# High-Level Planning and Control with Incomplete Information Using POMDP's

**Héctor Geffner*** and **Blai Bonet**
Departamento de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A, Venezuela
{hector,bonet}@usb.ve

**Abstract**

We develop an approach to planning with incomplete information that is based on three elements:

1. a high-level language for describing the effects of actions on both the world and the agent's beliefs that we call POMDP theories
2. a semantics that translates such theories into actual POMDPs
3. a real time dynamic programming algorithm that produces controllers from such POMDPs.

We show that the resulting approach is not only clean and general but that is *practical* as well. We have implemented a shell that accepts POMDP theories and produces controllers, and have tested it over a number of problems. In this paper we present the main elements of the approach and report results for the 'omelette problem' where the resulting controller exhibits a better performance than the handcrafted controller.

## Introduction

Consider an agent that has a large supply of eggs and whose 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 (Levesque 1996). The task is to devise a controller for achieving the goal. This is a typical problem of planning with incomplete information, a situation that is very common for agents that must act in the real world (Moore 1985). In AI there have been two approaches to this problem. On the one hand, proposals focused on extensions of Strips planning languages and algorithms; (e.g., (Etzioni *et al.* 1992; Collins & Pryor 1995)), on the other, proposals focused on formalizations of actions, knowledge and their interaction (e.g., (Moore 1985; Levesque 1996)). In general the two approaches haven't come together yet, and thus planners with incomplete information that both work and have a solid theoretical foundation are not common.

In this paper we attack this problem from a different perspective. We view problems of planning with incomplete information as Partially Observable Markov Decision Problems or POMDPs (Sondik 1971; Cassandra, Kaebling, & Littman 1994). POMDPs are very general mathematical models of sequential decisions problems that acommodate actions with uncertain effects and noisy sensors. The solutions of POMDPs are closed-loop controllers that map beliefs into actions.

While from a mathematical point of view, problems of planning with incomplete information *are* POMDPs, approaching these problems from the perspective of POMDPs can be *practical* only if:

1. high-level descriptions of the effects of actions on both the world and the agent's beliefs can be effectively translated into POMDPs, and
2. the resulting POMDPs can be effectively solved

Problem 2 has been a main impediment in the use of POMDPs in both Engineering and AI (Cassandra, Kaebling, & Littman 1995). Recently, however, new methods have been proposed, and some of the heuristic methods like the RTDP-BEL algorithm in (Geffner & Bonet 1998) have been shown to produce good solutions to large problems in a reasonable amount of time. Here we use the RTDP-BEL algorithm, which is a learning search algorithm (Korf 1990) based on the ideas of *real time dynamic programming* (Barto, Bradtke, & Singh 1995).

Problem 1 is also challenging as the language of POMDPs lacks the structure for representing interesting planning problems in a convenient way. We thus formulate a *high-level language* for suitably modeling complex planning problems with incomplete information, and a *semantics* for articulating the meaning of such language in terms of POMDPs. For this we make use of insights gained from the study of theories of actions and change (e.g., (Gelfond & Lifschitz 1993; Reiter 1991; Sandewal 1991)).

The result of this approach is a simple and general modeling framework for problems of planning with in-

Mailing address from US and Europe: Hector Geffner, Bamco CCS 144-00, P.O.BOX 02-5322, Miami Florida 33102-5322, USA.

complete information that we believe is also *practical.* We have actually implemented a *shell* that accepts suitable high level description of decision problems, compiles them into POMDPs and computes the resulting controller. We have tested this shell over a number of problems and have found that building the models and finding the solutions can be done in a very effective way. Here we report results for the 'omelette problem' (Levesque 1996) showing how the problem is modeled and solved.

The rest of the paper is organized as follows. We start with a brief overview of POMDPs and the algorithm used to solve them (Section 2). Then we introduce the language for expressing the effects of actions on the world (Section 3), and the extensions needed to express the effects of actions on the agent's beliefs (Section 4). Theories expressed in the resulting language determine unique POMDPs. The framework is then applied to the 'omelette problem' where suitable controllers are derived (Section 5).

## Background

POMDPs are a generalization of a model of sequential decision making formulated by Bellman in the 50's called *Markov Decision Processes* or MDPs, in which the state of the environment is assumed known (Bellman 1957). MDPs provide the basis for understanding POMDPs, thus we turn to them first. For lack of space we just consider the subclass of MDPs that we are going to use. For general treatments, see (Puterman 1994; Bertsekas & Tsitsiklis 1996); for an AI perspective, see (Boutilier, Dean, & Hanks 1995; Barto, Bradtke, & Singh 1995).

### MDPs

The type of MDPs that we consider is a simple generalization of the standard search model used in AI in which actions can have *probabilistic* effects. Goal MDPs, as we call them, are thus characterized by:

- a state space $S$
- initial and goal situations given by sets of states
- sets $A(s) \subseteq A$ of actions applicable in each state $s$
- costs $c(a, s)$ of performing action $a$ in $s$
- transition probabilities $P_a(s'|s)$ of ending up in state $s'$ after doing action $a \in A(s)$ in state $s$

Since the effects of actions are observable but not predictable, the solution of an MDP is not an action sequence (that would ignore observations) but a function mapping states $s$ into actions $a \in A(s)$. Such a function is called a *policy,* and its effect is to assign a probability to each state trajectory. The *expected cost* of a policy given an initial state is the weighted average of the costs of all the state trajectories starting in that state times their probability. The *optimal* policies minimize such expected costs from any state in the initial situation. In goal MDPs, goal states are assumed to be absorbing in the sense that they are terminal and involve zero costs. All other costs are assumed to be positive. General conditions for the existence of optimal policies and algorithms for finding them can be found in (Puterman 1994; Bertsekas & Tsitsiklis 1996).

### POMDPS

Partially Observable MDPs generalize MDPs allowing agents to have incomplete information about the state (Sondik 1971; Cassandra, Kaebling, & Littman 1994; Russell & Norvig 1994). Thus besides the sets of actions and states, the initial and goal situations, and the probability and cost functions, a POMDP also involves *prior beliefs* in the form of a probability distribution over $S$ and an *sensor model* in the form of a set $O$ of possible observations and probabilities $P_a(o|s)$ of observing $o \in O$ in state $s$ after having done the action $a$.

The techniques above are not directly applicable to POMDPs because while they do not presume that the agent can *predict* the next state, they do assume that he can *recognize* the next state once he gets there. In POMDPs this is no longer true, as the agent has to estimate the state probabilities from the information provided by the sensors.

The vector of probabilities $b(s)$ estimated for each of the states $s \in S$ at any one point is called the *belief* or *information state* of the agent. Interestingly, while the effects of actions on the states cannot be predicted, the effects of actions on *belief states* can. Indeed, the new belief state $b_a$ that results from having done action $a$ in the belief state $b$, and the new belief state $b_a^o$ that results from having done $a$ in $b$ and then having observed $o$ are given by the following equations (Cassandra, Kaebling, & Littman 1994):

$$
\begin{aligned}
b_a(s) &= \sum_{s' \in S} P_a(s|s') b(s') && (1) \\
b_a(o) &= \sum_{s \in S} P_a(o|s) b_a(s) && (2) \\
b_a^o(s) &= P_a(o|s) b_a(s) / b_a(o) \;\text{ if } b_a(o) \neq 0 && (3)
\end{aligned}
$$

As a result, the *incompletely observable* problem of going from an initial state to a goal state can be transformed into the *completely observable* problem of going from an *initial belief state* to a *final belief state* at a minimum expected cost. This problem corresponds to a goal MDP in which states are replaced by *belief states,* and the effects of actions are given by Equations 1–belaos. In such *belief* MDP, the *cost of an action* $a$ in $b$ is $c(a, b) = \sum_{s \in S} c(s, a) b(s)$, the set of *actions* $A(b)$ *available in* $b$ is the intersection of the sets $A(s)$ for $b(s) > 0$, and the *goal situation* is given by the belief states $b_F$ such that $b_F(s) = 0$ for all non-goal states $s$. Similarly, we assume a single *initial belief state* $b_0$ such that $b_0(s) = 0$ for all states $s$ not in the initial situation. Using the terminology of the logics of knowledge, these last three conditions mean that

1. the truth of action preconditions must be known
2. the achievement of the goal must be known as well
3. the set of feasible initial states is known

A solution to the resulting MDP is like the solution of any MDP: a policy mapping (belief) states $b$ into actions $a \in A(b)$. An optimal solution is a policy that given the initial belief state $b_0$ minimizes the expected cost to the goals $b_F$. The conditions under which such policies exist and the algorithms for finding them are complex because belief MDPs involve infinite state-spaces. Indeed the known methods for solving belief MDPs optimally (e.g., (Cassandra, Kaebling, & Littman 1994; Cassandra, Littman, & Zhang 1997)) can only solve very small instances (Cassandra, Kaebling, & Littman 1995). Recently, however, new methods have been proposed, and some of the heuristic methods like the RTDP-BEL algorithm in (Geffner & Bonet 1998) have been shown to produce good solutions to large problems in a reasonable amount of time. Here we use the RTDP-BEL algorithm, which is a learning search algorithm (Korf 1990) based on the ideas of *real time dynamic programming* (Barto, Bradtke, & Singh 1995).

RTDP-BEL is basically a *hill-climbing* algorithm that from any state $b$ searches for the goal states $b_F$ using estimates $V(b)$ of the expected costs (Figure 1). The main difference with standard hill-climbing is that estimates $V(b)$ are updated dynamically. Initially their value is set to $h(b)$, where $h$ is a suitable heuristic function, and every time the state $b$ is visited the value $V(b)$ is updated to make it consistent with the values of its successor states.

The heuristic function $h_{mdp}$ used in this paper is obtained as the optimal value function of a relaxed problem in which actions are assumed to yield complete information. If $V^*$ is the optimal value function of the underlying MDP, then

$$h_{mdp}(b) \stackrel{\text{def}}{=} \sum_{s \in S} V^*(s) \cdot b(s) \qquad (4)$$

For the implementation of RTDP-BEL, the estimates $V(b)$ are stored in a hash table that is initially empty. Then when the value $V(b')$ of a state $b'$ that is not in the table is needed, an entry $V(b') = h(b')$ is created. As in (Geffner & Bonet 1998), our implementation accepts an integer resolution parameter $r > 0$ such that the probabilities $b(s)$ are *discretized* into $r$ discrete levels before accessing the table. Best results have been obtained for values of $r$ in the interval $[10, 100]$. Higher values often generate too many entries in the table, while lower values often collapse the values of belief states that should be treated differently. In the experiments below we use $r = 20$.

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 $h(b_a^o)$ when $b_a^o$ not in table
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

Figure 1: RTDP-BEL

## POMDP theories

Most problems of planning with incomplete information can be modeled as POMDPs yet actually building the POMDP model for a particular application may be a very difficult task. For example, a simple 'blocks world' planning problem with 10 blocks involves more than a million states. Even if all actions are assumed deterministic and hence all transition probabilities are either zero or one, explicitly providing the states $s'$ such that $P_a(s'|s) \neq 0$ for each action $a$ and state $s$ is unfeasible. When the actions are probabilistic, the situation is even more complex.

This problem has been approached in AI through the use of convenient, high-level action description languages of which Strips (Fikes & Nilsson 1971) is the most common example. Since the 70's many extensions and variations of the Strips language have been developed and to a certain extent our language is no exception. Our POMDP *theories* differ from Strips mainly in their use of *functional* as opposed to *relational* fluents, and their ability to accommodate *probabilities*. On the other hand, POMDP theories have many features in common with logical theories of action (Gelfond & Lifschitz 1993; Reiter 1991; Sandewal 1991)), probabilistic extensions of Strips (Kushmerick, Hanks, & Weld 1995) and temporal extensions of Bayesian Networks (Dean & Kanazawa 1989; Russell & Norvig 1994). We go to the trouble of introducing another representation language because none of these languages is suitable for specifying rich POMDPs. For example, in none of these languages it is possible to express in a convenient way that the effect of an action is to increment the value of a certain variable with certain probability, or to make the value of certain term known. In the language below this is simple.

### State Language: Syntax

In order to express what is true in a state we appeal to a simplified first-order language that involves constant, function and predicate symbols but does not involve variables and quantification. We call this language the *state language* and denote it by L. All symbols in L have *types* and the way symbols get combined into *terms*, *atoms* and *formulas* is standard except that, as in any *strongly typed language*, the types of the symbols are taken into account. That is, if $f_{\alpha\beta}$ is a function symbol with type $\alpha\beta$, meaning that $f_{\alpha\beta}$ denotes a function that takes objects from a *domain* $D_\alpha$ and maps them into objects in the *domain* $D_\beta$, then $f_{\alpha\beta}(t)$ is a legal term when $t$ is a term of type $\alpha$. The type of $f_{\alpha\beta}(t)$ is $\beta$. Similarly, $p_\alpha(t)$ is an atom when $p$ is a predicate symbol of type $\alpha$ and $t$ is a term of the same type. *For simplicity, we assume in the presentation that the arity of function and predicate symbols is one unless otherwise stated.* All definitions carry to the general case by interpreting $t$, $\alpha$ and $D_\alpha$ as *tuples* of terms, types and domains. For uniformity we also treat constant symbols as function symbols of arity 0. So unless otherwise stated, terms of the form $f(t)$ include the constant symbols.

Types and domains can be either *primitive* or *defined.* When $\alpha$ is a primitive type, we assume that the domain of interpretation $D_\alpha$ is known. On the other hand, for non-primitive types $\beta$, the domain $D_\beta$ has to be specified. Such domains are specified by providing the unique *names* of the objects in $D_\beta$. It is thus assumed that such *defined domains* contain a finite number of objects, each with its own *name.* For example in a block planning scenario, the domain $D_{BLOCK}$ can be defined as the set $\{block_1, block_2, \ldots, block_n\}$.

Symbols are also divided into those that have a *fixed* and *known* denotation in all interpretations (e.g., symbols like '3', '+', '=', ..., and *names*) and those that don't. We call the first, *fixed symbols,*[1] and the second *fluent symbols.*[2] The fluent symbols are the symbols whose denotation (value) can be modified by the effect of actions and which persist otherwise. For the sake of simplicity, we assume that *all fluent symbols are function symbols.* Constant symbols like *temperature* can be captured by fluent symbols of 0-arity, while relational fluents can be captured by means of fluent symbols of type $\alpha\beta$ where $\beta$ is the boolean type.

Finally for reasons that will be apparent later we assume that all fluent symbols of arity greater than 0 take arguments of *defined* types only. This will guarantee that states can be *finitely* represented.

**Example 1** The first component of a POMDP theory are *the domain and type declarations* where all *defined* symbols, domains, and types are introduced. They are used to represent the objects of the target application, their attributes, their possible values, etc. For the 'omelette problem' the declarations are:

| | | |
|---|---|---|
| **Domain:** | $BOWL:$ | $small, large$ |
| **Types:** | $ngood:$ | $BOWL \mapsto Int$ |
| | $nbad:$ | $BOWL \mapsto Int$ |
| | $holding:$ | $Bool$ |
| | $good?:$ | $Bool$ |

meaning the there are two defined objects (bowls) with names *small* and *large*, and that each one has two associated integer attributes: the number of good eggs it contains and the number of bad eggs. In addition, there are two boolean features (function symbols of arity 0) representing whether the agent is holding an egg, and whether such an egg is good or not.

## State Language: Semantics

For a given POMDP theory, a *state* $s$ is a *logical interpretation* over the symbols in the theory in which each symbol $x$ of type $\alpha$ gets a denotation $x^s \in D_\alpha$. The denotation $t^s$ and $F^s$ of terms $t$ and formulas $F$ is obtained from the interpretations of the constant, function and predicate symbols in the standard way; e.g., $[f(t)]^s = f^s(t^s)$ for terms $f(t)$, etc.

**Variables and State Representation** Fixed symbols $x$ have a fixed denotation $x^*$ that is independent of $s$. For this reason, states $s$ can be represented entirely by the interpretation of the non-fixed symbols, which under the assumptions above, are the *fluent* symbols. Furthermore, since fluent symbols $f$ take arguments of defined types only in which each object has a unique name $id$ with a fixed denotation, $s$ can be represented by a finite table of entries of the form $f(id) \mapsto [f(id)]^s$. A useful way to understand the terms $f(id)$ for a fluent symbol $f$ and named arguments $id$ is as *state variables.* From this perspective, *the (representation of a) state is nothing else than an assignment of values to variables.* For example, the state of the theory that contains the declarations in Example 1 will be an assignment of integers to the four 'variables' $ngood(small)$, $nbad(small)$, $ngood(large)$, and $nbad(large)$, and of booleans to the two 'variables' *holding* and *good?*. The *state space* is the space of all such assignments to the six variables.

It's worth emphasizing the distinction between terms and variables: all variables are terms, but not all terms are variables. Otherwise, the expressive power of the language would be lost. For example, in a 'block' domain, '$loc(block_1)$' may be a term denoting the block (or table) on which $block_1$ is sitting, and '$clear(loc(block_1))$' may be a term representing the 'clear' status of such block (normally false, as $block_1$ is sitting on top). According to the definition above, the first term is a variable but the second term is not. The values of all terms, however, can be recovered from the values of the variables. Indeed, for any term $f(t)$, $[f(t)]^s = [f(id)]^s$ for the name $id$ of the object $t^s$.

## Transition Language: Syntax

While the state language allows us to say what is true in a particular state, the transition language allows us

[1] They are like the rigid designators in modal logic (Kripke 1971).

[2] From a computational point of view, the denotation of fixed symbols will be normally provided by the underlying programming language. On the other hand, the denotation of fluent symbols will result from the actions and rules in the theory.

to say how states change. This is specified by means of action descriptions.

An *action* is an expression of the form $p(id)$ where $p$ is an uninterpreted *action symbol* disjoint from the symbols in L, and $id$ is a (tuple) of name(s). Each action symbol has a type $\alpha$ which indicates the required type of its arguments $id$. For the 'omelette problem' for example, $pour(small, large)$ will be an action taking a pair of arguments of type $BOWL$.

The *action description* associated with an action $a$ specifies its costs, its preconditions, its effects on the world, and its effects on the agent's beliefs. In this section we focus on the syntax and semantics of the first three components.

The *preconditions* of an action $a$ are represented by a set of formulas $\mathrm{P}_a$, meaning that $a$ is applicable only in the states that satisfy the formulas in $\mathrm{P}_a$; i.e. $a \in A(s)$ iff $s$ satisfies $\mathrm{P}_a$.

The *costs* $c(a, s)$ associated with $a$ are represented by a sequence $\mathrm{C}_a$ of *rules* of the form $C \to w$, where $C$ is a state formula and $w$ is a positive real number. The cost $c(a, s)$ is the value of of the consequent of the first rule whose antecedent $C$ is true in $s$. If there is no such rule, as in the model below of the 'omelette problem', then $c(a, s)$ is assumed to be 1.

The *effects* of an action $a$ are specified by means of a *sequence* of action rules $\mathrm{R}_a$. *Deterministic* action rules have the form

$$C \to f(t) := t_1 \tag{5}$$

where $C$ is a formula, $f(t)$ and $t_1$ are terms of the same type and $f$ is a fluent symbol. The intuitive meaning of such rule, is that an effect of $a$ in states $s$ that satisfy the condition $C$ is *to set the variable $f(id)$ to $t_1^s$*, where $id$ is the name of the object $t^s$.

*Probabilistic action rules* differ from deterministic action rules in that the term $t_1$ in (5) is replaced by a finite list $L = (t_1\ p_1; t_2\ p_2; \ldots, t_n\ p_n)$ of terms $t_i$ and probabilities $p_i$ that add up to one. We call such a list a *lottery* and its type is the type of the terms $t_i$ which must all coincide. For a lottery $L$, the form of a probabilistic rule is:

$$C \to f(t) := L \tag{6}$$

where $C$ is a formula, $f(t)$ is a term of the same type as $L$, and $f$ is a fluent symbol. In principle, the meaning of such rule is that an effect of $a$ in states $s$ that satisfy the condition $C$ is *to set the* probability *of the variable $f(id)$ taking the value $t_i^s$ as $p_i$, where id is the name of the object $t^s$.* This interpretation, however, while basically correct misses the fact that two different terms $t_i$ and $t_j$ may have identical values $t_i^s$ and $t_j^s$, and hence that their probabilities must be added up. The precise meaning of probabilistic rules is provided below. For simplicity, since a deterministic rule $C \to f(t) := t_1$ can be expressed as a probabilistic rule of the form $C \to f(t) := (t_1\ 1)$, we'll take the former as an abbreviation of the latter. We also abbreviate rules $\mathbf{true} \to f(t) := \ldots$ as $f(t) := \ldots$.

As a further abbreviation, we often define the precondition and effects over *action schemas*. An action schema is an expression of the form $p(x)$, where $p$ is an action symbol and $x$ is a meta-variable. The preconditions and effects of action schemas can involve the meta-variable $x$ and other meta-variables as well. Such descriptions are abbreviations for the finite set of actions $a$, preconditions $\mathrm{P}_a$ and rules $\mathrm{R}_a$ that result from the action, precondition and effect schemas, by consistently replacing the meta-variables by all the names of the corresponding types.

**Example 2** As an illustration of the language, the effects of the action *move_up* that moves the blank up in the 8-puzzle can be written as:

$$p := up(p)\ ,\ tile(up(p)) := 0\ ,\ tile(p) := tile(up(p))$$

where $p$ tracks the position $1, \ldots, 9$ of the blank, $tile(i)$ tracks the identity of the tile at position $i$, and $up(i)$ is a *fixed* function symbol that denotes an *actual function* that given $i \in [0 \ldots 9]$ returns the position $j \in [0 \ldots 9]$ above it. Such function will normally be supplied by a program.

## Transition Language: Semantics

Let us define the probability distribution $P^{\mathsf{x}}_{s,L}$ induced by a lottery $L = (t_1\ p_1; \ldots; t_n\ p_n)$ on a variable $\mathsf{x}$ in state $s$ as:

$$P^{\mathsf{x}}_{s,L}(\mathsf{x} = v) \stackrel{\text{def}}{=} \sum_{i:t_i^s = v} p_i \quad \text{for } L = (t_i\ p_i)_{i=1,n} \tag{7}$$

The meaning of (6) can then be expressed as saying that the effect of the action $a$ in $s$ on the variable $\mathsf{x} = f(id)$ obtained by replacing $t$ by the name $id$ of $t^s$, is to set its probability to $P^{\mathsf{x}}_{s,L}$. More precisely, if we denote the probability of variable $\mathsf{x}$ in the states that follow the action $a$ in $s$ as $P^{\mathsf{x}}_{s,a}$, then *when (6) is the first rule in $R_a$ whose antecedent is true in $s$ and $\mathsf{x}$ is the name of $t^s$*

$$P^{\mathsf{x}}_{s,a}(\mathsf{x} = v) \stackrel{\text{def}}{=} P^{\mathsf{x}}_{s,L}(\mathsf{x} = v) \tag{8}$$

On the other hand, *when there is no rule in $R_a$ whose antecedent is true in $s$,* $\mathsf{x}$ persists:

$$P^{\mathsf{x}}_{s,a}(\mathsf{x} = v) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } v = \mathsf{x}^s \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

## Transition Probabilities

If $\mathsf{X}$ is the set of all the variables $\mathsf{x} = f(id)$ determined by the theory, then the transition probabilities $P_a(s'|s)$ for the POMDP are defined as:

$$P_a(s'|s) \stackrel{\text{def}}{=} \prod_{\mathsf{x} \in \mathsf{X}} P^{\mathsf{x}}_{s,a}(\mathsf{x} = \mathsf{x}^{s'}) \tag{10}$$

where the terms on the right hand side are defined in (8) and (9). This decomposition assumes that variables in $s'$ are mutually independent given the previous state $s$ and the action $a$ performed. This is a reasonable assumption in the absence of causal or ramification rules. For such extensions, see (Bonet & Geffner 1998).

**Example 3** Let us abbreviate the formulas $t = \textbf{true}$ and $t = \textbf{false}$ for terms $t$ of type boolean as $t$ and $\neg t$ respectively. Then the action descriptions for the 'omelette problem' can be written as:

| | |
|---|---|
| **Action:** | **grab-egg**() |
| **Precond:** | $\neg holding$ |
| **Effects:** | $holding := \textbf{true}$ |
| | $good? := (\textbf{true}\ 0.5\ ;\ \textbf{false}\ 0.5)$ |
| **Action:** | **break-egg**$(bowl : BOWL)$ |
| **Precond:** | $holding \wedge (ngood(bowl) + nbad(bowl)) < 4$ |
| **Effects:** | $holding := \textbf{false}$ |
| | $good? \rightarrow ngood(bowl) := ngood(bowl) + 1$ |
| | $\neg good? \rightarrow nbad(bowl) := nbad(bowl) + 1$ |
| **Action:** | **pour**$(b1 : BOWL, b2 : BOWL)$ |
| **Precond:** | $(b1 \neq b2) \wedge \neg holding$ |
| | $ngood(b1) + nbad(b1) + ngood(b2) + nbad(b2) < 4$ |
| **Effects:** | $ngood(b1) := 0\ ,\ nbad(b1) := 0$ |
| | $ngood(b2) := ngood(b2) + ngood(b1)$ |
| | $nbad(b2) := nbad(b2) + nbad(b1)$ |
| **Action:** | **clean**(bowl:BOWL) |
| **Precond:** | $\neg holding$ |
| **Effects:** | $ngood(bowl) := 0\ ,\ nbad(bowl) := 0$ |

There are no cost rules, thus, costs $c(a, s)$ are assumed to be 1. The description for the action *inspect* is given below.

## Initial and Goal Situations

In POMDP theories, the *initial* and *goal* situations are given by sets of formulas. The *effective* state-space of the POMDP is given by the set of states that satisfy the formulas in the initial situation or are reachable from them with some probability. The initial situation can contain constraints such that a block cannot sit on top of itself (e.g., $loc(block) \neq block$) or particular observations about the problem instance (e.g., $color(block_1) = color(block_2)$).

For the omelette problem, the initial and goal situations are:

$$
\begin{array}{ll}
\textbf{Init:} & ngood(small) = 0\ ;\ nbad(small) = 0 \\
 & ngood(large) = 0,;\ nbad(large) = 0 \\
\textbf{Goal:} & ngood(large) = 3\ ;\ nbad(large) = 0
\end{array}
$$

Note that for this problem the state-space is infinite while the *effective* state space is finite due to the constraints on the initial states and transitions (the preconditions preclude any bowl from containing more than 4 eggs). We impose this condition on all problems, and expect the *effective* state space to be always finite.

## Observations

The POMDP theories presented so far completely describe the underlying MDP. For this reason we call them MDP *theories.* In order to express POMDPs such theories need to be extended to characterize a prior belief state $P(s)$ and the observation model $P_a(o|s)$. For this extension, we make some simplifications:

1. We assume basically a *uniform prior distribution.* More precisely, the information about the initial situation $I$ is assumed to be known, and the prior belief state $b$ is defined as $b(s) = 0$ if $s$ does not satisfy $I$ and $b(s) = 1/n$ otherwise, where $n$ is the number of states that satisfy $I$ (that must be finite from our assumptions about the size of the effective state space).
2. We assume *no noise in the observations;* i.e., sensors may not provide the agent with complete information but whatever information they provide is accurate. Formally, $P_a(o|s)$ is either zero or one.

In addition to these simplifications, we add a *generalization* that is very convenient for modeling even if strictly speaking does not take us beyond the expressive power of POMDPs.

In POMDPs it assumed that there is a single variable, that we call $\mathsf{O}$, whose identity and domains are known a priori, and whose value $o$ at each time point is observed.[3] Although the values $o$ cannot be predicted in general, the variable $\mathsf{O}$ that is going to be observed *is* predictable and indeed it is always the same. We depart from this assumption and assume that *the set of expressions* $\mathsf{O}(s, a)$ *that are going to be observable in state* $s$ *after having done action* $a$ *is predictable but no fixed;* $\mathsf{O}(s, a)$ *will actually be a function of both* $s$ *and* $a$. Thus, for example, we will be able to say that if you do the action *lookaround* when you are near $door_1$, then the color of $door_1$ will be *observable,* and similarly, that whether the door is locked or not will be obsevable after moving the handle.

For this purpose, action descriptions are extended to include, besides their preconditions, effects and costs, a fourth component in the form of a set $K_a$ of *observation* or *knowledge gathering rules* (Scherl & Levesque 1993) of the form:

$$C \rightarrow \textbf{obs}(expr) \tag{11}$$

where $C$ is a formula, the expression *expr* is either a *symbol, a term or a formula,* and **obs** is a special symbol. The meaning of such rule is that the denotation (value) of *expr* will be be observable (known) in all states $s$ that satisfy the condition $C$ after having done the action $a$.

Thus a situation as the one described above can be modeled by including in $\mathrm{K}_a$ the *observation rule schema*

$$near(door) \rightarrow \textbf{obs}(color(door))$$

We use the notation $\mathsf{O}(s, a)$ to stand for *all* the expressions that are observable in $s$ after doing action $a$; i.e.,

$$\mathsf{O}(s, a) \stackrel{\text{def}}{=} \{x | C \rightarrow \textbf{obs}(x)\ \in \mathrm{K}_a\ \text{ and } C^s = \textbf{true}\} \tag{12}$$

The observations $o$ in the states $s$ after an action $a$ are thus the mappings that assign each expression $x \in \mathsf{O}(s, a)$ the denotation $x^o = x^s$. The probabilities $P_a(o|s')$ of the sensor model are then defined as 1

[3] In general, $\mathsf{O}$ can represent tuples of variables and $o$ corresponding tuples of values.

when $x^{s'} = x^o$ for all $x \in \mathsf{O}(s', a)$, and 0 otherwise. Clearly, when the observations provide only *partial* information about the state, many states can give rise to the same observation. That is, an agent that ends up in the state $s$ after doing an action $a$ may get and observation $o$ that won't allow him to distinguish $s$ from another state $s'$ if $P_a(o|s') = P_a(o|s)$.

**Example 4** The POMDP theory for the 'omelette problem' is completed by the following descriptions, where '$*$' stands for *all* actions:

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

**Action:** $*$
**Effect:** **obs**$(holding)$

Namely, *inspect* takes a *bowl* as argument and reveals whether it contains a bad egg or not, and *holding* is known after any action and state.

## Experiments

We have developed a shell that accepts POMDP theories, compiles them into POMDPs and solves them using the RTDP-BEL algorithm. We have modeled and solved a number of planning and control problems in this shell (Bonet & Geffner 1998) and here we focus on the results obtained for the 'omelette problem' (Levesque 1996) as described above. The theory is first compiled into a POMDP, an operation that is fast and takes a few seconds. The resulting POMDP has an *effective* state space of 356 states, 11 actions and 6 observations.

The curves in Fig. 2 show the *average cost to reach the goal* obtained by applying RTDP-BEL controllers to a number of simulations of the 'omelette world'. Action costs are all equal to 1 and thus the cost to the goal is the number of actions performed. The controller is the greedy policy that selects the best action according to the values stored in the table.[4]

We computed 10 runs of RTDP-BEL, each involving 2400 trials. In each run, we stopped RTDP-BEL after $i$ trials, for $i = 0, 50, 100, \ldots, 2400$, and applied the greedy controller with the values obtained at that point to 200 simulations of the 'omelette world'. Each point in the curve thus represents an average taken over 2000 simulations. A cutoff of 100 steps was used, meaning that trials were stopped after that number of steps. The cost to the goal in those trials was assumed to be 100. The belief resolution parameter $r$ used was 20, meaning that all probabilities were discretized into 20 discrete levels. The results are not sensitive to either the value of the cutoff or $r$ (although neither one should be made too small).

Figure 2(a) compares the performance of the RTDP-BEL controller vs. the handcrafted controller (Levesque 1996) for the case in which the probability of an egg being good is 0.5. The performance of the RTDP-BEL controller is poor over the first 1000 trials but improves until it converges after 1500 trials. The average time to compute 2000 trials is 192.4 seconds (around 3 minutes). At that point there were 2100 entries in the hash table on average. A reason for the poor performance of the algorithm over the first thousand trials is that the heuristic $h_{mdp}$ obtained from the underlying MDP (Section 2) assumes complete information about the next state and hence does not find the *inspect* action useful. Yet, *inspect* is a crucial action in this problem, and gradually, the algorithm 'learns' that. Interestingly, after the first 1500 trials the curve for the RTDP-BEL controller lies consistently below the curve for the handcrafted controller, meaning that its performance is better.[5] This difference in performance is actually more pronounced when the probability of an egg being good changes from 0.5 to a high value such as 0.85 (Fig. 2(b)). While in the first case, the difference in performance between the two controllers is 4%, in the second case, the difference rises to 14%.

## Summary and Discussion

We have formulated a theoretical approach for modeling and solving planning and control problems with incomplete information in which high level descriptions of actions are compiled into POMDPs and solved by a RTDP algorithm. We have also implemented a shell that supports this approach and given a POMDP theory produces a controller. We have shown how this approach applies to the 'omelette problem', a problem whose solution in more traditional approaches would involve the construction of a contingent plan with a loop. In (Bonet & Geffner 1998) this framework is extended to accommodate *ramification rules*, and variables that can take *sets* of values. The first extension allows the representation of noisy sensors and dependencies among variables; the second, the representation of the effects of actions like listing a directory.

While the ingredients that make this approach possible are well known, namely, POMDPs (Sondik 1971; Cassandra, Kaebling, & Littman 1994), RTDP algorithms (Barto, Bradtke, & Singh 1995) and action representation languages (Gelfond & Lifschitz 1993; Reiter 1991; Sandewal 1991), we are not aware of other approaches capable of modeling and solving these problems in an effective way. Some of the features that distinguish this framework from related *decision-theoretic approaches to planning* such as (Kushmerick, Hanks, & Weld 1995; Draper, Hanks, & Weld 1994; Boutilier, Dean, & Hanks 1995) are:

- a non-propositional action description language
- a language for observations that allows us to say

[4] Actually the controller is the RTDP-BEL algorithm in Fig. 1 *without the updates.*

[5] This is because the RTDP-BEL controller uses the *large* bowl as a 'buffer' when it's empty. In that way, half of the time it saves a step over the handcrafted controller.

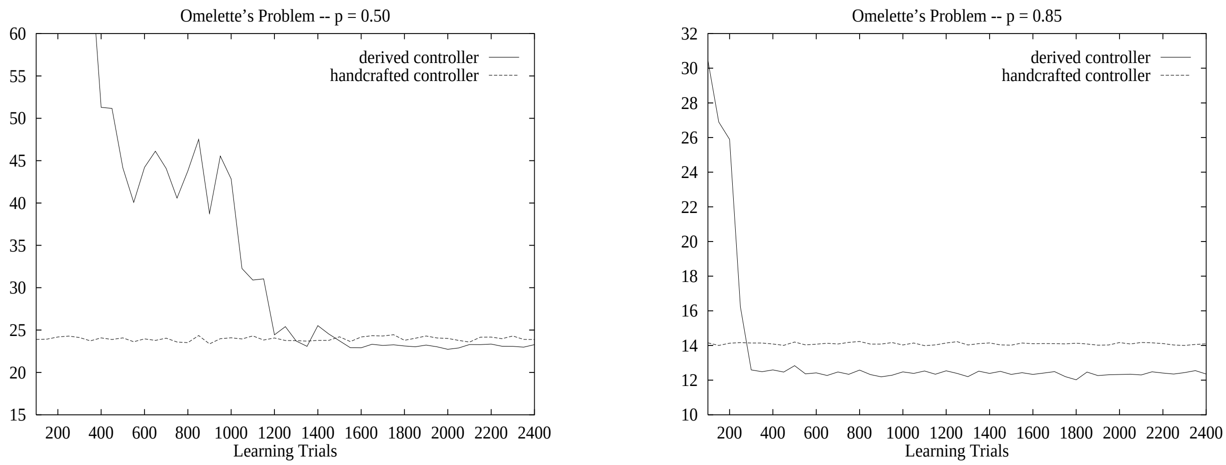


Figure 2: RTDP-BEL vs. Handcrafted Controller: $p = 0.5$ (a), $p = 0.85$ (b)

*which* expressions (either symbols, terms or formulas) are going to be observable and *when*,

- an effective algorithm that produces controllers for non-trivial problems with incomplete information

A weakness of this approach lies in the complexity of the RTDP-BEL algorithm that while being able to handle medium-sized problems well, does not always scale up to similar problems of bigger size. For instance, if the goal in the 'omelette problem' is changed to 50 good eggs in the large bowl in place of 3, the resulting model becomes intractable as the effective state space grows to more than $10^7$ states. This doesn't seem reasonable and it should be possible to avoid the combinatorial explosion in such cases. The ideas of finding concise representation of the *value* and *policy functions* are relevant to this problem (Boutilier, Dearden, & Goldszmidt 1995; Boutilier, Dean, & Hanks 1995), as well as some ideas we are working on that have to do with the representation of belief states and the mechanisms for belief updates.

## References


Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Bellman, R. 1957. *Dynamic Programming.* Princeton University Press.

Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming.* Athena Scientific.

Bonet, B., and Geffner, H. 1998. Planning and control with incomplete information using POMDPs: Experimental results. Submitted.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: structural assumptions and computational leverage. In *Proceedings of EWSP-95.*

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of IJCAI-95.*

Cassandra, A.; Kaebling, L.; and Littman, M. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings AAAI94*, 1023–1028.

Cassandra, A.; Kaebling, L.; and Littman, M. 1995. Learning policies for partially observable environments: Scaling up. In *Proc. of the 12th Int. Conf. on Machine Learning.*

Cassandra, A.; Littman, M.; and Zhang, N. 1997. Incremental pruning: A simple, fast, exact algorithm for Partially Observable Markov Decision Processes. In ??, ed., *Proceedings UAI-97*, ?? Morgan Kaufmann.

Collins, G., and Pryor, L. 1995. Planning under uncertainty: Some key issues. In *Proceedings IJCAI95.*

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second Int. Conference on Artificial Intelligence Planning Systems*, 31–36. AAAI Press.

Etzioni, O.; Hanks, S.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the Third Int. Conference on Principles of Knowledge Representation and Reasoning*, 115–125. Morgan Kaufmann.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 1:27–120.

Geffner, H., and Bonet, B. 1998. Solving large pomdp's using real time dynamic programming. Submitted.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *J. of Logic Programming* 17:301–322.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Kripke, S. 1971. Semantical considerations on modal logic. In Linsky, L., ed., *Reference and Modality.* Oxford University Press. 63–72.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239–286.

Levesque, H. 1996. What is planning in the presence of sensing. In *Proceedings AAAI-96*, 1139–1146. Portland, Oregon: MIT Press.

Moore, R. 1985. A formal theory of knowledge and action. In Hobbs, J., and Moore, R., eds., *Formal Theories of the Commonsense World.* Norwood, N.J.: Ablex Publishing Co.

Puterman, M. 1994. *Markov Decision Processes – Discrete Stochastic Dynamic Programming.* John Wiley and Sons, Inc.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy.* Academic Press. 359–380.

Russell, S., and Norvig, P. 1994. *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Sandewal, E. 1991. Features and fluents. Technical Report R-91-29, Linkoping University, Sweden.

Scherl, R., and Levesque, H. 1993. The frame problem and knowledge producing actions. In *Proceedings of AAAI-93*, 689–695. MIT Press.

Sondik, E. 1971. *The Optimal Control of Partially Observable Markov Processes.* Ph.D. Dissertation, Stanford University.