Planning with State Models, MDPs and POMDPs

Hector Geffner
Universidad Simón Bolívar
Caracas, Venezuela

The Control Problem

$$\textit{Goals} \longrightarrow \boxed{\textit{Agent}} \;\; \underset{\xleftarrow{\hspace{1em}}}{\overset{\textit{actions}}{\longrightarrow}} \;\; \boxed{\textit{World}}$$

$$\textit{observations}$$

Examples:

- a controller that has to schedule jobs
- mobile robot that has to navigate in building
- a program to coordinate a logistics effort
- $\vdots$

Questions:

- how to design ‘agent’
- how to select the ‘right’ actions

Approaches to the control problem in AI

- **Programming**

  write control by hand in suitable procedural language (e.g., Brooks' approach to mobile robots)

- **Planning**

  derive control from declarative description of actions (dynamics) and goals (e.g., Strips planning)

- **Learning**

  learn control from examples or trial and error (e.g., reinforcement learning, neural nets)

– Successes and problems in each approach

– We'll focus on **planning** approach

Planning

deriving control from declarative description of actions (dynamics), sensors, and goals

- Open-loop Planning

$$\begin{array}{c} \textit{Actions} \\ \textit{Goals} \end{array} \longrightarrow \boxed{\begin{array}{c} \textit{Open-loop} \\ \textit{Planner} \end{array}} \longrightarrow \textit{Plan} \overset{actions}{\longrightarrow} \textit{World}$$

- Closed-loop Planning

$$\begin{array}{c} \textit{Actions} \\ \textit{Goals} \end{array} \longrightarrow \boxed{\begin{array}{c} \textit{Closed-Loop} \\ \textit{Planner} \end{array}} \longrightarrow \textit{Plan*} \quad \begin{array}{c} \overset{actions}{\longrightarrow} \\ \overset{observations}{\longleftarrow} \end{array} \quad \textit{World}$$

Example: Simplified robot navigation

`Init:`	X_0, Y_0
`Goal:`	X_G, Y_G
`Actions:`	Up, $Down$, $Left$, $Right$
`Sensor:`	`Current` X, Y

- **Open-loop plan** U, U, R, R may solve problem but makes no use of sensor information
- **Closed-loop plan** more robust if actions don't work 100% as expected

`Up`	`if`	$Y < Y_G$
`Down`	`if`	$Y > Y_G$
`Right`	`if`	$X < X_G$
`Left`	`if`	$X > X_G$

- More interesting situations arise from presence of obstacles, stairs, initial uncertainty, unreliable sensors, ...

Example: Contingent Planning

- **Init:** two empty bowls and a large pile of eggs
- **Goal:** have three good eggs and no bad ones into bowl2
- **Actions:** `break` egg into bowl, `clean` bowl, `pour` one bowl into another, `inspect` bowl, . . .

Solution: closed-loop plan such as

grab *and* **break** *egg into bowl1*
inspect *bowl1*
if *bad,* **clean** *bowl1*
else pour *bowl1 into bowl2* $\star$
repeat until $\star$ *done* 3 *times*

General approach to planning

- **Mathematical Models** for making tasks precise
- **Representation Languages** for describing problems
- **Algorithms** for solving them

Models we will consider are:

- State Models

  determinism, complete information
- Markov Decision Processes (MDPs)

  non-determinism, full sensing
- Partially Observable MDPs (POMDPs)

  non-determinism, partial sensing

Illustration: Omelette Problem

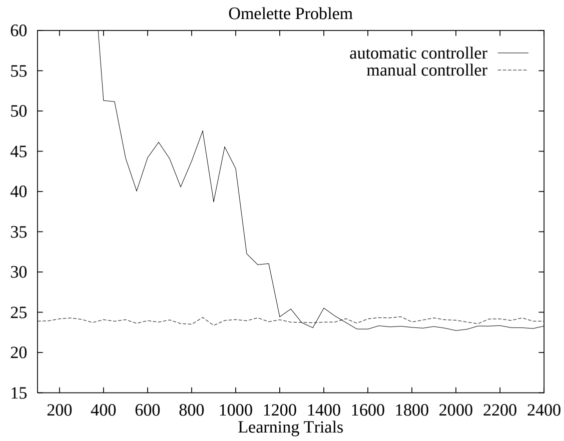
Action: **grab-egg**()
Precond: $\neg holding$
Effects: $holding :=$ **true**
$good? :=$ (**true** 0.5 ; **false** 0.5)

Action: **break-egg**($bowl : BOWL$)
Precond: $holding \wedge (ngood(bowl) + nbad(bowl)) < 4$
Effects: $holding :=$ **false**
$good? \ \rightarrow \ ngood(bowl) := ngood(bowl) + 1$
$\neg good? \ \rightarrow \ nbad(bowl) := nbad(bowl) + 1$

Action: **clean**(bowl:BOWL)
Precond: $\neg holding$
Effects: $ngood(bowl) := 0$, $nbad(bowl) := 0$

Action: **inspect**($bowl : BOWL$)
Effect: **obs**($nbad(bowl) > 0$)

Performance of controller obtained



Outline for rest of the talk

- Models
- Algorithms
- Languages
- Results

State Models

- State models are familiar in AI
- They are 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 $f(s, a)$ for $s \in S$ and $a \in A(s)$
  - action costs $c(a, s) > 0$
- A *solution* is a sequence of applicable actions that leads to goal
- A solution is *optimal* if it has minimum cost

Markov Decision Processes (MDPs)

- MDPs are 'small' departure from State models
- Action effects are **stochastic** and **fully observable**:
  - 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 (Puterman, Bertsekas, Sutton & Barto, . . .)
- Other MDP formulations: discounted, finite-horizon, . . .

Partially Observable MDPs (POMDPs)

- POMDPs generalize State Models and MDPs
- Action effects are **stochastic** and **partially observable**:
  - 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
  - **observations** o after action a with probabilities $P_a(o|s)$
- **Solutions** are policies that map belief states into actions (Sondik, Littman, Kaebling, Cassandra, . . .)
- **Optimal** policies minimize expected cost to go from b_0 to b_F

Beliefs in POMDPs

- Beliefs b are **probability distributions** over S
- Actions $A(b)$ applicable in belief state b are

$$A(b) = \{ \ a \mid a \in A(s) \text{ if } b(s) \neq 0 \ \}$$

- An action $a \in A(b)$ **deterministically** maps b into b_a

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s')$$

- In the **absence of feedback**, a POMDP is a **deterministic search problem** in belief space
- The task is to find a sequence of applicable actions that maps b_0 into target belief
- In the **presence of feedback** things are different ...

Beliefs in POMDPs (cont.)

- In the **presence of feedback**, beliefs are influenced by **observations** which cannot be predicted.
- As a result, evolution of beliefs cannot be predicted either
- ... yet probability of observing o after doing a in b is

$$b_a(o) = \sum_{s \in S} P_a(o|s) b_a(s)$$

- ... and new belief after observing o is

$$b_a^o(s) = P_a(o|s) b_a(s) / b_a(o)$$

- That is, action a maps belief b into belief b_a^o with probability $b_a(o)$
- A POMDP becomes an MDP over **belief space** (Astrom 1965)

Beliefs in Non-deterministic POMDPs

- When probabilities are **uniform** or actions and sensing is **non-deterministic**, belief states can be represented by **sets** of states
- The equations for the successor belief states b_a and b_a^o become simpler

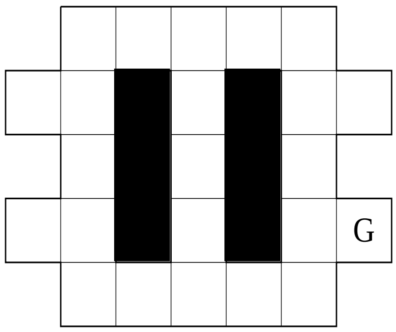
  e.g., if non-deterministic transition function $F(a, s)$ maps a state into a **set** of states, the equation for b_a becomes

  $$b_a = \{s' \mid s' \in F(a, s) \text{ and } s \in b\}$$

- **Model checking** techniques can be used for **representing** and **updating** beliefs efficiently
- For full POMDPs, modal checking techniques not common yet; yet see (Boutilier *et al.* UAI 99)

Examples: Robot Navigation as a POMDP
(Kaebling et al)

- **states:** $[x, y; \theta]$
- **actions** $rotate$ $+90$ and -90, $move$
- **costs:** uniform except when hitting walls
- **transitions:** e.g, $P_{move}([2,3;90] \mid [2,2;90]) = .7$, if $[2,3]$ is empty, ...
- **initial** b_0: e.g,, uniform over set of states
- **final** b_F: certain to have reached goal
- **observations:** presence or absence of wall with probabilities that depend on position of robot, walls, etc



Example: Sorting as a POMDP

(Bonet & Geffner, ICML'98)

- **Task:** Sort a vector of numbers of fixed size n
- **Actions:** $\textsc{cmp}(i,j)$ and $\textsc{swap}(i,j)$, $1 \leq i < j \leq n$
- **States:** the $n!$ vectors
- **Initial belief state** b_0 uniform over all states
- **Final belief state** b_F for which $b_F(s_G) = 1$, where $s_G[i] = i$, $i = 1, \ldots, n$, is the 'sorted' state
- **Observations** $i < j$ ($j < i$) result from $\textsc{cmp}(i,j)$ when $s[i] < s[j]$ ($s[i] > s[j]$)

A solution to the problem becomes a policy of **swaps** and **comparisons** that would take us from b_0 to b_F

Road Map

– Models $\surd$

– Algorithms

– Languages

State Models, MDPs and POMDPs
models are great, yet

How to solve them? $\longrightarrow$ **algorithms**

How to build them? $\longrightarrow$ **language**

Algorithms

1. **Heuristic search** algorithms
   - for solving State Models
2. **Dynamic programming** methods
   - for solving State Models and MDPs
3. Combinations and extensions of 1 + 2:
   - **Real Time Dynamic Programming** Methods
   - **Reinforcement Learning** Methods
4. We'll focus on a few methods; those that appear simpler and most effective

Heuristic Search: Greedy Algorithm

- **Greedy search** is a simple search algorithm

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 $Q(\mathbf{a}, s)$
3. **Exit** if s_a is goal, else go to 1 with $s := s_a$

- **Greedy Policy** is closed-loop version; in each state s it selects the action $\pi_h(s)$:

$$\pi_h(s) = \text{argmin}_{a \in A(s)} Q(a, s)$$

- Greedy policy is **optimal** when $h = h^*$; otherwise non-optimal and may even get trapped into loops

How to get h*: Dynamic Programming

- Optimal value function h^* is solution of Bellman equation

$$h^*(s) = \min_{a \in A(s)} [c(a,s) + h^*(s_a)]$$

- For MDPs, equation becomes

$$h^*(s) = \min_{a \in A(s)} [c(s,a) + \sum_{s' \in S} P_a(s'|s) h^*(s')]$$

- Bellman equation can be solved by **value iteration** method where estimates V of h^* are updated until convergence as

$$V(s) := \min_{a \in A(s)} [c(s,a) + \sum_{s' \in S} P_a(s'|s) V(s')]$$

- Yet more recent method that integrates **updates** with a **greedy search** tends to scale up better ...

Real Time Dynamic Programming (Barto et al 95)

- RTDP combines greedy search with DP updates

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 $Q(\mathbf{a}, s)$
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
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 is admissible, **after repeated trials**, greedy policy eventually becomes **optimal**
- if h is good, very large problems can be solved

RTDP for POMDPs (Bonet and Geffner 98)

Given that POMDPs are MDPs over belief states, algorithm for POMDPs becomes

1. **Evaluate** each action a applicable in b as
$$Q(a, b) = c(a, b) + \sum_{o \in O} b_a(o) V(b_a^o)$$
2. **Apply** action **a** that minimizes $Q(a, b)$
3. **Update** $V(b)$ to $Q(\mathbf{a}, b)$
4. **Observe** o
5. **Compute** new belief state b_a^o
6. **Exit** if b_a^o is a final belief state, else set b to b_a^o and go to 1

- Algorithm can easily be applied to non-deterministic POMDPs
- In probabilistic case, beliefs are discretized and asymptotic optimality not guaranteed

Road Map

- **Models:** State Models, MDPs, MDPs $\surd$
- **Algorithms:** RTDP + variations $\surd$
- **Languages:** ...
- **Results:** ...

Languages: High-Level Planning and Control

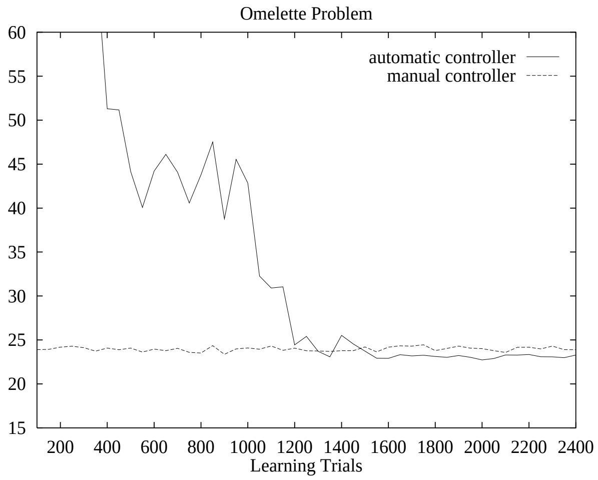
- Language of MDPs and POMDPs often not suitable for **modeling**
- **Logical** languages can be used to specify MDPs and POMDPs in a compact and modular way
- Strips and ADL are examples; yet more expressive languages needed for modeling **sensors** and **probabilities**
- We'll illustrate the language we have developed through examples
- A tool that we call GPT accepts problems expressed in this language, compiles them into MDPs/POMDPs, and computes resulting controller

Example: Omelette Problem (Levesque)

- Representation (incomplete):

Action:	**grab-egg**$()$
Precond:	$\neg holding$
Effects:	$holding :=$ **true**
	$good? := ($**true** 0.5 ; **false** $0.5)$
Action:	**clean**(bowl:BOWL)
Precond:	$\neg holding$
Effects:	$ngood(bowl) := 0$, $nbad(bowl) := 0$
Action:	**inspect**$(bowl : BOWL)$
Effect:	**obs**$(nbad(bowl) > 0)$

- Performance resulting controller (2000 trials in 192 sec)



Example: Information Gathering (Thrun)

- initial position is 6
- *goal* and *penalty* at either 0 or 4; which one not known
- noisy *map* at position 9

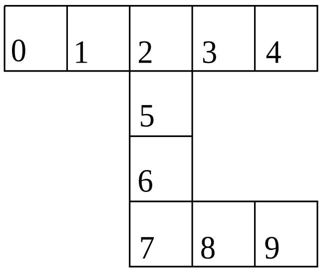
- Representation (incomplete)

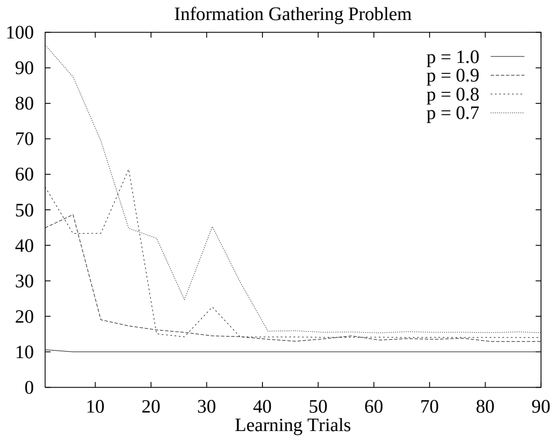
Action: **go-up**() ; same for down,left,right
Precond: $\text{FREE}(\text{UP}(pos))$
Effects: $pos := \text{UP}(pos)$

Action: $*$
Effects: $pos = pos9 \;\rightarrow\; \mathbf{obs}(ptr)$
$pos = goal \;\rightarrow\; \mathbf{obs}(goal)$
Costs: $pos = penalty \;\rightarrow\; 50.0$

Ramif: $\mathbf{true} \;\rightarrow\; ptr = (goal\ p\ ;\ penalty\ 1-p)$
Init: $pos = pos6\ ;\ goal = pos0\ \vee\ goal = pos4$
$penalty = pos0\ \vee\ penalty = pos4\ ;\ goal \neq penalty$
Goal: $pos = goal$

- Controller performance (60 trials/second)





Example: Key and Boxes (Collins and Pryor)

- Representation (incomplete)

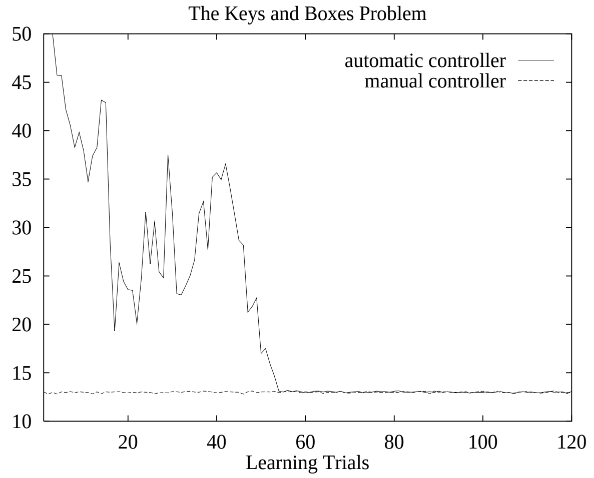
Action: **go-out**$()$
Precond: $position \neq outside$; $in(key, contents(atDoor))$
Effects: $position := outside$

Action: **go-in**$(p)(loc : LOC)$
Precond: $position = outside, \;\; loc \neq outside$, ...
Effects: $position := loc$

Action: **pickup**$(loc : LOC)$ similar for putdown
Precond: $position = loc, \;\; holding = nothing$
Effects: $holding := pick(contents(loc))$, **obs**$(holding = nothing)$
$contents(loc) := remove(holding^*, contents(loc))$

Ramif: $items(loc) = count(contents(loc))$
Init: $contents(table) :=$ **empty** ; $contents(atDoor) := \ldots$
Goal: $number\_red(contents(outside)) > 0$

- Performance of resulting controller (60 trials/5.61 seconds)



Summary

- A general **approach** to planning based on
  1. high-level action **languages**
  2. various state **models** (MDPs, POMDPs, ...)
  3. **algorithms** that combine ideas from heuristic search and dynamic programming
- Promising **results** over a number of tasks

  contingent planning, navigation, sorting, decision tree induction, ...
- A number of **challenges**
  - better lower bounds
  - scaling up
  - learning generalizations
  - optimality bounds
  - integration with other techniques (e.g., BDDs)
  - ⋮

To learn more

- Recent books by Sutton and Barto, and Bertsekas and Tsitsiklis, on MDPs and reinforcement learning

- Articles on Real Time Search by Korf, and on RTDP by Barto, Bradtke & Singh, both in AI Journal

- Articles by various authors: Kaebling, Russell, Littman, Boutilier, Koenig, and others
- Papers and software at `www.ldc.usb.ve/~hector`