## High-level Planning with Incomplete Information using POMDPs

**Héctor Geffner and Blai Bonet, USB**

- Approach to planning with incomplete information based on
  1. high-level descriptions of actions, situations and goals,
  2. partially observable markov decision models
  3. real-time dynamic programming algorithms

- Supported by a shell that accepts high-level descriptions of problems, producing suitable controllers

$$\begin{array}{c}\textbf{Action}\\ \textbf{Model}\\ \textbf{Goals}\end{array} \longrightarrow \textbf{Planner} \longrightarrow \textbf{Controller} \begin{array}{c}\xrightarrow{actions}\\ \xleftarrow{observations}\end{array} \textbf{World}$$

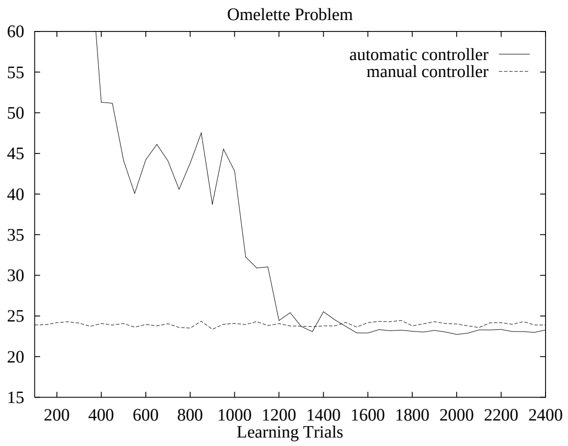
## Example 1: Omelette Problem (Levesque)

An agent has a large supply of eggs and its goal is to get three good eggs and no bad ones into one of two bowls. The eggs can be either good or bad, and at any time the agent can find out whether a bowl contains a bad egg by inspecting it. The problem is to devise a controller for achieving the goal.

- Formal representation:

| | |
|---|---|
| **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 resulting controller (2000 trials in 192 seconds)



## Partially Observable Markov Decision Problem (POMDPs)

POMDPs are models of sequential decision making that accommodate stochastic actions and noisy sensors:

- states $s \in S$
- actions $A(s) \subseteq A$ $s$
- 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 $P_a(o|s)$

- Belief states are probability distributions $b(s)$ over the real states $s$ of the environment
- An action $a$ maps a belief state $b$ into a unique belief state $b_a$ given by:

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s') \qquad (1)$$

- If in addition an observation $o$ is gathered, the belief is revised into $b_a^o$:

$$b_a^o(s) = P_a(o|s)b_a(s) / \sum_{s \in S} P_a(o|s)b_a(s) \qquad (2)$$

- A *solution* is a function (controller) that maps belief states into actions
- *Optimal* solutions minimize the expected costs from $b_0$ to $b_F$

## Algorithm: Real Time Dynamic Programming

- RTDP-BEL is a *hill-climbing* algorithm that from any state $b$ searches for the goal states $b_F$ using estimates $V(b)$ of the expected costs
- Main difference is that estimates $V(b)$ are updated dynamically
- Initial values, represented by $h(b)$, obtained by suitable relaxation of the problem (assuming full information after next move)
- Resulting controller is the policy that is greedy relative to these estimates

1. **Evaluate** each action $a$ in $b$ as

   $$Q(b, a) = c(b, a) + \sum_{o \in O} b_a(o) V(b_a^o)$$

   initializing $V(b_a^o)$ to $h(b_a^o)$
2. **Select** action **a** that minimizes $Q(b, a)$ breaking ties randomly
3. **Update** $V(b)$ to $Q(b, \mathbf{a})$
4. **Apply** action **a**
5. **Observe** $o$
6. **Compute** $b_a^o$
7. **Exit** if goal observed, else set $b$ to $b_a^o$ and go to 1

## Example 2: Assembly line (Draper, Hanks, Weld)

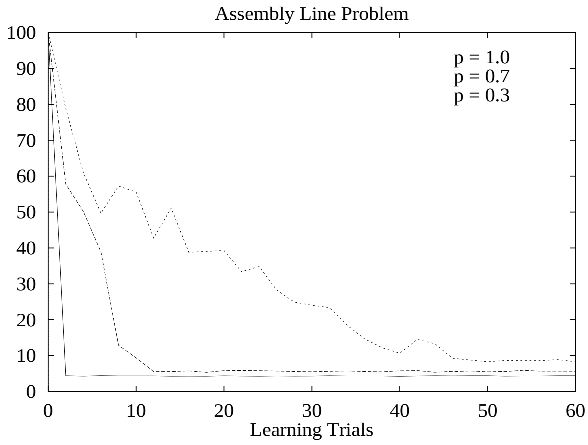
A robot has to decide whether to dispatch or reject pieces that come into an assembly line. Pieces are supposed to be dispatched or rejected according to whether they are believed to be flawed or not. Dispatched pieces are supposed to be painted first. The robot cannot observe directly whether a piece is flawed or not, but can sense whether the piece is blemished. It also knows that a flawed piece will appear as blemished if it's not painted.

- Formal Representation

| | |
|---|---|
| **Action:** | **paint**; |
| **Precond:** | $\neg painted$ |
| **Effects:** | $painted := \textbf{true}$ |
| **Action:** | **dispatch** |
| **Precond:** | $\neg processed$ |
| **Effects:** | $processed := \textbf{true}$ |
| **Costs:** | $\neg painted \;\rightarrow\; 50.0$ |
| | $flawed \;\rightarrow\; 50.0$ |
| **Action:** | **reject** |
| **Precond:** | $\neg processed$ |
| **Effects:** | $processed := \textbf{true}$ |
| **Costs:** | $\neg flawed \;\rightarrow\; 50.0$ |
| | **inspect** |
| **Effects:** | **obs**$(blemished)$ |

**Ramifications:**
$\neg painted \;\rightarrow\; blemished := flawed$
$painted \;\rightarrow\; blemished := \textbf{false}$
$\neg blemished \;\rightarrow\; b\_reading := \textbf{false}$
$blemished \;\rightarrow\; b\_reading := (\textbf{true}\ p\ ;\ \textbf{false}\ 1-p)$

- Performance of resulting controller for different levels $p$ of noise in sensor (100 trials/sec)



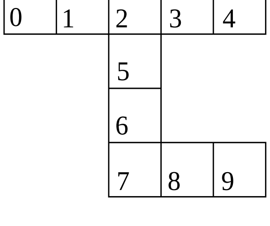
## Example 3: Information Gathering (Thrun)

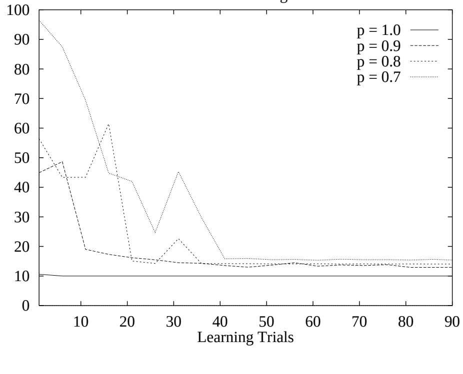
Agent starts in position $6$ and has to reach the *goal* which may be either at position $0$ or at position $4$. One of these two positions — the one that is not the goal — is a high penalty state. At position $9$ there is a 'map' that will report the true position of the goal with some probability.

- Formal Representation

| | |
|---|---|
| **Action:** | **go-up**() ; same for down,left,right |
| **Precond:** | $\text{FREE}(\text{UP}(pos))$ |
| **Effects:** | $pos := \text{UP}(pos)$ |
| **Action:** | $*$ |
| **Effects:** | $pos = pos9 \ \rightarrow\ \textbf{obs}(ptr)$ |
| | $pos = goal \ \rightarrow\ \textbf{obs}(goal)$ |
| **Costs:** | $pos = penalty \ \rightarrow\ 50.0$ |
| **Ramif:** | $\textbf{true} \ \rightarrow\ ptr = (goal\ p\ ;\ penalty\ 1-p)$ |
| **Init:** | $pos = pos6\ ;\ goal = pos0\ \vee\ goal = pos4$ |
| | $penalty = pos0\ \vee\ penalty = pos4\ ;\ goal \neq penalty$ |
| **Goal:** | $pos = goal$ |

- Map and Controller Performance for different levels of noise $p$ in the 'map' sensor (60 trials/second)



## Example 4: Key and Boxes (Collins and Pryor)

A robot is inside a room where there is a table, two boxes, a pile of red things by the door, and a key that may be in either of the two boxes. The goal is to get one red thing outside the room. In order to get out of the room the key must be placed by the door. The robot cannot hold two things at the same time, and while it knows whether it's holding something or not, it does not know *what* it is holding.

- Formal Representation (incomplete)

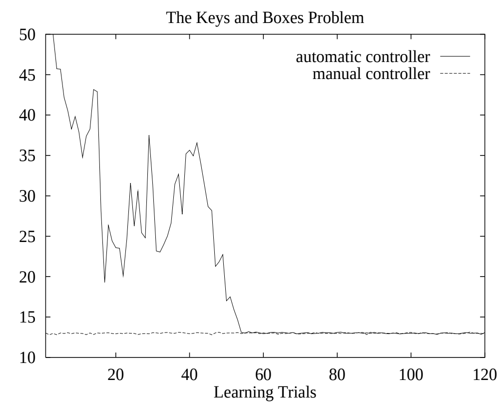
**Action:** **go-out**()
**Precond:** $position \neq outside$ ; $in(key, contents(atDoor))$
**Effects:** $position := outside$

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

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

**Ramif:** $items(loc) = count(contents(loc))$
**Init:** $contents(table) := \mathbf{empty}$ ; $contents(atDoor) := \{red1, red2\}, \ldots$
**Goal:** $number\_red(contents(outside)) > 0$

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



## Summary

- General, well-founded and (we believe) practical approach to planning with incomplete information given by
  1. POMDP models
  2. high-level action languages
  3. real time dynamic programming algorithms
- Empirical results over a number of tasks
  - contingent planning
  - sorting
  - decision tree induction
- A number of open challenges
  - generalization
  - optimality bounds
  - ⋮