PC-SHOP: a Probabilistic-Conditional Hierarchical Task Planner

Abdelbaki BOUGUERRA Lars KARLSSON

SOMMARIO/*ABSTRACT*

In this paper we report on the extension of the classical HTN planner SHOP to plan in partially observable domains with uncertainty. Our algorithm PC-SHOP uses belief states to handle situations involving incomplete and uncertain information about the state of the world. Sensing and acting are integrated in the primitive actions through the use of a stochastic model. PC-SHOP is showed to scale up well compared to some of the state-of-the-art planners. We outline the main characteristics of the algorithm, and present performance results on some problems found in the literature.

Keywords: Planning, uncertainty, HTN

1 Introduction

In recent years, there has been a growing interest in planning under uncertainty; that is planning that takes into account incomplete information about the real world and the non-determinism of actions [7, 19, 1].

Classical planners assume that actions change the state of the world deterministically. Those planners can be considered as rigid, because they only generate sequences of actions to be executed in an open-loop process. However, in many real world environments, the classical approach would not work, due to the presence of exogenous events and other disturbances, and the fact that planners do not have immediate access to all the relevant information. Therefore more robust planners are needed to handle the uncertainty and complexity of these environments.

Such planners can be classified [**?**] according to whether their actions are deterministic, nondeterministic (i.e. an action can have several alternative outcomes) or

Developing more robust planners, has involved the relaxation of classical planning assumptions, leading to different approaches to handle uncertainty. Pure conditional planners assume that, at execution time, parts of the state of the world are observable, and construct plans that have the structure of an *alternative* "if-then-else" [19, 17]. Probabilistic conditional planners assign probabilities to the outcomes of actions, and search for plans with a probability of success that exceeds a certain threshold [7, 16, 10].

In this paper, we propose an algorithm that deals with partial observability at execution time, where observations reveal only a part of the information about the state of the world. Our planner PC-SHOP "Probabilistic Conditional SHOP", follows the success of the classical HTN planner SHOP [13, 14], and introduces mechanisms to extend SHOP to handle uncertainty and sensing. HTN planning is one of the oldest and most well-tried approaches to planning [18, 20, 6]. It allows one to code domain-dependent knowledge in a powerful way through procedures that describe how to solve the planning problem, resulting in more efficient search and better support for large domains. SHOP is known to be a simple but high-performing HTN planner, where tasks are planned for in the same order that they will be executed. The main additions of PC-SHOP to SHOP are:

- handling of partial observability.
- actions may have probabilistic effects.
- belief states model uncertainty about the state of the world.

These additions allow PC-SHOP to generate plans that are no longer just sequences of actions, but plans that have conditional branches. The objective of developing PC-SHOP is to take advantage of the efficient hierarchical planning techniques to deal with the inherent complexity of planning in uncertain domains. We should also mention that PC-SHOP has been applied to solve actual robotic problems such as recovery from ambiguous cases in perceptual anchoring with the same results as those reported in [5].

The rest of the paper is organized as follows. In the next section we outline the different entities used to represent planning domains. The planning algorithm is then presented followed by experimental results over some domains. Before concluding with some remarks, we give an overview of some of related approaches dealing with uncertainty.

2 Representation

HTN planning specifies tasks to achieve goals. Tasks can be primitive, corresponding to executable actions, or abstract (compound) if they need to be decomposed into other tasks as specified by methods. To describe a planning domain in PC-SHOP, fluents, actions, and methods are used.

The underlying representation for actions and belief states uses a rich LISP-style syntax language that supports among other structures: conditionals, probabilistic effects, quantified formulas, and partial state description. We do not give a detailed description of the language in this paper due to space limitations.

Throughout this paper, we use the tiger scenario from [9] to illustrate the representation of PC-SHOP entities: there are two doors, one to the left (ld) and one to the right (rd). Behind one of them lurks a tiger, and behind the other there is a reward. The aim is to devise a plan that allows an agent to collect the reward.

2.1 Ground States

The state of the world is described in terms of fluents (state variables) and their values. A fluent literal has the form $(f t_1 ... t_n{=}v)$, denoting that the fluent f with term parameters $t_1, ..., t_n$ has the value v (which might be non boolean). If the value is omitted, it is assumed to be t (true). The terms t_i can be constants or functional, where other non-boolean fluents may serve as functions. Examples of fluent literals are (room r1) and (robot-in = r1). A fluent formula is a logical combination of fluent literals using the standard connectives and quantifiers. Besides fluents that are stored directly in states, it is also possible to define axioms in terms of fluents (using fluent formulae and/or special forms for computing the value of the axiom). Fluents evaluated by external function calls are also supported.

2.2 Belief States

To model uncertainty about the state of the world at a certain point in time we use belief states. Formally, a belief state bs is a triple $\langle S_{bs}, P_{bs}, O_{bs}\rangle$ where S_{bs} is a set of ground states s, P_{bs} is a probability distribution over $s \in S_{bs}$, and O_{bs} is a set of observations the agent can make at run-time. The global probability $p(bs)$ of being in a belief state bs is defined as the sum of the probabilities of its element states i.e. $p(bs) = \sum_{s \in S_{bs}} P_{bs}(s)$.

Example 1. The initial belief state bs_0 for the tiger scenario has no observations $O_{bs_0} = \phi$, but contains two ground states $S_{bs_0} = \{s_0, s_1\}$. In state s_0 the tiger is behind the left door, and in state s_1 the tiger being is behind the right door. The agent can assume being in either state is equally probable i.e. $P_{bs_0}(s_0) = P_{bs_0}(s_1) = 0.5$.

2.3 Actions

PC-SHOP uses action schemas to model primitive tasks with conditional and probabilistic effects and information gathering during execution. An action schema consists of a tuple $\langle act, result\rangle$ where act is the action name, and $result$ is a result description expressed as an assertion formula that specifies the probabilistic and conditional effects of the action.

Example 2. To determine behind which door the tiger lurks, the agent can listen at the doors. Unfortunately, there is a 15% chance of error. This is an action that yields observations but no concrete effects. The fluent (tiger = *d*) stands for that the tiger is behind door *d*.

```
act:   (listen)
res:   (case
         ((tiger = ld)
          (alt (0.85 (obs (tiger = ld)))
               (0.15 (obs (tiger = rd)))))
         ((tiger = rd)
          (alt (0.85 (obs (tiger = rd)))
               (0.15 (obs (tiger = ld))))))
```

Where the obs form denotes that the observation of the fluent literal has been made, and the alt form is used for alternative effects, each with an associated probability (they should sum up to 1.0). The case form is for conditional results with semantics similar to the macro cond in Lisp.

2.4 Results of Actions

The application of an instantiated action A in a belief state bs is computed in two stages:

Stage1: the result description $result$ of A is applied in all ground states $s \in S_{bs}$ leading to the creation of a set of temporary belief states $\{bs'\}$, where each $S_{bs'}$ contains one new ground state corresponding to one alternative outcome of $result$ in s. The probability distribution $P_{bs'}$ is calculated as follows $P_{bs'}(s') = P_{bs}(s) \cdot P_a$ where P_a is the probability of the alternative outcome which produced s' from s. If the outcome includes making observations then $O_{bs'}$ receives those observations, otherwise $O_{bs'} = \phi$.

Stage2: The final belief states are created by grouping the temporary bs' with the same set of observations i.e. for all bs' with the same set of observations, create a new belief state bs_{new} where $O_{bs_{new}} = O_{bs'}$, $S_{bs_{new}} = \bigcup S_{bs'}$, and $P_{bs_{new}}(s') = P_{bs'}(s')$.

Example 3. Applying (listen) to the initial belief state bs_0 of example 1 results in two new belief states bs_{00} and bs_{01}. The process is shown in figure 1, where the subscripts of belief states components are omitted.

Note how the resulting states are partitioned into belief states based on their observations.

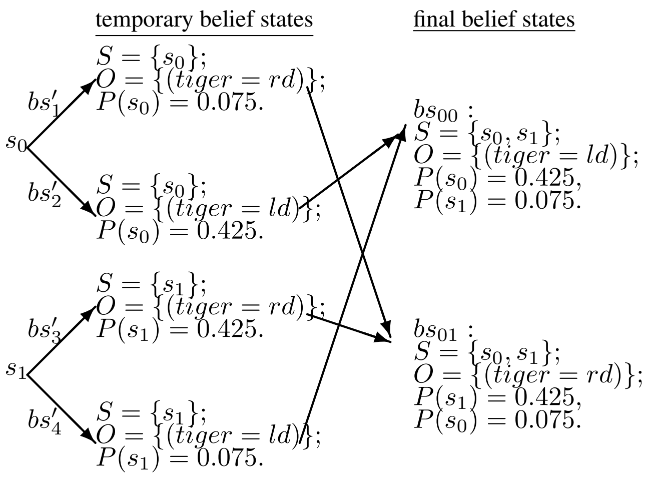
temporary belief states
final belief states
s_0
s_1
bs'_1
bs'_2
bs'_3
bs'_4
$S = \{s_0\}$; $O = \{(tiger = rd)\}$; $P(s_0) = 0.075$.
$S = \{s_0\}$; $O = \{(tiger = ld)\}$; $P(s_0) = 0.425$.
$S = \{s_1\}$; $O = \{(tiger = rd)\}$; $P(s_1) = 0.425$.
$S = \{s_1\}$; $O = \{(tiger = ld)\}$; $P(s_1) = 0.075$.
bs_{00} : $S = \{s_0, s_1\}$; $O = \{(tiger = ld)\}$; $P(s_0) = 0.425$, $P(s_1) = 0.075$.
bs_{01} : $S = \{s_0, s_1\}$; $O = \{(tiger = rd)\}$; $P(s_1) = 0.425$, $P(s_0) = 0.075$.

Figure 1: Results of `(listen)` in the initial belief state

2.5 Methods

In SHOP, methods are used to control search and they provide the knowledge of how to decompose abstract tasks. PC-SHOP uses the same syntax as SHOP to code methods. A method has the following form:

$(h\, p_1\, t_1\, p_2\, t_2 \ldots p_n\, t_n)$

where, *h* is the method's head and should unify with an abstract task. p_i is a list of precondition formulae. Each precondition formula has the form `(` `(`*var**`)` *fluent-literal fluent-formula** `)` * `)`, and t_i is an ordered list of tasks that specifies the reduction of h. Note that new variables not in h may be bound in p_i. The keyword `:first` can be inserted in the beginning of a precondition, to signify that only the first variable binding found should be used. That is the first binding of the variables of that precondition which satisfies the entire precondition list.

Semantically, a method specifies that the abstract task *h* is further decomposed to the tasks in t_k if the precondition p_k holds and all $p_{j<k}$ are false in the axiom set and all the ground element states composing the current belief state. It is worth noting that several methods might have the same head to specify different ways of decomposing the abstract task *h*.

In order to handle feedback and conditional branching, PC-SHOP is augmented with a special task operator *cond* that has the following syntax:

(`cond` $(c_1\, e_1)\, (c_2\, e_2) \ldots (c_m\, e_m)$)

where c_i is a conjunct of observations, and e_i is a list of tasks to expand if c_i is found to hold in the observations set of one belief-state. The task operator `cond` specifies that a branch should be generated for each pair (c_i e_i) if there is a belief state whose observations satisfy c_i. The branch includes the tasks in the task list e_i. We impose

```
Procedure PC-SHOP($BS, T, D$)
01. if $T$ = nil then return (success, $p(bs \in BS)$) endif
02. $t$ = the first task in $T$
03. $U$ = the remaining tasks in $T$
04. if $|BS| \geq 2$ and $t$ is not conditional then
05.     $t$ = (cond (nil ($t$))) endif
06. switch(t)
07. Case1: $t$ is primitive and has a simple plan $p$
08.     (Plan, prob)=PC-SHOP(result($bs \in BS, p$), $U, D$)
09.     if Plan = fail then return (fail, 0 ) endif
10.     return ($p$;Plan, prob)
11. Case2: $t$ is conditional
12.     for each branch ($c$  $e$) of $t$
13.       ($br_i, prob_i$)=apply-branching($BS$, ($c$ ($e; U$)), $D$)
14.     endfor
15.     if all $br_i$ = fail then return (fail, 0) endif
16.     return ((cond $br_1\ br_2 \ldots br_m$),$\sum_{i=1}^{m} prob_i$)
17. Case3: $t$ is compound and reduces in $bs \in BS$
18.     for each reduction $R_i$ of $t$ in $bs$
19.       ($plan_i, prob_i$) = PC-SHOP($BS$, ($R_i; U$), $D$)
20.     endfor
21.       return the ($plan_i, prob_i$) with the maximal $prob_i$.
22. Otherwise: return (fail, 0)
23. endswitch
END
```

Figure 2: The PC-SHOP planning algorithm

that all the conditions c_i of a `cond` be exclusive.

Example 4. The following method specifies how to open a door based on the observation made by the action `(listen)`.

```
method:      (!choose-door)
precond:     (((?d1 ?d2)(door ?d1)
             (and (door ?d2)(not(= ?d1 ?d2)))))
reduction:   ((listen)
               (cond ((tiger=?d1)(open ?d2))
                   ((tiger=?d2)(open ?d1))))
```

3 The Planning Algorithm

PC-SHOP is a total order forward search algorithm. The input is a set of belief states BS (initially one), an ordered list of tasks to achieve T, and a domain description D. The output is a plan with a success probability; if the returned success probability is zero "0", then the returned plan is a *fail*, which indicates that no plan was found to solve the planning problem. The algorithm starts by recursively decomposing the first task of T until it reduces to a primitive task, which is applied to produce one or more belief states used to apply the next task. The same process continues recursively until the list of tasks to achieve is empty. The planning algorithm is shown in figure 2.

The plans generated by PC-SHOP have the structure of a tree, where nodes with zero or one child are ground instances of actions, or a *cond* with one branch. Nodes that have more than one child are the conditional operator *cond*

```
**Procedure** apply-branching ($BS, branch, D$)
1. $C$ = conditional part of $branch$
2. $E$ = expansion part of $branch$
3. success-prob = 0; CPlan = nil
4. **for** all $bs \in BS$ where $C$ holds in $O_{bs}$ **do**
5.     $C'$ = satisfier of $C$ in $O_{bs}$
6.     $(BR, p)$ = PC-SHOP($\{bs\}, E, D$)
7.     success-prob += $p$
8.     CPlan = ($C'$ BR);Cplan
9. **endfor**
10. return (CPlan,success-prob)
**END**
```

Figure 3: The apply-branching procedure

with more than one branch.

Let $P(T)$ be a plan returned by PC-SHOP to achieve the tasks specified in T starting from BS, t the first task of T, and U the rest of the tasks. Then $P(T)$ has one of the following forms:

- $P(T) = success$: The algorithm returns success at step 1, if T is an empty task list.
- $P(T) = (p; P(U))$: if t is a primitive task and p is a ground action that achieves it. Then the algorithm returns $(p; P(U))$ at step 10. The semicolon ";" denotes the concatenation operator.
- $P(T) = P((R; U))$ is returned at step 21, if t is an abstract task and R is one of its decompositions. At this step t is replaced by one of its valid reductions that ensures maximum success probability.
- $P(T) = (cond(c_1 P((e_1; U)))\dots(c_m\ P((e_m; U))))$ is returned at step 16, if t is a *cond*. Each c_i is a ground conjunct of observations.

In the last case, each pair $(c_i\ P((e_i; U)))$ is returned by the procedure *apply-branching* defined in figure 3, which takes as input a list of belief states BS, the domain D, and a branch $(c_i\ (e_i; U))$. The output is a plan for $(e_i; U)$ with the ground c_i, if c_i is satisfied in the observations of one of the belief states in BS. A plan branch $P((e_i; U))$ might be equal to $fail$ if $(e_i; U)$ is not solvable.

Note that the planner may decide to introduce a *cond*, if BS has more than one belief state and the current task t is not already conditional (steps 4 and 5). In such a case, the new *cond* has one branch where the condition part is *nil* and the expansion part is the task t. The value *nil*, in the condition part, makes the expansion of t applicable in all the belief states in BS resulting in as many branches as there are belief states in BS.

Example 5. The following is a plan which is a solution to the tiger scenario with a success probability equal to 0.85, where `:success` denotes predicted plan success, and `:fail` denotes failure.

```
((listen)
  (cond ((tiger=rd) (open ld)
        (cond ((rew=t) :success)
              ((dead=t) :fail))))
       ((tiger=ld) (open=rd)
        (cond ((rew=t) :success)
              ((dead=t) :fail)))))
```

3.1 Application of Conditional Plans

The application of a plan to a belief state bs results in a set of new belief states. First, the application of an action a to bs is defined as in the previous section. Next, the application of a sequence is defined as applying the first action a to bs, and then the rest of the plan to each of the resulting belief states. Finally, the application of a conditional plan element `(cond` $(c_1\ p_1)\dots(c_n\ p_n)$`)` is defined as an application of the branch p_j whose context condition c_j matches with O_{bs} (there should be only one such branch).

4 Experimental Results

PC-SHOP was run on different classes of planning problems found in the literature. All the tests were performed on a 2 GHz Pentium 4 machine with 512 MB of RAM, using Allegro Common LISP as the programming language. Each test was performed 5 times and the median execution time, including garbage collection, was recorded.

4.1 Conditional Planning

In this class of planning problems, the domains are fully observable. The original problems did not include any probabilities. A uniform probability distribution over the ground initial states is assumed. PC-SHOP was asked to solve the corresponding planning problems with a success probability equal to one. Two planning domains were considered: the *open safe* domain (OSMC) [17] and the *medicate* domain [19]. OSMC problems involve the generation of a plan to open a safe using a fixed number of combinations. In such a domain, after trying a combination, two possible outcomes arise; either the safe is open, which achieves the goal, or when the safe remains closed which involves trying another combination and then performing the same test to determine whether to stop planning or continue trying the other combinations.

In the *medicate* domain, a patient is either without a disease or has one of d diseases. There is one action to diagnose the disease, and one action to medicate for each disease. The patient is cured if the right medicate action is applied, while applying the wrong one kills him. The goal is to cure the patient. The plan generated by PC-SHOP diagnoses the disease to conditionally select the right medicate action.

Table 1 gives execution times (in seconds, with a precision of 1ms) for the two domains along with execution

Table 1: Execution times for the domains: *Medicate* with n diseases, and *OSMC* with C combinations (nm = not mentioned).

Medicate			OSMC		
n	PlanPKS	PC-SHOP	C	PlanPKS	PC-SHOP
20	0.08	<1ms	60	0.08	<1ms
60	3.13	<1ms	80	0.18	<1ms
100	20.39	0.016	100	0.34	<1ms
200	nm	0.109	500	nm	0.047
500	nm	0.656	1000	nm	0.109
1000	nm	2.516	1500	nm	0.500

times by the conditional planner PlanPKS[1] [17].

We also ran PC-SHOP on a domain involving recursive task refinement. The domain consists of a robot, a set of rooms, a fire extinguisher located in one of the rooms, and a fire. The complete description of the domain is given in figure 4. Plans were generated to help the robot fight the fire in two scenarios. In the first scenario, the robot is uncertain about the location of the fire extinguisher; hence it starts with a belief state with a uniform probability distribution of the location of the extinguisher. Table 2 gives execution times taken by PC-SHOP to generate plans that, when executed, locate and fetch the fire extinguisher, go to the fire place, put the fire out and finally return the extinguisher to its initial location. The second scenario adds one dimension to the observation space, where it is possible that there are people in the different rooms. In this case the robot has to warn those people it encounters in the rooms it visits while searching for the extinguisher. We can remark that the execution times for the second scenario increase exponentially, whereas those for the first scenario do linearly. This is because the size of plans in the second scenario increase exponentially: each check for persons in a room leads to two new branches which continue to split. Looking for a fire extinguisher, on the other hand, results in two branches were one branch is just a short sequence. The exponential execution times could be reduced if PC-SHOP was able to exploit similarities between branches by joining branches or parameterizing ground actions of similar branches. Note that the belief states of the two branches are not completely identical, so to join the branches would be a non-trivial task.

We believe that the performance of PC-SHOP in these three domains is due to the procedural nature of solving the problems. PC-SHOP was also used to solve more trivial problems such as the *buy coffee, sugar, cream* problem [16]. For such domains, the knowledge of how to solve the problem is expressed as a set of ordered tasks to achieve, and by explicit specification of what to do if a contingency is observed during execution.

[1]Execution times of PlanPKS are taken from the paper describing the planner and they are reported for a 450MHz sun machine with 4 GB of memory, see [17] for more details.

Table 2: Execution times for the fire-fighting domain.

$\#rooms$	20	50	100	200	400	500
scenario1	0.016	0.062	0.25	1.188	7.50	14.39
$\#rooms$	3	10	11	12	13	14
scenario2	<1ms	1.047	2.175	4.438	9.11	20.25

Table 3: Execution times for the omelette domain.

	Probability of success s				
	0.6	0.7	0.8	0.9	0.999
$p = 0.2$	2.562	5.219	9.609	19.344	432.297
$p = 0.5$	0.016	0.016	0.016	0.031	0.360
$p = 0.8$	0.078	0.141	0.234	0.360	5.171

4.2 Contingent Probabilistic Planning

In this class of problems, the actions may have probabilistic effects, leading to probability distributions over the next states. The solution for this class of problems is a plan that reaches the goal state with a success probability. To solve such problems PC-SHOP uses an iterative deepening strategy to cut off deep recursive calls of methods, since it is a depth first-search algorithm.

We tested PC-SHOP on the omelette problem reported in [3] where the domain is completely observable i.e. the observations are not probabilistic. The aim is to have one of two bowls contain three good eggs. Eggs have a probability p of being good and $(1 - p)$ of being bad. The agent breaks an egg in an empty bowl and uses a sensing action to check whether the broken egg is good. Table 3 gives execution times taken to generate plans with different success probabilities s knowing the probability of a good egg p.

We also tested PC-SHOP on the tiger domain to produce plans that get the reward with different success probabilities.The execution times (0.000s for 0.85 chance for success, 0.015s for 0.939, and 0.047s for 0.9733) were slightly faster than those taken by PTL-PLAN [10].

5 Related work

In this section we provide a brief overview of some of the approaches used in planning under uncertainty. As mentioned earlier, most proposed approaches try to relax the assumptions of classical planners by allowing actions to have several outcomes and by introducing sensing actions for gathering information during the execution phase of the plan.

Approaches that use a stochastic model of actions and sensing include C-BURIDAN [7], Mahinur [16], PTL-PLAN [10], C-MAXPLAN [12], and GPT [3]. Planners under this category of approaches solve planning problems by generating plans with a success threshold expressed generally as a probability degree. Furthermore the use of sens-

```
action:    (goto ?r)
results:   (and (at-fire-place = f) (in-room ?r = t)) )

action:    (check-in ?r)
results:   (and (checked ?r = t)
           (case
            ((ext-in = ?r) (and (has-extinguisher = t)(origin =?r)
                           (obs (found-ext ?r = t)) ) )
            ((not (ext-in = ?r )) (obs (found-ext ?r = f)) ) ) )

action:    (go-fight-fire ?r)
results:   (at-fire-place)

action:    (extinguish )
results:   (fire = f)

method:    (!check-rooms )
precond:   ((?r) (room ?r = t) (checked ?r = f))
reduction:((check-in ?r)
            (cond ((found-ext ?r = t)   (go-fight-fire ?r))
                  ((found-ext ?r = f)   (!check-rooms ))))

method:    (!put-back)
precond:   ((?r) (room ?r) (and (origin ?r) (at-fire-place)))
reduction:((goto ?r)))

method:    (!put-fire-out)
precond:   (() (has-extinguisher) (at-fire-place))
reduction:((extinguish ))

method:    (!fight-fire)
precond:   nil
reduction:((!check-rooms ) (!put-fire-out) (!put-back))
```

Figure 4: The fire fighting domain.

ing actions generates contingent plans. Decision-theoretic planners [4, 9] can also be considered under this category, since the policies they generate are a form contingency plans that maximize utility.

The literature includes also a category of planners that generate conditional plans but do not use probabilistic action models. The generated plan ensures that goal states are reached by specifying a plan branch for all possible contingencies. In other words this category of planners can be considered as a special case of the first category where the probability of success is 1.0. PlanPKS [17], SGP [19] and the planner presented in [1] fall under this category.

Examples of planning under uncertainty using hierarchical decomposition include Drips [8] and ND-SHOP2 [11]. In Drips abstract actions are used to build abstract plans that are compared with respect to their expected utilities. Plans with higher utilities are repeatedly refined until an optimal plan is found. However Drips does not support sensing, and generated plans are sequences of actions. The more recent non-deterministic HTN planner ND-SHOP2, extends SHOP2 [15] (a successor of SHOP) to address fully observable domains with nondeterministic actions and multiple initial states. However PC-SHOP differs from ND-SHOP2, as PC-SHOP handles planning problems involving partial observability and probabilities.

For a good characterization of different types of planning problems with sensing and uncertainty, as well as some heuristic techniques based on goal distance estimates, the reader is referred to [2].

6 Conclusions

In this paper, we have presented a hierarchical task planner that handles uncertainty both in the state of the world and the effects of actions. Belief states have been used to represent incomplete and uncertain information in the environment, and actions are allowed to have context-dependent and stochastic effects that might include making observations. PC-SHOP has been shown to generate plans that can handle contingencies at execution time through the use of a conditional operator that selects which branch to follow depending on what observations are made.

The proposed algorithm uses extensive domain-knowledge to describe how to solve the planning problem using an extension of the syntax of the classical HTN planner SHOP [13, 14]. In our experiments PC-SHOP performed very well on domains found in the literature. The performance of PC-SHOP is justified by the fact that many planning domains can take advantage of the efficiency of controlling search through hierarchical task refinement networks.

Regarding future work, efforts will be more focused on finding better ways to search plans that exceed a threshold of success, and more work will be done to make it possible to join branches if they share a similar sequence of actions, which we believe will further improve the execution times of the algorithm.

7 Acknowledgments

This work has been supported by the Swedish KK foundation and the Swedish Research Council.

REFERENCES

[1] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in non-deterministic domains under partial observability via symbolic model checking. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence. (IJCAI)*, pages 473–478, 2001.

[2] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. of the 5th Int. Conf. on Artificial Intelligence Planning Systems*, pages 52–61, 2000.

[3] B. Bonet and H. Geffner. GPT: A tool for planning with uncertainty and partial information. In *IJCAI-01 Workshop on Planning with Uncertainty and Incomplete Information*, pages 82–87, Seattle, WA, August 2001.

[4] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Artificial Intelligence Research*, 11:1–94, 1999.

[5] M. Broxvall, S. Coradeschi, L. Karlsson, and A. Saffiotti. Recovery planning for ambiguous cases in perceptual anchoring. In *Proc. of the 20th AAAI Conf.*, Pittsburgh, PA, 2005. AAAI Press. Online at http://www.aass.oru.se/˜asaffio/.

[6] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[7] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, pages 31–63, 1994.

[8] P. Haddawy and M. Suwandi. Decision-theoretic refinement planning using inheritance abstraction. In *Proc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, pages 266–271, 1994.

[9] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[10] L. Karlsson. Conditional progressive planning under uncertainty. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 431–438, 2001.

[11] U. Kuter and D. Nau. Forward-chaining planning in nondeterministic domains. In *Proc. of the 19th National Conference on Artificial Intelligence (AAAI).*, 2004.

[12] S. Majercik and M. Littman. Contingent planning under uncertainty via stochastic satisfiability. In *Proc. of the 16th National Conf. on Artificial Intelligence (AAAI)*, 1999.

[13] D. Nau, Y. Cao, A. Lotem, and H. Muñoz Avila. SHOP: Simple hierarchical ordered planner. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 968–973, 1999.

[14] D. Nau, Y. Cao, A. Lotem, and H. Muñoz Avila. SHOP and M-SHOP: Planning with ordered task decomposition. Technical Report CS TR 4157, University of Maryland, College Park, MD, 2000.

[15] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Artificial Intelligence Research*, 20:379–404, 2003.

[16] N. Onder and M. Pollack. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proc. of the 16th National Conf. on Artificial Intelligence (AAAI)*, pages 577–584, 1999.

[17] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of the 6th Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, pages 212–222, 2002.

[18] A. Tate. Generating project networks. In *Proc. of 5th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 888–93, Cambridge, Massachusetts, 1977. Morgan Kaufmann.

[19] D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proc. of the 15th National Conf. on Artificial Intelligence (AAAI)*, pages 897–904, 1998.

[20] D. E. Wilkins. *Practical Planning: extending the classical AI paradigm.* Morgan Kaufmann, 1988.

CONTACTS
Abdelbaki BOUGUERRA, Lars KARLSSON
Centre for Applied Autonomous Sensor Systems
Department of Technology; Örebro University
SE-70182 Örebro
Sweden
Email:{abdelbaki.bouguerra;lars.karlsson}@tech.oru.se

Abdelbaki BOUGUERRA is currently a PhD student at the department of technology at Örebro University in Sweden. His main research interests include plan-based control, execution monitoring, and failure recovery for mobile robots.

Lars Karlsson is assistant professor at the department of technology at Örebro University in Sweden. His main research interests are sensor-based planning, plan-based control of robots, and perceptual anchoring.