum_{s' \in S} P_a(s'|s)V_i(s')$$

2. **Apply** action $a$ that minimizes $Q(a, s)$

3. **Update** $V(s)$ to $Q(a, s)$

4. **Observe** resulting state $s'$

5. **Exit** if $s'$ is goal, else go to 1 with $s := s'$

$V(s)$ initialized to $h(s)$; if $h < V^*$, RTDP eventually **optimal**
Variations on RTDP: Reinforcement Learning

Q-learning is a model-free version of RTDP

1. Apply action $a$ that minimizes $Q(a, s)$ with probability $1 - \epsilon$, with probability $\epsilon$, choose $a$ randomly

2. Observe resulting state $s'$

3. Update $Q(a, s)$ to

   $$(1 - \alpha)Q(a, s) + \alpha[c(a, s) + \max_a Q(a, s')]$$

4. Exit if $s'$ is goal, else with $s := s'$ go to 1

Q-learning learns asymptotically to solve MDPs optimally (Watkins 89)
Bibliography

- Chris Watkins Peter Dayan. Q-learning, Machine Learning, 8, 279-292.