Hierarchical Task Planning under Uncertainty*

Abdelbaki BOUGUERRA and Lars KARLSSON

Department of Technology
Örebro University, SE-70182 Örebro; Sweden
Email: {abdelbaki.bouguerra;lars.karlsson}@tech.oru.se

Abstract. In this paper we present an algorithm for planning in non-deterministic domains. Our algorithm C-SHOP extends the successful classical HTN planner SHOP, by introducing new mechanisms to handle situations where there is incomplete and uncertain information about the state of the environment. Being an HTN planner, C-SHOP supports coding domain-dependent knowledge in a powerful way that describes how to solve the planning problem.
To handle uncertainty, belief states are used to represent incomplete information about the state of the world, and actions are allowed to have stochastic outcomes. This allows our algorithm to solve problems involving partial observability through feedback at execution time. We outline the main characteristics of the algorithm, and present performance results on some problems found in literature.

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 the non-determinism of actions [4][13][1].

Classical planners assume that only actions change deterministically the state of the world. 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 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.

In developing more robust planners, researchers have tried to relax the assumptions of classical planners, leading to different approaches that handle uncertainty. Pure conditional planners assume that, at execution time, the state of the world is fully observable, and construct plans that have the structure of an *alternative* "if-then-else" [13][12]. Probabilistic conditional planners assign probabilities to the outcomes of actions, and search for plans with a probability of success that exceeds a certain threshold [4][11].

* This work has been supported by the Swedish KK foundation and the Swedish research council

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 C-SHOP "Conditional SHOP", follows the success of the classical HTN planner SHOP [9][10], and introduces a mechanism to extend SHOP to handle uncertainty. Being an HTN planner, C-SHOP 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 support of larger domains. SHOP is known to be a simple but an outperforming HTN planner, where tasks are planned for in the same order that they will be executed. The main additions of C-SHOP to SHOP are:

- executable tasks may have different effects.
- sensing is integrated with executable tasks to make observations.
- uncertainty about the state of the environment is represented in belief states.
- observations and effects are associated with probabilities.

These additions allow C-SHOP to generate plans that are no longer just sequences of actions, but plans that have conditional branches.

The rest of the paper is organized as follows. Next section gives an overview about some of the state-of-the-art planners dealing with uncertainty. The planning algorithm is, then, presented along with domain-knowledge representation. Experimental results over some domains is shown next. The paper terminates with some concluding remarks.

2 RELATED WORK

In real world applications, environments tend to be unpredictable. Agents, executing "blindly" their plans in such environments, are doomed to fail because of incomplete information, or actions themselves are not deterministic due to unreliable actuators.

To cope with the problem of uncertainty, many approaches have been proposed, each of which trying to relax the assumptions of classical planners by allowing actions to have several outcomes and by introducing actions for gathering information during the execution phase of the plan.

C-BURIDAN [4], which is one of the first conditional probabilistic planners, makes the distinction between actions that change the state of the world and actions that change the knowledge about the world. Generating plans by C-BURIDAN proved expensive because of the blow up of the search space. Mahinur [11] improved on C-BURIDAN by deliberately choosing what contingencies to plan for and leaving others until execution time. The selected contingencies are those determined as having a great impact on the success of goal achievement. C-MAXPLAN [8] is a conditional probabilistic planner that transforms the planning problem into a stochastic satisfiability problem, and then apply known techniques to solve it. A more recent possibilistic/probabilistic conditional planner is PTLPLAN [7] which uses temporal logic to code domain knowledge to help reduce the search space.

The literature of pure conditional planners include the planner presented in [1] where the use of Binary Decision Diagrams (BDDs) and symbolic model checking approach is shown to reduce the search space allowing the proposed conditional planner to outperform most contemporary planners. PlanPKS [12] is a conditional planner that extends the representation of the state of the world to more than the STRIPS set of facts and adds three more sets of facts that contain formulas of a first order logic. Each set of facts represents one form of knowledge, and actions specify which of the sets to update. Nevertheless, PlanPKS is unable to handle planning problems involving complex possible world configurations.

An automated manufacturing planning system is presented in [3]. The planner integrates hierarchical and conditional planning techniques to generate plans that take into account contingencies induced by the states of the sensors in the planning problem. the planner uses hierarchical task decomposition to describe tasks, but contingencies are solved through the call to a separate conditional planner that returns conditional plans for a certain level of abstraction. Drips [5] is a planner that supports hierarchical decomposition as well as abstraction of actions. Abstract actions are produced by grouping outcomes of operators, with probabilistic and conditional effects. Drips uses abstract actions 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. Sensing is not handled, and generated plans are sequences of actions.

Another track of research on planning under uncertainty view the planning as a decision taking problem which involves the generation of policies that maximize utilities. MDPs, in the case of fully observable domains, and POMDPs, in the case of partial observability, were proposed as tools to solve these classes of the problem [2][6]. However such techniques require to enumerate all the states of the world which makes them unable to solve large states problems.

3 REPRESENTATION

To describe a domain in SHOP as well as C-SHOP, methods, operators, and axioms are used. An HTN planner specifies tasks to achieve goals. Tasks can be primitive, or abstract (compound) if they need to be decomposed into other tasks. Throughout this paper, we use the fire-fighting scenario to illustrate the representation of C-SHOP entities. The scenario includes a robot that has to fetch a fire extinguisher to fight a fire. The definition of the fire-fighting domain is given in figure 1.

3.1 Belief states

An agent planning for a domain with partial observability does not know exactly in which state the executor will be after executing a plan step. Therefore belief states are used to model uncertainty about the state of the world.

Belief states in C-SHOP are probability distributions over the possible elementary states (an elementary state is a set of ground atoms). Furthermore,

O1	(:operator (!goto ?r) ((() 1.0 ((at-fire-place)) ((in-room ?r)) ())))
O2	(:operator (!check-in ?r) ((((ext-in ?r)) 1.0 () ((checked ?r)) ((found-ext ?r))) (((not(ext-in ?r))) 1.0 () ((checked ?r)) ((not-found-ext ?r)))))
O3	(:operator (!go-fight-fire ?r) ((() 1.0 () ((origin ?r)(at-fire-place)) ())))
O4	(:operator (!extinguish) ((() 1.0 ((fire)) () ()))
M1	(:method (check-rooms) ((room ?r)(not (checked ?r))) '((!check-in ?r)(COND (((found-ext ?r))((!go-fight-fire ?r))) (((not-found-ext ?r))((check-rooms))))))
M2	(:method (put-back) ((at-fire-place)(room ?r)(origin ?r)) '((!goto ?r)))
M3	(:method (put-fire-out) ((at-fire-place)) '((!extinguish)))
M4	(:method (fight-fire) () '((check-rooms) (put-fire-out) (put-back)))

Fig. 1. The fire fighting domain.

each belief state has a set of observations that will be made during plan execution. Observations represent the feedback for the plan executor to determine the condition to branch on.

We adopt the representation used in PTLPLAN [7] for belief states and associate a global probability $p(bs)$ for each belief state bs calculated as the sum of the probabilities of its constituent elementary states.

Example 1. Suppose that the robot has to fetch the fire extinguisher in one of three rooms $\{r_1,r_2,r_3\}$. Since the robot does not know exactly in which room the fire extinguisher is, it can assume that it is in each room with a probability of 1/3. Hence, the belief state would look like:

($obs = \phi$; $p = 1.0$;
state1: 1/3; {(ext-in r_1),(fire),(room r_1),(room r_2),(room r_3)}
state2: 1/3; {(ext-in r_2),(fire),(room r_1),(room r_2),(room r_3)}
state3: 1/3; {(ext-in r_3),(fire),(room r_1),(room r_2),(room r_3)})

3.2 Methods

In SHOP, methods are used to control search and they provide the knowledge of how to decompose tasks in order to solve a more abstract task. C-SHOP uses

the same syntax as SHOP to code methods. A method has the following form:

(:method 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 conjunct, and t_i is a task list. A method specifies that 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 elementary states composing the current belief state.

In order to handle feedback and conditional branching, C-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 one belief-state observations. The task operator *COND* specifies that a branching 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 that all the conditions C_i of a *COND* be exclusive.

Example 2. The method $M1$, defined in figure 1, checks the different rooms for the extinguisher. If there is a room ?r which was not checked yet, then the primitive task (!check-in ?r) is applied to check it. Then, *COND* generates a plan to fight the fire, if the fire extinguisher is observed to be in the room, otherwise a different room is checked.

3.3 Operators

Operators are used to alter the state of the world by adding new facts and deleting old ones. Operators in C-SHOP can model tasks with conditional and probabilistic effects and information gathering during execution. An operator in C-SHOP has the form:

(:operator h *results*)

where the operator's head h is a primitive task, and *results* is a list describing the different outcomes of the task. Each element of *results* is a 5-tuple (C, p, D, A, O), where C is a conjunct describing in what context the result is applicable. D (resp. A) is a list of atoms to delete from (resp. add to) the elementary state, O is a list of observations made after executing the operator, and p is the probability of the outcome. In any state, the sum of p of the applicable results must be "1".

Example 3. The operator $O2$, in figure 1, permits to look for the extinguisher in a room ?r. The first outcome states that if the extinguisher is really in room ?r then it will be observed with probability of 1 that it was found in room ?r, whereas, the second outcome leads to the observation that it was not found in room ?r. Both outcomes specify that, after the execution of the operator, the room will be set as checked.

3.4 Semantics

The application of an instantiated operator, (:operator h *Results*), in a belief state bs is denoted by result(bs,h) and is computed as follows:

For each outcome $R_i = (C_i, p_i, D_i, A_i, O_i) \in Results$ and for each elementary state $s_1 \in bs$, if C_i holds in s_1 then a new elementary state s_2 is created where $s_2 = (s_1 - D) \cup A$ and the observations in O_i are made. The probability $p(s_2)$ of being in s_2 is given by $p(s_2) = p(s_1) \cdot p_i$. Then, new belief states are created by grouping all the new elementary states s_2 that were created with the same observations set. The probability of each of the new belief states is calculated as the sum of the probabilities of its constituent elementary states i.e. $p(bs) = \sum_{s \in bs} p(s)$.

Example 4. Applying the operator $O2$ to achieve the primitive task (!check-in r_1), in the belief state given in example 1, gives the following new belief states:

$bs1$ =(obs = {(found-ext r_1)}; $p = 1/3$;
state1: 1/3; {(ext-in r_1),(fire),(checked r_1),(room r_1),(room r_2),(room r_3)})

$bs2$ =(obs = {(not-found-ext r_1)}; $p = 2/3$;
state2: 1/3; {(ext-in r_2),(fire),(checked r_1),(room r_1),(room r_2),(room r_3)}
state3: 1/3; {(ext-in r_3),(fire),(checked r_1),(room r_1),(room r_2),(room r_3)})

4 THE PLANNING ALGORITHM

C-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 C-SHOP have the structure of a tree, where nodes with zero or one child are ground instances of operators, or a *COND* with one branch. Nodes that have more than one child are the conditional operator *COND* with more than one branch.

Let $P(T)$ be a plan returned by C-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 operator that achieves it. then the algorithm returns $p; P(U)$ at step 9.
- $P(T) = P(R; U)$ is returned at step 18, 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.

Procedure C-SHOP (BS,T,D)
1. **IF** $T = nil$ **THEN** return (*success,* $p(BS)$) **ENDIF**
2. t = the first task in T
3. U = the remaining tasks in T
4. **IF** length($BS \geq 2$) AND t is not conditional **THEN**
5. $t = (COND\ (nil\ (t)))$
6. **IF** t is primitive and there is a simple plan p for t **THEN**
7. (*Plan, prob*) = C-SHOP (result $(BS, p), U, D$)
8. **IF** *Plan* = *fail* **THEN** return (*fail, 0*)
9. return (cons (*p,Plan*), *prob*)
10. **ELSE IF** t is conditional **THEN**
11. **FOR** each *branch* (C E) of t
12. $(br_i, prob_i)$ = Apply-branching (BS,(C append(E, U)),D)
13. **ENDFOR**
14. **IF** all $br_i = nil$ **THEN** return (*fail, 0*)
15. return (cons ((COND br_1 br_2 ... br_m)),$\sum_{i=1}^{m} prob_i$)
16. **ELSE IF** t is non-primitive and there is a simple reduction for t in BS **THEN**
17. choose a reduction R of t in BS
18. return C-SHOP(BS, append(R,U),D)
19. **ELSE**
20. return (*fail, 0*)
21. **ENDIF**
22. **END**

Fig. 2. The C-SHOP planning algorithm

- $P(T) = (COND\ (C_1\ P(E_1; U)) \ldots (C_m\ P(E_m; U)))$ is returned at step 15, if t is a $COND$. Each C_i is a ground conjunct of observations.

In the latter 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.

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.

The application of a plan P starting in an initial belief state *bs* is defined as applying the first step of P in *bs* and applying the rest of P to the the resulting belief state(s). If a plan step is *COND* then the branch to select is the one that has its condition part satisfied in the observations of the belief state (at execution time, there should be only one branch to follow).

Example 5. The following plan is a valid plan generated by C-SHOP to solve the planning problem presented by the fire-fighting scenario with three rooms

Procedure apply-branching $(BS, branch, D)$
C = conditional part of $branch$
E = expansion part of $branch$
success-prob = 0; *CPlan* = *nil*
FOR all $bs \in BS$ where C holds in $obs(bs)$
C' = satisfier of C in $obs(bs)$
(BR, p) = C-SHOP(bs,E,D)
success-prob += p
CPlan = *append (cons (C′ BR),Cplan)*
ENDFOR
return (*CPlan*,*success-prob*)
END

Fig. 3. The apply-branching procedure

and the initial task list ((fight-fire)). In this scenario, the robot has to put back the fire extinguisher in the room where it was found.

```
((!check-in r1)
(COND (((found-ext r1)) ((!go-fight-fire r1)(!extinguish)(!goto r1)))
   (((not-found-ext r1)) ((!check-in r3)
    (COND (((found-ext r3)) ((!go-fight-fire r3)(!extinguish) (!goto r3)))
       (((not-found-ext r3)) ((!check-in r2)
        (COND (((found-ext r2)) ((!go-fight-fire r2)(!extinguish)
           (!goto r2)))))))))))
```

5 Experimental Results

C-SHOP was run on several domains 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. The first experiments were performed to solve planning problems in the *open safe* domain (OSMC) [12] where a plan has to be devised to open a safe using a fixed number of combinations. The specification of the domain is easily handled by C-SHOP. 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.
The other scenario, we tested C-SHOP on, is the *medicate* domain [13], where a patient is either without a disease or he has one of d diseases. There is one action to diagnose the disease, which is cured if the right medicate action is applied, while applying the wrong one kills the patient. The goal is to cure the patient. Once again, this problem turned to be easy to be modeled and solved by C-SHOP. The plan involves diagnosing the disease then conditionally choosing the right medicate action. Table 1 gives execution times (in seconds, with a precision

of 1ms) for the two domains along with execution times by PlanPKS[1]. The table shows also execution times for the *fire-fighting* domain.
We believe that the performance of C-SHOP in these two domains is due to the hierarchical nature of solving the problems. C-SHOP was also used to solve more trivial problems such as the *buy coffee, sugar, cream* problem [11]. For such domains, it is easy to code the knowledge of how to solve the problem by giving the set of the ordered tasks to achieve, and by explicitly specifying the different courses of action if an exception (branching) is observed during execution.
To solve problems involving a certain degree of success, we let C-SHOP select an abstract task's reduction only if it is optimal i.e. the one that has the highest probability of success. Since, C-SHOP employs depth first search, an iterative deepening strategy is used to cut off deep recursive calls of methods. We run the algorithm on the tiger domain [6] for different success degrees. The execution times were slightly faster than those taken by PTLPLAN [7]. The tiger domain does not benefit from a hierarchical approach: each step is essentially a choice between listening once more and opening a door. In future work, more work will be done to improve the way the algorithm maximizes the probability of success.

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

Medicate	n	20	60	100	200	500	1000
	PlanPKS	0.08	3.13	20.39	nm	nm	nm
	C-SHOP	0	0	0.016	0.016	0.156	0.578
OSMC	C	60	80	100	500	1000	1500
	PlanPKS	0.08	0.18	0.34	nm	nm	nm
	C-SHOP	0	0	0	0.047	0.109	0.500
Fire	n	15	20	30	50	100	200
Fighting	C-SHOP	0	0.015	0.031	0.234	2.141	25.812

5.1 CONCLUSION

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. To handle partial observability, probabilities are used to represent noise in actions and in sensing. C-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.

[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 [12] for more details

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 [9][10]. In our experiments C-SHOP performed very well on domains found in the literature. The performance of C-SHOP is justified by the fact that many planning domains can be described as an ordered list of tasks to achieve through a hierarchical refinement process.
Regarding future work, efforts will be more focused on finding better ways to search plans that exceed a threshold of success. Furthermore we are currently integrating C-SHOP with an executor on a mobile robot, to test its performances in real world environments.

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. C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
3. L. Castillo, J. Fdez-Olivares, and A. González. Integrating hierarchical and conditional planning techniques into a software design process for automated manufacturing. In *Workshop on Planning under Uncertainty and Incomplete Information, 13th Int. Conf. on Automatic Planning and Scheduling (ICAPS)*, 2003.
4. 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.
5. P. Haddawy and Suwandi M. Decision-theoretic refinement planning using inheritance abstraction. In *Proc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, pages 266–271, 1994.
6. L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
7. L. Karlsson. Conditional progressive planning under uncertainty. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 431–438, 2001.
8. S. Majercik and M. Littman. Contingent planning under unceratinty via stochastic statisfiability. In *Proc. of AAAI*, 1999.
9. 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.
10. 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.
11. N. Onder and M. Pollack. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proc. of AAAI*, pages 577–584, 1999.
12. 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.
13. D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proc. of AAAI*, pages 897–904, 1998.