Planning with Incomplete Information as Heuristic Search in Belief Space

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

Abstract

The formulation of planning as heuristic search with heuristics derived from problem representations has turned out to be a fruitful approach for classical planning. In this paper, we pursue a similar idea in the context planning with incomplete information. Planning with incomplete information can be formulated as a problem of search in *belief space,* where *belief states* can be either *sets of states* or more generally *probability distribution over states.* While the formulation (as the formulation of classical planning as heuristic search) is not particularly novel, the contribution of this paper is to make it explicit, to test it over a number of domains, and to extend it to tasks like planning with sensing where the standard search algorithms do not apply. The resulting planner appears to be competitive with the most recent conformant and contingent planners (e.g., CGP, SGP, and CMBP) while at the same time is more general as it can handle probabilistic actions and sensing, different action costs, and epistemic goals.

Introduction

The formulation of classical planning as heuristic search has turned out to be a fruitful approach leading to powerful planners and a perspective on planning where the extraction of good heuristics is a key issue (McDermott 1996; Bonet, Loerincs, & Geffner 1997; Refanidis & Vlahavas 1999). In this paper, we take this idea into the domain of *planning with incomplete information.* Planning with incomplete information is distinguished from classical planning in the type and amount of information available at planning and execution time. In classical planning, the initial state is completely known, and no information is available from sensors. In planning with incomplete information, the initial state is not known, but sensor information may be available at execution time.

Conformant planning, a term coined in (Smith & Weld 1998), refers to planning with incomplete information but *no sensor feedback.* A *conformant plan* is a sequence of actions that achieves the goal from any initial state compatible with the available information. The problem has been addressed in (Smith & Weld 1998) with an algorithm based on the ideas of Graphplan (Blum & Furst 1995) that builds separate plan graphs for each possible initial state and searches all graphs simultaneously. The approach was tested on a number of problems and proven to scale better than conformant planners based on the ideas of partial-order planning (Kushmerick, Hanks, & Weld 1995; Peot 1998). Conformant and contingent planners based on variations of the SAT formulation (Kautz & Selman 1996) are reported in (Majercik & Littman 1998; Rintanen 1999).

From a mathematical point of view, the problem of conformant planning can be seen as the problem of finding a sequence of actions that will map an *initial belief state* into a *target belief state.* A belief state in this context is a *set of states:* the initial belief b_0 is the set of *possible* initial states, and the target beliefs are the sets that contain goal states only. Actions in this setting map one belief state into another.

Conformant plans can be found in this *belief space* by either blind or heuristic search methods. E.g., a breadth first search will generate first all belief states at a distance one from b_0, then all belief states at a distance two, and so on, until a target belief is found. We'll see that for many of the examples considered in the literature this approach actually works quite well. One reason for this is that the nodes in this type of search can be generated very fast without the overhead of more sophisticated searches. This is an important lesson from the recent planners such as Graphplan (Blum & Furst 1995), Blackbox (Kautz & Selman 1999), and HSP (Bonet & Geffner 1999): run-time operations have to be fast even if that requires a suitable amount of preprocessing.

Breadth-first search in belief space is the strategy used by the planners based on model-checking techniques (Cimatti & Roveri 1999; Giunchiglia & Traverso 1999). The model-checking part provides the data structures and algorithms for making the operations on beliefs (sets of states) more efficient. In our approach, the leverage does not come from the representation of beliefs but from the use of *heuristics.* We illustrate the benefits as well as the costs of using domain-independent heuristics over a number of problems.

In the presence of *sensor feedback,* the problem of planning with incomplete information is no longer a *deterministic* search problem in belief space. Since the *observations* cannot be predicted, the effect of actions over belief states becomes *non-deterministic,* and the selection of actions must be *conditional* on the observations gathered. We show that this problem can be formulated as a search problem in belief space as well, and that while standard heuristic search algorithms do not apply, a generalization of Korf's (1990) LRTA* algorithm due to Barto *et al.* (1995) does. The resulting planner appears competitive with the most recent contingent planners and applies with little modification to problems where actions and sensors are probabilistic. Such problems are known as POMDPs (Sondik 1971; Kaebling, Littman, & Cassandra 1998).[1]

The formulation of planning with incomplete information as heuristic search in *belief space* (as the formulation of classical planning as heuristic search in state space) is not particularly original, and follows from viewing these problems as variations of the general POMDP model (and see also (Genesereth & Nourbakhsh 1993)). The contribution of the paper is to make the formulation explicit, to test it over a number problems, and to extend it to tasks like contingent planning where the standard heuristic search algorithms do not apply. Preliminary results were presented in (Bonet & Geffner 1998a). Here we expand those results by considering probabilistic and non-probabilistic models, and a significantly larger set of experiments.

The paper is organized as follows. First we focus on the mathematical models underlying the various planning tasks (Sect. 2) and the algorithms and language needed for solving and expressing them (Sect. 3 and 4). Then we report results over a number of experiments (Sect. 5) and draw some conclusions (Sect. 6).

Mathematical Models

We consider first the models that make precise the semantics of the different planning tasks in terms of the type of the action dynamics, sensor feedback, and prior information available.

Classical Planning

Classical planning can be understood in terms of a state model defined over a discrete and finite state space, with an initial situation given by a single state s_0, a goal situation given by a non-empty set of states S_G, and a finite number of actions that deterministically map each state into another. This is a standard model in AI (Nilsson 1980) that for convenience we call the *deterministic control problem* model (Bertsekas 1995) and is characterized by

S1. A finite state space S,
S2. an initial situation given by a state $s_0 \in S$,
S3. a goal situation given by a non empty set $S_G \subseteq S$,
S4. actions $A(s) \subseteq A$ applicable in each state $s \in S$,
S5. a dynamics in which every action $a \in A(s)$ deterministically maps s into the state $s_a = f(a, s)$, and
S6. positive action costs $c(a, s)$ for doing a in s.

[1] POMDP stands for Partially Observable Markov Decision Process.

A *solution* to a deterministic control problem is a sequence of actions $a_0, a_1, \ldots, a_n$ that generates a state trajectory s_0, $s_1 = f(s_0)$, $\ldots$, $s_{n+1} = f(s_i, a_i)$ such that each action a_i is applicable in s_i and s_{n+1} is a goal state, i.e., $a_i \in A(s_i)$ and $s_{n+1} \in S_G$. The solution is *optimal* when the total cost $\sum_{i=0}^{n} c(s_i, a_i)$ is minimal. In planning, it's common to assume that all costs $c(a, s)$ are equal and focus on the plans with minimal *length.* While we don't need to make this assumption here, it will be convenient for simplicity to assume that actions costs do not depend on the state and thus can be written as $c(a)$. The generalization to state-dependent costs is straightforward.

Classical planning is a deterministic control problem that can be solved by *searching* the state-space S1–S5. This is the approach taken by heuristic search planners such as HSP (Bonet & Geffner 1999). Classical planning, however, can be formulated and solved in a number of other ways; e.g., as a SAT problem (Kautz & Selman 1996), as a Constraint Satisfaction Problem (Beek & Chen 1999), as an Integer Programming problem (Vossen *et al.* 1999), etc. In any case, regardless of the formulation chosen, the model S1–S5 provides a clear description of the semantics of the task; a description that is also useful when we move to *non-classical* planning.

Conformant Planning

Conformant planning introduces two changes: first, the initial state is no longer assumed to be known, and second, actions may be non-deterministic. The first change can be modeled by defining the initial situation as a *set* of states S_0, and the second by changing the deterministic transition function $f(a, s)$ in S5 into a non-deterministic function $F(a, s)$ that maps a and s into a non-empty *set* of states. Conformant planning can also be conveniently formulated as *deterministic* planning in *belief space* as we show below (see also (Genesereth & Nourbakhsh 1993)).

We use the term *belief state* to refer to *sets of states.*[2] A belief state b stands for the states that the agent executing the policy deems possible at one point. The initial belief state b_0 is given by the set S_0 of possible initial states, and if b expresses the belief state prior to performing action a, the belief state b_a describing the possible states in the next situation is

$$b_a = \{s \mid s \in F(a, s') \text{ and } s' \in b\} \qquad (1)$$

The set $A(b)$ of actions that can be safely applied in a belief state b are the actions a that can be applied in *any* state that is possible according to b:

$$A(b) = \{a \mid a \in A(s) \text{ for all } s \in b\} \qquad (2)$$

The task in conformant planning is to find a sequence of applicable actions that maps the initial belief state b_0 into a *final belief state* containing only *goal states.* That is, the set B_G of *target* beliefs is given by

$$B_G = \{b \mid \text{ such that for all } s \in b, s \in S_G\} \qquad (3)$$

[2] The terminology is borrowed from the logics of knowledge (Fagin *et al.* 1995) and POMDPs (Kaebling, Littman, & Cassandra 1998).

Provided with these definitions, the problem of conformant planning with deterministic and non-deterministic actions, can be formulated as a *deterministic control problem* over *belief space* given by

C1. The finite space B of *belief states* b over S,
C2. an initial situation given by a belief state $b_0 \in B$,
C3. a goal situation given by the target beliefs (3),
C4. actions $A(b) \subseteq A$ applicable in b given by (2),
C5. a dynamics in which every action $a \in A(b)$ deterministically maps b into b_a as in (1), and
C6. positive action costs $c(a)$.

The search methods that are used for solving deterministic control problems over *states* can then be used for solving deterministic control problems over *beliefs*.

The model above can be extended to express *epistemic goals* such as finding out whether a proposition p is true or not This requires a change in the definition (3) of the target beliefs. For example if the states are truth-valuations, the goal of knowing whether p is true can be expressed by defining the target beliefs as the sets of states in which the value of p is uniformly true, or uniformly false.[3]

Contingent Planning

In the presence of *sensor feedback* the model for conformant planning needs to be extended as actions may produce *observations* that affect the states of belief and the selection of the next actions.

Sensing can be modeled by assuming that every action a produces an observation $o(a, s)$ when the *real* state produced by the action is s.[4] This observation provides *information* about the state s but does not necessarily identify it uniquely as the observation $o(a, s)$ may be equal to the observation $o(a, s')$ for a state $s' \neq s$ (this is often called 'perceptual aliasing'; (Chrisman 1992)). On the other hand, upon gathering the observation $o = o(a, s)$, it is known that the *real* state of the environment is *not* s' if $o \neq o(a, s')$.

We call the function $o(\cdot, \cdot)$ the *sensor model.* This model is quite general for noise-free sensing and can be further generalized by making the observation $o(a, s)$ depend on the state s' in which the action a is taken. A generalization of this model is used for defining 'noisy' sensors in POMDPs (see below).

Prior to performing an action a in a belief state b, there is a set of possible observations that may result from the execution of a, as the observations depend on the (unobservable) state s that results. This set of possible observations, that we denote as $O(a, b)$, is given by

$$O(a, b) = \{o(a, s) \mid \text{ for } s \in b_a\} \tag{4}$$

[3]If there are actions that can change the value of p, then a 'dummy' proposition p' that does not change and is equal to p in the initial state, must be created. Then, the target beliefs should be defined in terms of p'. This 'trick' is needed because no explicit temporal information is kept in the model, so a way to find out the truth-value of p is to set p to a given value.

[4]Normally $o(a, s)$ will be a *collection* of primitive observations but this makes no difference in the formulation.

After a is done one of these observations o must obtain, allowing the agent to exclude from b_a the states that are not compatible with o. We call the resulting belief b_a^o:

$$b_a^o = \{s \in S \mid s \in b_a \text{ and } o = o(a, s)\} \tag{5}$$

Since the observation o that will be obtained cannot be predicted, the effect of actions on beliefs is *non-deterministic* and thus action a in b can lead to *any* belief b_a^o for $o \in O(a, b)$.

Contingent planning can thus be modeled as a *non-deterministic* control problem over belief space given by

T1. The finite space B of *belief states* b over S,
T2. an initial situation given by a belief state b_0,
T3. a goal situation given by the target beliefs (3),
T4. actions $A(b) \subseteq A$ applicable in b given by (2),
T5. a dynamics in which every action a non-deterministically maps b into b_a^o for $o \in O(a, b)$,
T6. positive action costs $c(a)$, and
T7. observations $o \in O(a, b)$ after doing a in b.

One way to characterize the *solutions* of this model is in terms of *graphs* where nodes b stand for beliefs, node labels $a(b)$ stand for the action in b, and every node b has as successors the nodes b_a^o for the beliefs that can result from $a(b)$ in b. The terminal nodes are the target beliefs. When the resulting graphs are *acyclic,* standard definitions can be used for characterizing the *solution* and *optimal solution* graphs, and heuristic search algorithms such as AO* (Nilsson 1980) can be used to search for them. However, often the graph is not acyclic as different paths may lead to the same beliefs.[5] A more general approach in that case can be obtained using the ideas of *dynamic programming* (Puterman 1994; Bertsekas 1995).

In a dynamic programming formulation, the focus is on the function $V^*(b)$ that expresses the *optimal cost* of reaching a target belief from any belief b. For the non-deterministic control problem given by T1–T7 and assuming that we are interested in finding 'plans' that *minimize worst possible cost,* this cost function is characterized by the following Bellman equation:[6]

$$V^*(b) = \min_{a \in A(b)} \left(c(a) + \max_{o \in O(a,b)} V^*(b_a^o) \right) \tag{6}$$

with $V^*(b) = 0$ for $b \in B_G$. Provided with this value function V^*, an optimal 'plan' can be obtained by selecting in each belief b the action $a = \pi^*(b)$ given by

$$\pi^*(b) = \operatorname*{argmin}_{a \in A(b)} \left(c(a) + \max_{o \in O(a,b)} V^*(b_a^o) \right) \tag{7}$$

[5]One way to get ride of cycles in the graph is by treating different occurrences of the same belief as different nodes. In that case, the decision graphs will be larger but algorithms such as AO* (Nilsson 1980) can be used. A related idea is to extend AO* for handling cyclic graphs. This has been proposed recently in (Hansen & Zilberstein 1998).

[6]For V^* to be well defined for all beliefs b, it is sufficient to assume a 'dummy' action with infinite cost that maps every state into a goal state.

The function π^*, called also a *policy,* constitutes the optimal solution of the contingent planning problem. The 'plan' can be obtained by unfolding π^*: first, the action $a = \pi^*(b)$ is taken for $b = b_0$, then upon getting the observation o, the action $a' = \pi^*(b_a^o)$ is taken, and so on. The graphs discussed above can be understood as the *representation* of the policy π^*. A policy, however, can also be represented by a list of condition-action rules. The formulation above makes no commitment about representations; it just describes the conditions that the optimal policy must obey. We will see that these conditions can be used for computing policies that use a tabular representation of V^*.

Probabilistic Contingent Planning

Problems of contingent planning where actions and sensors are *probabilistic* can be modeled as POMDPs with transition functions replaced by transition probabilities $P_a(s'|s)$ and the sensor model $o(a, s)$ replaced by sensor probabilities $P_a(o|s)$ (Astrom 1965; Sondik 1971; Kaebling, Littman, & Cassandra 1998). POMDPs are partially observable problems over state space, but like the models considered above, they can be formulated as *fully observable* problems over *belief space.* In such formulation, belief states are no longer *sets of states* but *probability distributions* over states. The probability that s is the real state given a belief b is expressed by $b(s)$. The effect of actions and observations on beliefs is captured by equations analogous to (1) and (5) above that are derived from the transition and sensor probabilities using Bayes' rule:

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s') \tag{8}$$

$$b_a^o(s) = P_a(o|s)b_a(s)/b_a(o) \text{ ; for } b_a(o) \neq 0 \tag{9}$$

Here $b_a(o)$ stands for the probability of observing o after doing action a in b and is given by

$$b_a(o) = \sum_{s \in S} P_a(o|s)b_a(s) \tag{10}$$

As before, the target beliefs are the ones that make the goal G certain, while the set $A(b)$ of applicable actions are those that are applicable in all the states that are possible according to b.

The problem of probabilistic contingent planning then becomes a *fully observable probabilistic control problem* over belief space given by

P1. The infinite space of belief states b that are *probability distributions* over S,
P2. an initial belief state b_0,
P3. a non-empty set of target beliefs,
P4. actions $A(b) \subseteq A$ applicable in each b,
P5. a dynamics where actions and observations map b into b_a^o with probability $b_a(o)$ given by (10),
P6. positive action costs $c(a, s)$, and
P7. observations o after doing action a in b with probability $b_a(o)$.

A 'plan' in this setting must map the initial belief b_0 into a target belief. The *optimal* 'plan' is the one that achieves this with minimum *expected* cost. Such plans can be formalized with a dynamic programming formulation similar to the one above (Sondik 1971; Kaebling, Littman, & Cassandra 1998). The Bellman equation for the optimal cost function V^* is

$$V^*(b) = \min_{a \in A(b)} \left(c(a) + \sum_{o \in O} V^*(b_a^o) b_a(o) \right) \tag{11}$$

with $V^*(b) = 0$ for $b \in B_G$, while the *optimal policy* π^* is

$$\pi^*(b) = \operatorname*{argmin}_{a \in A(b)} \left(c(a) + \sum_{o \in O} V^*(b_a^o)\ b_a(o) \right) \tag{12}$$

The computation of the optimal cost function and policy is more difficult than before because the belief-space in P1–P7 is infinite and continuous. As a result, only small problems can usually be solved to *optimality* (Kaebling, Littman, & Cassandra 1998). A common way to compute *approximate* solutions is by introducing a suitable discretization over the belief space (Lovejoy 1991; Hauskrecht 1997). Below we'll follow this approach.

Algorithms

We have shown that the problems of conformant and contingent planning can be formulated as *deterministic, non-deterministic,* and *probabilistic* control problems in a suitably defined belief space. These formulations are not particularly novel, and all can be considered as special cases of the belief-space formulation of POMDPs (Astrom 1965; Sondik 1971). Our goal in this paper is to use these formulations for solving planning problems and for comparing the results with the best available planners. In this section we turn to the *algorithms* for solving these models, and in the next section we discuss an *action language* for expressing them conveniently.

A*

The model C1–C6 for conformant planning can be solved by any standard search algorithm. In the experiments below we use the A* algorithm (Nilsson 1980) with two domain-independent heuristics. The first is the trivial heuristic $h = 0$. In that case, A* is just *uniform-cost search* or breadth-first search if all costs are equal. The second is an heuristic derived from the problem by a general transformation. Basically, we compute the optimal cost function V_{dp}^* over the states of the 'relaxed' problem where *full state observability* is assumed. Such function can be computed by solving the Bellman equation:

$$V_{dp}^*(s) = \min_{a \in A(s)} \left(c(a) + \max_{s' \in F(a,s)} V_{dp}^*(s') \right) \tag{13}$$

with $V_{dp}^*(s) = 0$ for $s \in S_G$, where S_G denotes the goal states and $F(a, s)$ denotes the set of states that may follow a in s. This computation is polynomial in $|S|$, and can be computed reasonably fast if $|S|$ is not too large (e.g., $|S| \leq 10^5$) (Puterman 1994; Bertsekas 1995).

With the function V^*_{dp} available, the heuristic $h_{dp}(b)$ for estimating the cost of reaching a target belief from *any* belief b is defined as

$$h_{dp}(b) \stackrel{\text{def}}{=} \max_{s \in b} V^*_{dp}(s) \tag{14}$$

It is simple to show that this heuristic is admissible and hence the solutions found by A* are guaranteed to be optimal (Nilsson 1980).

Greedy Policy

A* and the standard search algorithms do not apply to contingent planning problems where solutions are not sequences of actions. Algorithms like AO* (Nilsson 1980) can be applied to problems that do not involve cycles, and extensions of AO* for cyclic graphs have been recently proposed (Hansen & Zilberstein 1998). The benefit of these algorithms is they are optimal, the problem is that may need a long time and lot of memory for finding a solution. These limitations are even more pronounced among the optimal algorithms for POMDPs (e.g., (Kaebling, Littman, & Cassandra 1998)). We have thus been exploring the use of an anytime algorithm that can solve planning problems reasonably fast and can also improve with time. A convenient way for introducing such algorithm is as a variation of the simple *greedy policy.*

The greedy policy π_h takes an heuristic function h over belief states as input, and in each state b selects the action

$$\pi_h(b) = \operatorname*{argmin}_{a \in A(b)} \left(c(a) + \max_{o \in O} h(b^o_a) \right) \tag{15}$$

or

$$\pi_h(b) = \operatorname*{argmin}_{a \in A(b)} \left(c(a) + \sum_{o \in O} h(b^o_a) b_a(o) \right) \tag{16}$$

according to whether we are minimizing *worst* possible cost (non-deterministic contingent planning) or *expected* cost (probabilistic contingent planning). In both cases, if the heuristic function h is equal to the optimal cost function V^*, the greedy policy is optimal. Otherwise, it may not be optimal or may even fail to solve the problem.

Real Time Dynamic Programming

The problems with the greedy policy are two: it may lead to the goal through very long paths, or it may get trapped into loops and not lead to the goal at all. A simple modification due to Korf (1990) and generalized by Barto *et al* (1995) solves these two problems when the heuristic h is admissible and the space is finite. The resulting algorithm is called *real-time dynamic programming* as it combines a real-time (greedy) search with dynamic programming updates (see also (Bertsekas & Tsitsiklis 1996)).

The RTDP algorithm is obtained from the greedy policy by regarding the heuristic h as the initial estimate of a cost function V that is used to guide the search.

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

   $Q(a, b) = c(a) + \max_{o \in O} V(b^o_a)$ (non-det)
   $Q(a, b) = c(a) + \sum_{o \in O} b_a(o) V(b^o_a)$ (prob)

2. **Apply** action **a** that minimizes $Q(a, b)$ breaking ties randomly
3. **Update** $V(b)$ to $Q(\mathbf{a}, b)$
4. **Generate** observation o randomly from $O(a, b)$ (non-det), or with probability $b_a(o)$ (prob)
5. **Exit** if b^o_a is target belief, else set b to b^o_a and go to 1

Figure 1: RTDP over beliefs (probabilistic and non-deterministic versions)

Then, every time an action a is selected in b, the value of the cost function V for b is updated to

$$V(b) := \min_{a \in A(b)} \left(c(a) + \max_{o \in O} V(b^o_a) \right) \tag{17}$$

or

$$V(b) := \min_{a \in A(b)} \left(c(a) + \sum_{o \in O} h(b^o_a) b_a(o) \right) \tag{18}$$

according to whether we are minimizing *worst* possible cost (non-deterministic contingent planning) or *expected* cost (probabilistic contingent planning). The greedy policy π_V and the updates are then applied to a successor state b^o_a, and the cycle repeats until a target belief is reached. Since V is initially equal to h, the policy π_V behaves initially like the greedy policy π_h, yet the two policies get apart as a result of the updates on V.

When the belief space is finite, it follows from the results in (Korf 1990; Barto, Bradtke, & Singh 1995; Bertsekas & Tsitsiklis 1996) that RTDP will not be trapped into loops and will eventually reach the goal. This is what's called a single RTDP *trial*. In addition, after *consecutive trials*, the greedy policy π_V can be shown to eventually approach the optimal policy π^*. For this it is necessary that the heuristic h be admissible (non-overestimating). We note that the belief space in non-deterministic contingent planning is finite, while the belief space in probabilistic contingent planning can be made finite by a suitable discretization. In that case, the convergence RTDP does not guarantee the optimality of the resulting policy, but if the discretization is fine enough, the resulting policy will approach the optimal policy. The advantage of RTDP over other POMDP algorithms (e.g., (Lovejoy 1991), is that it can solve *fine discretizations* by using a suitable heuristic function for focusing the updates on the states that are most relevant.

In the experiments below, we use the h_{dp} heuristic defined above for non-deterministic problems, and a similar heuristic h_{mdp} for probabilistic problems. The heuristic h_{mdp} is obtained by solving a 'relaxed' problem similar to the one considered in Sect. 3.1 but with 'max' values replaced by expected values (Bonet & Geffner 1998b).

The RTDP algorithm is shown in Fig. 1. For the implementation of RTDP, the values $V(b)$ are stored in a hash table and when a value $V(b)$ that is not in table is needed, an entry for $V(b)$ set to $h(b)$ is allocated.

Language

We have considered a number of models and some algorithms for solving them. Planning problems however are not expressed in the language of these models but in suitable action languages such as Strips (Fikes & Nilsson 1971). The mapping of a *classical planning problem* expressed in Strips to the state model S1–S5 is straightforward: the states s are collection of atoms, the applicable actions $A(s)$ are the actions a for which $Prec(a) \subseteq s$, the state transition function is such that $f(a,s) = s - Del(a) + Add(a)$, etc. We have developed a language that extends Strips in a number of ways for expressing *all* the models considered in Sect. 2 in a compact form. The main extensions are

- function symbols, disjunction, and negation
- non-deterministic and probabilistic actions with conditional effects
- logical and probabilistic ramification rules
- observation-gathering rules
- cost rules

We have developed a planner that supports these extensions, and maps descriptions of conformant or contingent planning problems, with or without probabilities, into the corresponding models.[7] The models are then solved by the algorithms discussed in Sect. 3. The logical aspects of this language are presented in (Geffner 1999), while some of the other extensions are discussed in (Bonet & Geffner 1998a; Geffner & Wainer 1998). All the experiments reported below have been modeled and solved using this tool that for convenience we will call GPT.

Results

GPT accepts problem descriptions in a syntax based on PDDL (McDermott 1998) and converts these descriptions into C++ code. This translation together with the translation of C++ into native code takes in the order of 2 seconds. The experiments were run on a Sun Ultra with 128M RAM running at 333Mhz. We take a number of examples from (Smith & Weld 1998), (Cimatti & Roveri 1999), and (Weld, Anderson, & Smith 1998) where the conformant planners CGP and CMBP, and the contingent planner SGP are presented. CGP and SGP

problem		sequential				parallel	
name	$\|S\|$	$\|P\|$	CMBP	GPT(h)	GPT(0)	$\|L\|$	CGP
BT(2)	4	2	0.000	0.047	0.059	1	0.000
BT(4)	8	4	0.000	0.050	0.048	1	0.000
BT(6)	12	6	0.020	0.064	0.068	1	0.010
BT(8)	16	8	0.150	0.139	0.157	1	0.020
BT(10)	20	10	1.330	0.610	0.683	1	1.020
BTC(6)	24	11	0.160	0.064	0.087	11	0.860
BTC(7)	28	13	0.520	0.107	0.122	13	2.980
BTC(8)	32	15	1.850	0.179	0.186	15	13.690
BTC(9)	36	17	6.020	0.415	0.359	17	41.010
BTC(10)	40	19	16.020	0.796	0.765	19	157.590

Table 1: Results for BT and BTC problems

are parallel planners based on the ideas of Graphplan (Blum & Furst 1995), while CMBP is an optimal sequential planner based on model checking techniques. We take the results for CGP and CMBP from (Cimatti & Roveri 1999) where an extensive comparison is presented. Those results were obtained on a Pentium-II with 512M of RAM running at 300Mhz. CMBP is implemented in C while CGP and SGP are implemented in Lisp. We also include a number of problems of our own to illustrate the capabilities of our planner and contribute to the set of benchmarks used in the area.

Conformant Planning

We consider three types of conformant planning problems. The results are shown in Tables 1 to 3. The column $|S|$ refers to the size of the state space, while $|P|$ ($|L|$) refers to the length of the sequential (parallel) plans found. GPT solves these problems by using the A* algorithm. The column GPT(h) refers to the results obtained by running A* with the h_{dp} heuristic, while the column GPT(0) refers to the results with the heuristic $h = 0$. Long dashes (—) in the tables indicate that the planner exhausted memory or time (2 hours).

BT Problems. The first problems are variations of the 'bomb in the toilet' problem. Following (Cimatti & Roveri 1999), the problems are called BT(p), BTC(p), BTUC(p), and BMTC(p,t). BT(p) is the standard problem where the bomb can be in any of p packages and the bomb is disarmed by dunking it into the toilet. In parallel planners, this problem can be solved in one step by dunking all packages in parallel. BTC(p) is the sequential variation where dunking a package clogs the toilet and dunking does not disarm the bomb until the toilet is flushed. BTUC(p) is a non-deterministic variation where dunking may or may not clog the toilet. Finally, BMTC(p,t) involves p packages and t toilets. In the 'low uncertainty' case, the location of the bomb is not known and toilets are known to be not clogged; in the 'high uncertainty' case, none of these conditions are known. The results for these problems are in Tables 1, 2, and 5 (last page). GPT appears to scale better than CGP and CMBP in all problems except the BT(p) problems that are trivial for a parallel planner like CGP. The heuristic, however, does not help in these examples, but does not hurt either (the heuristic may hurt when it's expensive to compute and does not improve the search).

Navigation. The second class of problems SQUARE(n)

[7] The observation-gathering rules are all deterministic and cannot by themselves represent 'noisy' sensing. Noisy sensing is represented by the combination of observation-gathering rules and ramification rules; e.g., if action a makes the value of a variable x known with probability p, then we write that a makes the value of a 'dummy' variable y known with certainty, and use ramification rules to express x and y are equal with probability p. This is a general transformation: noisy sensing is mapped into noise-free sensing of a correlated variable.

name	$\|S\|$	$\|P\|$	CMBP	GPT(h)	GPT(0)
BTUC(6)	24	11	0.170	0.091	0.090
BTUC(7)	28	13	0.530	0.118	0.126
BTUC(8)	32	15	1.830	0.247	0.241
BTUC(9)	36	17	6.020	0.497	0.483
BTUC(10)	40	19	17.730	1.095	1.063

Table 2: Results for BTUC problems

name	$\|S\|$	$\|P\|$	GPT(h)	GPT(0)
SQUARE(12)	144	22	0.118	2.995
SQUARE(14)	196	26	0.159	7.103
SQUARE(16)	256	30	0.219	14.909
SQUARE(18)	324	34	0.290	29.580
SQUARE(20)	400	38	0.386	53.851
CUBE(6)	216	15	0.165	6.022
CUBE(7)	343	18	0.266	20.347
CUBE(8)	512	21	0.450	66.539
CUBE(9)	729	24	0.654	—
CUBE(10)	1000	27	0.991	
SORTN(3)	6	3	0.061	0.061
SORTN(4)	24	5	0.060	0.065
SORTN(5)	120	9	0.688	0.653
SORTN(6)	720	12	119.544	164.482
SORTN(7)	5040		—	—

Table 3: Results for SQUARE, CUBE, and SORTN problems

and CUBE(n) deals with a navigation problem in a square or cube with side n. The goal is to reach a given corner given that the initial location is completely unknown. There are 4 actions in SQUARE(n) and 6 actions in CUBE(n) that correspond to the possible directions. Moves against a boundary leave the agent in the same position. The optimal solution is given by $n-1$ movements along each axis in the direction of the goal. The worst possible cost of this plan is $2(n-1)$ for SQUARE(n) and $3(n-1)$ for CUBE(n). This is a problem taken from (Parr & Russell 1995). The results in Table 3 show that the heuristic h_{dp} makes a substantial difference in this case.

Sorting Networks. A sorting network refers to a sorting algorithm in which comparisons and swaps are merged into a single operation that takes two entries i and j and swaps them if and only if they are not ordered. A conformant plan is given by the sequence of pairs i and j on which to apply this operation. The number of states in the problem is given by the possible ways in which the entries can be ordered; this is $n!$ for SORTN(n). The optimal cost of these problems is known for small values of n only ($n \leq 8$ according to (Knuth 1973)). The heuristic does not help much in this type of problems, still both GPT(h) and GPT(0) find optimal solutions in a couple of minutes for n's up to 6 (Table 3).

Planning with Sensing

We consider now problems that involve sensing. Some of these problems are non-deterministic and others are probabilistic. The results were obtained with the probabilistic version of RTDP assuming uniform probability

Name	$\|S\|$	$V^*(b_0)$	Trial	$Avg(\|P\|)$	Time
MEDICAL(2)	20	3.000	20	3.000	0.720
MEDICAL(3)	32	4.333	25	4.333	1.173
MEDICAL(4)	36	5.000	25	5.000	1.315
MEDICAL(5)	20	4.600	25	4.600	1.759

Table 4: Results for MEDICAL problems

distributions for the non-deterministic problems.[8] The results for these problems are shown in Table 4, 6 and 7, and Fig. 4 (last page). RTDP is a stochastic algorithm that may produce different policies in different runs, and at the same time, in non-deterministic or probabilistic domains, the same policies may produce different results. We thus assess the performance of RTDP, by taking averages and standard deviations over many runs. The measures of interest that are displayed in the tables are

- the average cost to reach the goal in a given trial, denoted as $avg(|P|)$,
- the average time accumulated up to and including that trial,
- the success rate in that trial (percent of simulations in which the goal was reached within a given number of steps)[9], and
- the changing cost estimate $V(b_0)$ of the initial state.

The measure $V(b_0)$ is important because it's a lower bound on the *optimal* expected cost of the problem $V^*(b_0)$. Since $V(b_0) \leq V^*(b_0)$ and $V^*(b_0) \leq avg(|P|)$, $V(b_0) \approx avg(|P|)$ normally indicates convergence to the optimal policy. The subtlety though is that as we run RTDP on a discretized belief space[10] the updates do not guarantee that $V(b_0)$ remains always a lower bound on $V^*(b_0)$. Nonetheless, this is often true, and in the examples below this can be verified since the optimal expected costs $V^*(b_0)$ can be computed analytically.

BTCS Problems. The first set of problems is from (Weld, Anderson, & Smith 1998) and involve a sensing variant of the 'bomb in the clogged toilet' problems BTC(p) where there are a number of sensors for detecting whether a package contains the bomb. Weld *et al.* note that the time for SGP to solve these problems scales linearly with number of sensors, and for five packages and four sensors it is 0.5 seconds. The results for RTDP are in Table 6 (last page). For this problem, the optimal expected cost $V^*(b_0)$ is given by the formula $(p^2 + 3p - 2)/2p$ which for $p = 4, 6, 8$ results in 3.250, 4.333 and 5.375. As it can be seen from the table, these values are closely approximated by $avg(|P|)$ when $avg(|P|)$ and $V(b_0)$ converge. Also, the average costs of the policies derived in the *first* trial are never more

[8] Minimizing expected costs assuming uniform probabilities, however, is *not* equivalent to minimizing worst-possible cost. We use this formulation as it applies to all problems, probabilistic or not.

[9] For all the experiments, the cutoff used was 250; i.e., trials taking more than 250 steps to reach the goal were aborted and counted as failures.

[10] In the experiments, probability values are discretized into 10 intervals.

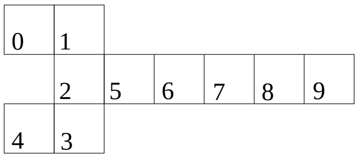

Figure 2: Map for grid problem

than 3 steps away from the optimal cost. The first trial takes less than 0.6 seconds in most cases. The curves for $avg(|P|)$ and $V(b_0)$ as a function of the number of trials for $p = 8$, is shown in Fig. 4 (last page).

Medical. The MEDICAL(n) problem is also from (Weld, Anderson, & Smith 1998) This is a problem that involves a patient that can be heathly or may have n diseases. The medication cures the patient if he has the right disease but kills the patient otherwise. SGP solves these problems for $n = 2, 3, 4, 5$ in .020, .040, .230, and 2.6 seconds. The results for RTDP are in Table 4. RTDP takes more time in the smaller problems but scales more smoothly and solves the larger problems faster. The optimal policy is derived in less than 30 trials and 1.75 seconds. The success rate for MEDICAL(n) and BTCS(n) is 100% from the first trial.

Grid. This is a navigation problem over the grid showed in Fig. 2 due to S. Thrun. An agent starts in position 6 in the grid and has to reach a *goal* that is at position 0 or 4. The position that is not the goal is a high penalty state. At position 9 there is a sensor that reports the true position of the goal with probability p. When $p = 1$, the optimal solution is to go to position 9, 'read' the sensor once, and head up for the goal. When $p < 1$, the agent has to stay longer in 9 accumulating information from the sensor before heading for the goal. Successive readings of the sensor are assumed to be independent. Fig. 3 shows the average cost to the goal as a function of the number of trials for $p = 0.75$. Table 7 shows results for this and other values of p. For $p = 0.75$ the average cost of the policy obtained after a single trial is 35.675 while after 400 trials is 17.265. In all cases convergence is achieved in less than 2 seconds. Also all trials reach the goal.

Omelette. The final problem is from (Levesque 1996) and was modeled and solved in (Bonet & Geffner 1998a). The goal is to have 3 good eggs and no bad ones in one of two bowls. There is a large pile of eggs, and eggs can be grabbed and broken into a bowl, while contents of a bowl can be discared or passed to the other bowl, etc. There is also a sensing action for testing whether a bowl contains a bad egg or not. We assume that sensing is noise-free and that eggs are good with a probability p equal to 0.25, 0.5 or 0.75. The results for this problem are shown in Table 7 and Fig. 4 (both in last page). As it can be seen from the table, the success rate during the first iteration in these problems is very low. For $p = 0.25$ only 4.6% of the first trials reach the goal. However, after a sufficient number of trials, a success rate of 100% is achieved in all cases, with an average cost that corresponds to the policy that gets a good egg in the target bowl first, and then uses the

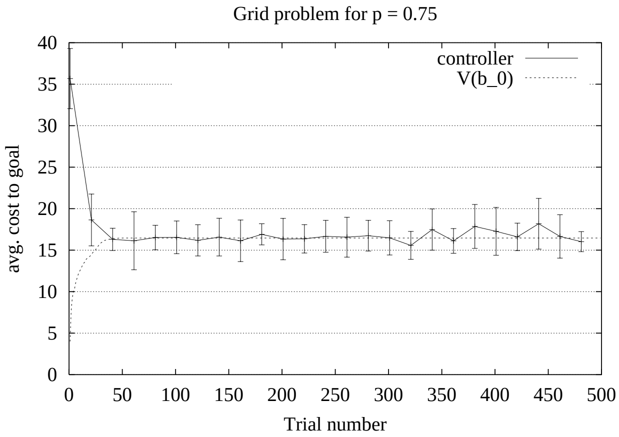

Figure 3: Curve for an instance of GRID problem.

other bowl as the buffer where eggs are inspected and passed to the target bowl if good, and are discarded otherwise. It can be shown that this policy has an expected cost given by the expression $11 + 12(1 - p)/p$ which for $p = 0.75$, 0.5 and 0.25 yields 15, 23, and 47 steps respectively. These values are closely approximated by the asymptotic values of $avg(|P|)$ and $V(b_0)$ in Table 7 in less than 76.33 seconds.

Discussion

We have shown that planning with incomplete information can be formulated as a problem of heuristic search in belief space. When there is no sensor feedback, the plans can be obtained by standard search algorithms such as A* and the results are competitive with the best conformant planners. In the presence of sensor feedback, the standard search algorithms do not apply, but algorithms like RTDP, that combines heuristic search with dynamic programming updates, can be used and yield competitive results as well. An additional benefit of this approach is that it is quite flexible as it can accommodate probabilistic actions and sensing, actions of different costs, and epistemic goals. The limitations, on the other hand, are mainly two. First, RTDP is not an *optimal* search algorithm like A*; it's guaranteed to yield optimal policies only asymptotically, and if the (belief) space is finite. In non-finite spaces such as those arising from probabilistic beliefs, this is not guaranteed. The second limitation is that the complexity of a number of preprocessing and runtime operations in GPT scale with the size of the state space. So if the state space is sufficiently large, our approach does not even get off the ground. In spite of these limitations, the approach appears to offer a performance and a flexibility that few other approaches currently provide. In the near future we would like to explore the issues that must be addressed for modeling and solving a number of challenging problems such as Mastermind, Minesweeper, the Counterfeit Coin problem, and others. Many of these problems are purely information gathering problems for which the heuristics we have considered are useless. Other general heuristics, however, can be devised. Indeed, if all actions are purely information gathering actions, and none produces more than $|o|$ observations, the cost of finding the true state s in an initial belief state b can be bounded by the function $\log_{|o|}(|b|)$. This function is an admissible heuristic that can be used for solving a wide range of *state-identification* problems like the Counterfeit Coin problem in (Pearl 1983).

Acknowledgments

This work has been partially supported by grant S1-96001365 from Conicit, Venezuela and by the Wallenberg Foundation, Sweden. Blai Bonet is currently at UCLA with an USB-Conicit fellowship.

References

Astrom, K. 1965. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.* 10:174–205.

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

Beek, P. V., and Chen, X. 1999. CPlan: a constraint programming approach to planning. In *Proc. AAAI-99*.

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

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Vols 1 and 2*. Athena Scientific.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*.

Bonet, B., and Geffner, H. 1998a. High-level planning and control with incomplete information using POMDPs. In *Proceedings AAAI Fall Symp. on Cognitive Robotics*.

Bonet, B., and Geffner, H. 1998b. Solving large POMDPs using real time dynamic programming. In *Proc. AAAI Fall Symp. on POMDPs*.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of ECP-99*. Springer.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, 714–719. MIT Press.

Chrisman, L. 1992. Reinforcement learning with perceptual aliasing. In *Proceedings AAAI-92*.

Cimatti, A., and Roveri, M. 1999. Conformant planning via model checking. In *Proceedings of ECP-99*. Springer.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning about Knowledge*. MIT Press.

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 Wainer, J. 1998. Modeling action, knowledge and control. In *Proceedings ECAI-98*. Wiley.

Geffner, H. 1999. Functional strips: a more general language for planning and problem solving. Logic-based AI Workshop, Washington D.C.

Genesereth, M., and Nourbakhsh, I. 1993. Time-Saving tips for problem solving with incomplete information. In *Proceedings of AAAI-93*.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *Proceedings of ECP-99*. Springer.

Hansen, E., and Zilberstein, S. 1998. Heuristic search in cyclic AND/OR graphs. In *Proc. AAAI-98*, 412–418.

Hauskrecht, M. 1997. *Planning and Control in Stochastic Domains with Incomplete Information*. Ph.D. Dissertation, MIT.

Kaebling, L.; Littman, M.; and Cassandra, T. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99–134.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, 1194–1201.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. In *Proceedings IJCAI-99*.

Knuth, D. 1973. *The Art of Computer Programming, Vol. III: Sorting and Searching*. Addison-Wesley.

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

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.

Lovejoy, W. 1991. Computationally feasible bounds for partially observed markov decision processes. *Operations Research* 162–175.

Majercik, S., and Littman, M. 1998. Maxplan: A new approach to probabilistic planning. In *Proc. AIPS-98*, 86–93.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*.

McDermott, D. 1998. PDDL – the planning domain definition language. Available at `http://ftp.cs.yale.edu/pub/mcdermott`.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Tioga.

Parr, R., and Russell, S. 1995. Approximating optimal policies for partially observable stochastic domains. In *Proceedings IJCAI-95*.

Pearl, J. 1983. *Heuristics*. Morgan Kaufmann.

Peot, M. 1998. *Decision-Theoretic Planning*. Ph.D. Dissertation, Dept. Engineering-Economic Systems, Stanford University.

Puterman, M. 1994. *Markov Decision Processes: Discrete Dynamic Stochastic Programming*. John Wiley.

Refanidis, I., and Vlahavas, I. 1999. GRT: A domain independent heuristic for Strips worlds based on greedy regression tables. In *Proceedings of ECP-99*. Springer.

Rintanen, J. 1999. Constructing conditional plans by a theorem prover. *J. of AI Research* 10:323–352.

Smith, D., and Weld, D. 1998. Conformant graphplan. In *Proceedings AAAI-98*, 889–896. AAAI Press.

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

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *Proceedings IJCAI-99*.

Weld, D.; Anderson, C.; and Smith, D. 1998. Extending Graphplan to handle uncertainty and sensing actions. In *Proc. AAAI-98*, 897–904. AAAI Press.

problem		low uncertainty						high uncertainty			
name	$\|S\|$	$\|P\|$	CMBP	GPT(h)	GPT(0)	$\|L\|$	CGP	$\|P\|$	CMBP	GPT(h)	GPT(0)
BMTC(7,2)	56	12	2.100	0.266	0.284	7	508.510	14	3.390	0.500	0.513
BMTC(8,2)	64	14	7.960	0.590	0.589	7	918.960	16	12.330	1.138	1.139
BMTC(9,2)	72	16	22.826	1.326	1.268		—	18	35.510	2.614	2.654
BMTC(10,2)	80	18	72.730	2.975	2.964			20	121.740	6.095	6.061
BMTC(7,4)	224	10	14.210	1.847	1.920	3	2.410	14	40.410	14.669	15.420
BMTC(8,4)	256	12	77.420	4.648	4.707	3	8.540	16	932.820	38.720	39.773
BMTC(9,4)	288	14	—	11.715	11.695		—	18	—	98.067	99.745
BMTC(10,4)	320	16		29.999	30.158			20		240.010	243.976
BMTC(5,6)	640	5	3.080	1.653	2.120	1	0.060	10	40.770	41.917	50.285
BMTC(6,6)	768	6	17.490	4.606	5.315	1	0.100	12	1819.520	136.818	151.250
BMTC(7,6)	896	8	5939.520	13.447	14.379	3	211.720	14	—	401.947	435.495
BMTC(8,6)	1024	10	—	37.868	39.392	3	1015.160			—	—
BMTC(9,6)	1152	12		106.995	110.629	3	3051.990				
BMTC(10,6)	1280	14		241.494	246.984		—				

Table 5: Results for BMTC problems with low and high uncertainty

		1 sense action			2 sense actions			4 sense actions		
name	trial	$V(b_0)$	$Avg(\|P\|)$	acc. time	$V(b_0)$	$Avg(\|P\|)$	acc. time	$V(b_0)$	$Avg(\|P\|)$	acc. time
BTCS(4)	1	3.225	$4.312 \pm .218$	0.497	3.250	$4.182 \pm .124$	0.486	3.250	$4.098 \pm .151$	0.502
	401	3.250	$3.278 \pm .071$	1.324	3.250	$3.306 \pm .101$	1.483	3.250	$3.258 \pm .110$	2.120
BTCS(6)	1	3.333	$6.570 \pm .381$	0.494	3.333	$6.440 \pm .440$	0.512	3.333	$6.166 \pm .465$	0.523
	1001	4.333	$4.312 \pm .262$	4.435	4.333	$4.386 \pm .123$	6.303	4.333	$4.326 \pm .220$	11.657
BTCS(8)	1	3.500	$8.017 \pm .283$	0.512	3.500	$7.888 \pm .317$	0.542	3.500	$7.458 \pm .300$	0.587
	5901	5.375	$5.298 \pm .223$	49.550	5.375	$5.307 \pm .163$	85.710	5.375	$5.468 \pm .173$	161.476

Table 6: Results for BTCS problems.

name	trial	$V(b_0)$	$Avg(\|P\|)$	%succ.	acc. time
GRID(1.00)	1	$10.000 \pm .000$	13.650 ± 0.268	$100 \pm .000$	0.342
	401	$10.000 \pm .000$	10.000 ± 0.000	$100 \pm .000$	1.145
GRID(0.75)	1	$14.319 \pm .322$	35.675 ± 3.629	$100 \pm .000$	0.298
	401	$16.464 \pm .000$	17.265 ± 2.882	$100 \pm .000$	1.633
GRID(0.50)	1	$30.500 \pm .000$	32.230 ± 5.766	$100 \pm .000$	0.315
	401	$30.500 \pm .000$	28.910 ± 6.021	$100 \pm .000$	0.937
OMELETTE(0.25)	1	$14.277 \pm .250$	3.133 ± 1.371	$4.6 \pm .017$	0.623
	4001	$46.999 \pm .000$	46.072 ± 1.575	$100 \pm .000$	62.202
OMELETTE(0.50)	1	$12.945 \pm .049$	9.435 ± 1.578	$30.8 \pm .042$	0.622
	4001	$23.000 \pm .000$	22.732 ± 0.700	$100 \pm .000$	29.951
OMELETTE(0.75)	1	$12.059 \pm .045$	13.097 ± 0.419	$79.7 \pm .005$	0.621
	18401	$14.999 \pm .000$	15.017 ± 0.285	$100 \pm .000$	76.337

Table 7: Results for GRID and OMELETTE problems.

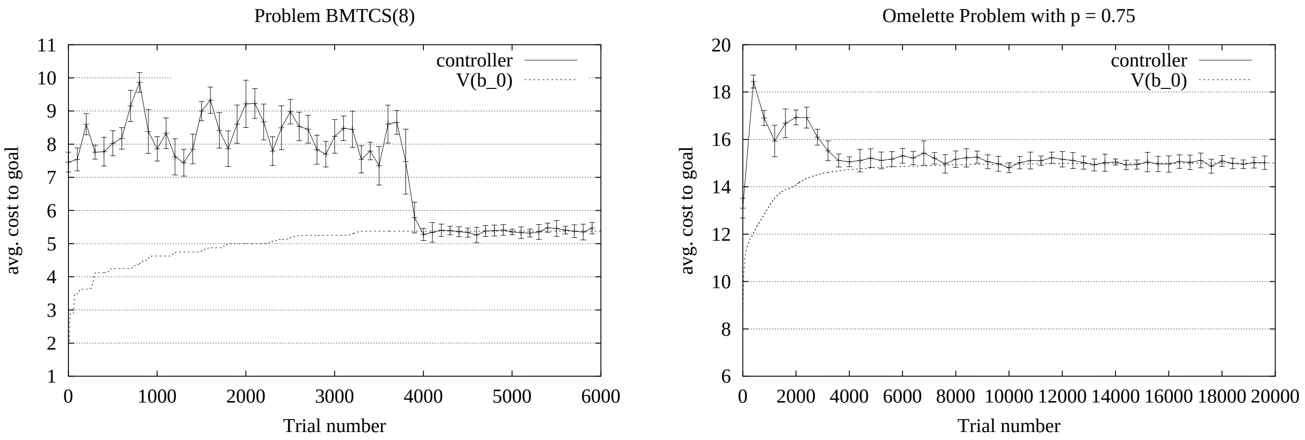

Figure 4: Curves for instances of BTCS and OMELETTE problems.