ITC irst

CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e−mail: prdoc@itc.it − url: http://www.itc.it

CONDITIONAL PLANNING UNDER PARTIAL OBSERVABILITY AS HEURISTIC−SYMBOLIC SEARCH IN BELIEF SPACE

Bertoli P., Cimatti A.,
Roveri M.

August 2001

Technical Report # 0108−01

Conditional Planning under Partial Observability as Heuristic-Symbolic Search in Belief Space ⋆

Piergiorgio Bertoli[1], Alessandro Cimatti[1], Marco Roveri[1,2]
[1] ITC-IRST, Via Sommarive 18, 38055 Povo, Trento, Italy
{bertoli,cimatti,roveri}@irst.itc.it
[2] DSI, University of Milano, Via Comelico 39, 20135 Milano, Italy
Phone: +39 0461 314 517. Fax: +39 0461 302 040.

Abstract. Planning under partial observability is one of the most significant and challenging planning problems. It combines the need to deal with belief states, as in conformant planning, with the need for and-or search, typical of conditional planning. In this paper, we tackle the problem of planning under partial observability within the framework of planning via symbolic model checking. We propose a new algorithm for planning under partial observability, able to generate conditional plans that are guaranteed to achieve the goal despite of the uncertainty in the initial condition, the uncertain effects of actions, and the partial observability of the domain. The proposed algorithm implements and-or search in the space of beliefs, by combining heuristic search with symbolic BDD-based techniques, and can tackle significant problems. The experimental evaluation compares the heuristic-symbolic algorithm with a recently proposed approach based on a depth-first search (DFS) style. We show that, while DFS search may often result in deep plans, and suffers sometimes from inefficiencies resulting from a "bad" initial choice, heuristic-symbolic search, even considering only a fixed simple selection function, constructs plans of better quality, and, for certain classes of problems, may significantly improve the efficiency of the search.

Keywords: Planning under Uncertainty, Conditional Planning, Binary Decision Diagrams, Symbolic Model Checking.

⋆ This paper has not already been accepted by and is not currently under review for a journal or another conference, nor will it be submitted for such during ECP's review period.

1 Introduction

Research in planning is more and more focusing on the problem of planning in nondeterministic domains and with incomplete information, see for instance [PC96; KBSD97; WAS98; CRT98; Rin99a; BG00]. A crucial assumption, upon which the search mechanism and the structure of the generated plans depend, is how information is available at run-time. For instance, the approaches in [KBSD97; CRT98] construct conditional plans under the assumption of full observability, i.e. the state of the world can be completely observed at run-time. At the other end of the spectrum, conformant planning [SW98; BG00; CR00; BCR01] constructs sequential plans that are guaranteed to solve the problem assuming that no information at all is available at run-time.

In this paper, we tackle the problem in the middle of the spectrum, i.e. planning under *partial observability*, the general case where only part of the domain information is available at run time. This problem is significantly more difficult than the two limit cases of fully observable and conformant planning. Compared to planning under full observability, planning under partial observability must deal with uncertainty about the state in which the actions will be executed. This makes the search space no longer the set of states of the domain, but its powerset, i.e. the space of "belief states" [BG00]. Compared to conformant planning, the structure of the plan is no longer sequential, but tree-shaped, in order to represent a conditional course of actions. Several approaches have been previously proposed, based e.g. on the extension of non linear planning [PC96] and of GraphPlan [WAS98], or as Partially Observable Markov Decision Processes (POMDP) [BG00].

We propose a new algorithm for planning under partial observability, able to generate conditional plans that are guaranteed to achieve the goal despite of the uncertainty in the initial condition, the uncertain effects of actions, and the partial observability of the domain. The algorithm is based on a general model of partial observability, that can represent both observations resulting from the execution of sensing actions [PC96; WAS98] and automatic sensing that depends on the current state of the world [TK00]. The planning algorithm performs a search of the (possibly cyclic) and-or graph induced by the domain, and generates conditional, acyclic plans. The planning algorithm combines the advantages of the symbolic, BDD-based techniques, with a heuristic-style search amenable to the use of selection functions. We call the approach *heuristic-symbolic* planning. The algorithm is implemented in the MBP planner [BCP+01].

A recent approach to planning under partial observability has been proposed in [BCRT01]. Its distinguishing feature is the depth-first style of the search (DFS), that allows to explore one path at a time, and makes it easy to deal with cycles. The experimental evaluation presented in [BCRT01] shows that DFS is very efficient, and outperforms the other conditional planners. There are, however, shortcomings related to DFS. In certain cases, the plans are extremely deep. Furthermore, the algorithm is very sensitive to initial bad choices: for instance, if it enters a portion of state space that admits no solution, the algorithm has to complete the exploration before being able to backtrack and consider a different portion of state space. Furthermore, DFS is not fully amenable to the use of selection functions. The heuristic-symbolic algorithm presented in this paper gives up the DFS exploration style building explicitly the and-or graph, and allows for the exploration of many paths at the time, combined with the use of selection functions. The experimental evaluation shows that the heuristic-symbolic algorithm constructs better plans and, for certain classes of problems, significantly improves the performance with respect to DFS.

The paper is organized as follows. In Section 2 we provide a formal definition of partially observable planning domains and of conditional planning. In Section 3 we present the planning

algorithm, and, in Section 4, its implementation in the MBP planner. In Section 5 we report on the experimental evaluation, and in Section 6 discuss further related work, and draw some conclusions.

2 Domains, Plans, and Planning Problems

We consider nondeterministic domains under the hypothesis of partial observability, i.e. where a limited amount of information can be acquired at run time.

Definition 1. *A partially observable planning domain is a tuple* $\langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{O}, \mathcal{X} \rangle$*, where*

- $\mathcal{P}$ *is a finite set of propositions;*
- $\mathcal{S} \subseteq Pow(\mathcal{P})$ *is the set of states;*
- $\mathcal{A}$ *is a finite set of actions.*
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ *is the transition relation.*
- $\mathcal{O}$ *is a finite set of observation variables;*
- $\mathcal{X} : \mathcal{O} \rightarrow Pow(\mathcal{S} \times \mathcal{A} \times \{\top, \bot\})$ *is the observation relation.*

Intuitively, a state is a collection of the propositions holding in it. The transition relation describes the effects of action execution. An action a is applicable in a state s iff there exists at least one state s' such that $\mathcal{R}(s, a, s')$. The set $\mathcal{O}$ contains observation variables, whose value can be observed at run-time, during execution. Without loss of generality, we assume that observation variables are boolean. We use o to denote observation variables. We call $\mathcal{X}(o)$, written $\mathcal{X}_o$ in the following, the observation relation of o. Given an action (that has been executed) and the resulting state, $\mathcal{X}_o$ specifies what are the values that can be assumed at run-time by the observation variable o. In a state $s \in \mathcal{S}$ after an action $a \in \mathcal{A}$, an observation variable $o \in \mathcal{O}$ may convey no information: this is specified by stating that both $\mathcal{X}_o(s, a, \top)$ and $\mathcal{X}_o(s, a, \bot)$ hold, i.e. both the true and false values are possible. In this case, we say that o is *undefined* in s after a. If $\mathcal{X}_o(s, a, \top)$ holds and $\mathcal{X}_o(s, a, \bot)$ does not hold, then the value of o in state s after a is true. The dual holds for the false value. In both cases, we say that o is *defined in* s *after* a. An observation variable is always associated with a value, i.e. for each $s \in \mathcal{S}$ and for each $a \in \mathcal{A}$, at least one of $\mathcal{X}_o(s, a, \top)$ and $\mathcal{X}_o(s, a, \bot)$ holds.

Consider the example of a simple robot navigation domain in Figure 1, in the upper left corner, containing a 2x2 room with an extra wall. The propositions of the domain are `NW`, `NE`, `SW`, and `SE`, corresponding to the four positions in the room. Exactly one of them holds in each of the four states in $\mathcal{S}$. The robot can move in the four directions (deterministic actions `GoNorth`, `GoSouth`, `GoWest` and `GoEast`), provided that there is not a wall in the direction of motion. The action is not applicable, otherwise. At each time tick, the information of walls proximity in each direction is available to the robot (observation variables `WallN`, `WallS`, `WallW` and `WallE`). For instance, we have that for any action a of the domain $\mathcal{X}_{\texttt{WallE}}(\texttt{NW}, a, \bot)$ and $\mathcal{X}_{\texttt{WallW}}(\texttt{NW}, a, \top)$. In this case, every observation variable is defined in every state and after the execution of any action of the domain. We call *action-independent* the observation variables that provide useful information automatically, independently of the previous execution of an action. We require an action-independent observation variable o to satisfy, for any $a_1, a_2 \in \mathcal{A}$ and any $s \in \mathcal{S}$, the following condition:

$$(\mathcal{X}_o(s, a_1, \top) \wedge \mathcal{X}_o(s, a_2, \top)) \vee (\mathcal{X}_o(s, a_1, \bot) \wedge \mathcal{X}_o(s, a_2, \bot))$$

In the following, we write $\mathcal{X}_o \subseteq \mathcal{S} \times \{\top, \bot\}$ when o is action-independent. In a different formulation of the domain, an action (e.g. `ObsWallE`) could be required in order to acquire the value

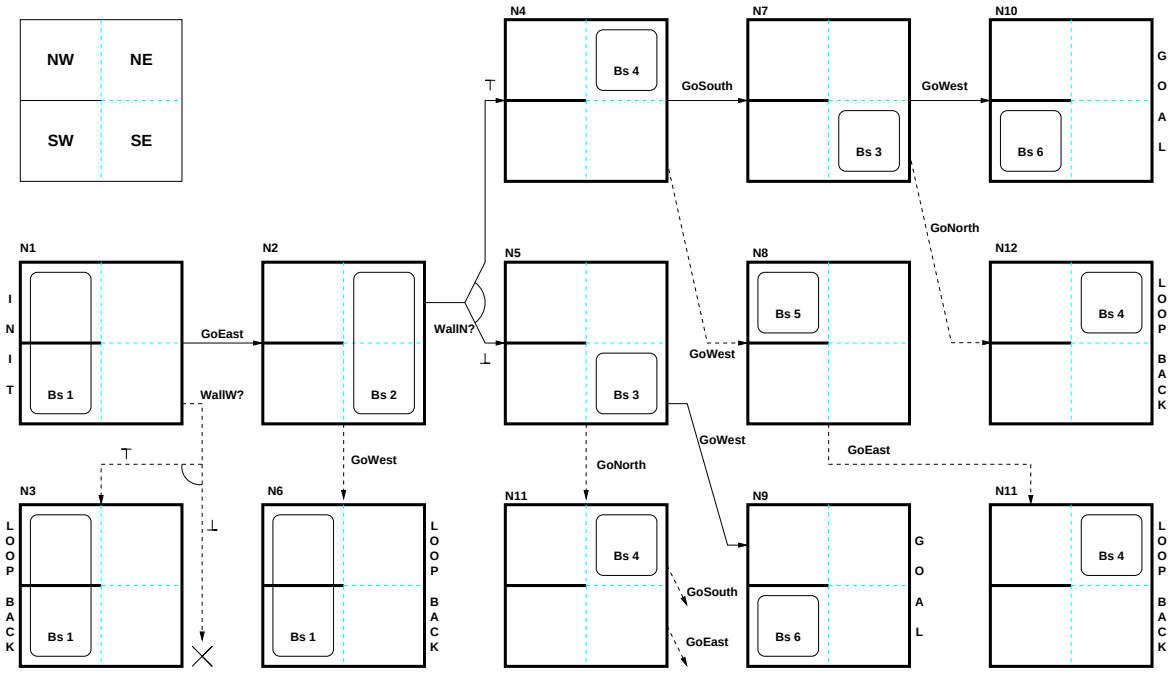

Fig. 1. A simple robot navigation domain

of a corresponding variable (e.g. `WallE`). In this case, `WallE` would be defined in any state after the action `ObsWallE`, and undefined otherwise. Such observation variables are modeled as *action-dependent*. Action-independent observation variables model "automatic sensing" [TK00], i.e. information that can always be acquired, as usual in embedded controllers, where a signal from the environment is sampled and acquired at a fixed rate, latched and internally available. Action-dependent observations are used in most observation-based approaches to planning (e.g. [WAS98]), where the value of a variable can be observed as the explicit effect of an action, like `ObsWallW`.

We consider conditional plans, that branch on the value of observable variables. A plan for a domain $\mathcal{D}$ is either the empty plan ϵ, an action $a \in \mathcal{A}$, the concatenation $\pi_1; \pi_2$ of two plans π_1 and π_2, or the conditional plan $o \ ? \ \pi_1 \ : \ \pi_2$ (read "if o then π_1 else π_2"), with $o \in \mathcal{O}$. Consider, for the example domain in Figure 1, the plan $\texttt{GoEast} \ ; \ \texttt{WallN} \ ? \ (\texttt{GoSouth} \ ; \ \texttt{GoWest}) \ : \ (\texttt{GoWest})$. The corresponding execution, starting from the uncertain initial condition `NW` or `SW`, is outlined in Figure 1 (solid lines). The initial condition is represented by node N1, where the set of the two possible initial states is depicted. After the action `GoEast`, in node N2, the robot is guaranteed to be either in `NE` or `SE`. The plan then branches on the value of `WallN`. The observation allows to distinguish between state `NE`, when the value associated with `WallN` is $\top$, and state `SE`, when the value is $\bot$. When the robot is in `NE`, in node N4, then it moves south, ending in node N7, and then it moves west, ending in node N10. When the robot is in `SE`, it simply moves west, ending in node N11. At this point, the robot is guaranteed to be in `SW`, regardless of the uncertainty in the initial condition.

The execution described above takes into account, at each node, an uncertainty condition, represented by a set of states that can not be distinguished. Such a set is called a "belief state". We say

that an action a is applicable to a non empty belief state Bs iff a is applicable in all states of Bs. In order to formalize the notion of plan execution, we define $\mathcal{X}_o[\top, a] \doteq \{s \in \mathcal{S} : \mathcal{X}_o(s, a, \top)\}$ as the set of states in $\mathcal{S}$ where o is true, and $\mathcal{X}_o[\bot, a] \doteq \{s \in \mathcal{S} : \mathcal{X}_o(s, a, \bot)\}$ as the set of states in $\mathcal{S}$ where o is false. If o is undefined in a state s after a, then $s \in \mathcal{X}_o[\top, a] \cap \mathcal{X}_o[\bot, a]$. In the case of action-independent observations, we have $\mathcal{X}_o[\top] \doteq \{s \in \mathcal{S} : \mathcal{X}_o(s, \top)\}$, and $\mathcal{X}_o[\bot] \doteq \{s \in \mathcal{S} : \mathcal{X}_o(s, \bot)\}$. We now formalize the notion of plan execution in the case of action-independent observations.

Definition 2. *Let $\emptyset \neq Bs \subseteq \mathcal{S}$. The execution of a plan in a set of states is defined as follows:*

1. *Exec*$[\pi](\emptyset) \doteq \emptyset$;
2. *Exec*$[a](Bs) \doteq \{s' | \mathcal{R}(s, a, s'),$ *with* $s \in Bs\}$,
   if a is applicable in Bs;
3. *Exec*$[a](Bs) \doteq \emptyset$, *if a is not applicable in Bs;*
4. *Exec*$[\epsilon](Bs) \doteq Bs$;
5. *Exec*$[\pi_1; \pi_2](Bs) \doteq$ *Exec*$[\pi_2]($*Exec*$[\pi_1](Bs))$;
6. *Exec*$[o\ ?\ \pi_1 : \pi_2](Bs) \doteq$
   Exec$[\pi_1](Bs \cap \mathcal{X}_o[\top]) \cup$ *Exec*$[\pi_2](Bs \cap \mathcal{X}_o[\bot])$, *if*
   (a) if $Bs \cap \mathcal{X}_o[\top] \neq \emptyset$, then *Exec*$[\pi_1](Bs \cap \mathcal{X}_o[\top]) \neq \emptyset$
   (b) if $Bs \cap \mathcal{X}_o[\bot] \neq \emptyset$, then *Exec*$[\pi_2](Bs \cap \mathcal{X}_o[\bot]) \neq \emptyset$
7. *Exec*$[o\ ?\ \pi_1 : \pi_2](Bs) \doteq \emptyset$ *otherwise.*

We say that a plan π is applicable in $Bs \neq \emptyset$ iff *Exec*$[\pi](Bs) \neq \emptyset$. If the plan is applicable, then its execution is the set of all states that can be reached after the execution of the plan. For conditional plans, we collapse into a single set the execution of the two branches (item *6*). The conditions *(a)* and *(b)* guarantee that both branches are executable. Definition 2 can be extended to the case of action-dependent observations by replacing $\mathcal{X}_o[\top]$ with $\mathcal{X}_o[\top, a]$ and $\mathcal{X}_o[\bot]$ with $\mathcal{X}_o[\bot, a]$, where a is the last action executed in the plan. Notice that, at starting time, action-dependent observation variables must be undefined since no action has been previously executed. For lack of space, we omit the explicit formal definition. We formalize the notion of planning problem under partial observability as follows.

Definition 3 (Planning Problem and Solution). *A planning problem is defined as a 3-tuple $\langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$, where $\mathcal{D}$ is a planning domain, $\emptyset \neq \mathcal{I} \subseteq \mathcal{S}$ is the set of initial states, and $\emptyset \neq \mathcal{G} \subseteq \mathcal{S}$ is the set of goal states. The plan π is a solution to the problem $\langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ iff $\emptyset \neq$ Exec$[\pi](\mathcal{I}) \subseteq \mathcal{G}$.*

3 Planning under Partial Observability

3.1 The Search Space

When planning under partial observability, the search space can be seen as an and-or graph, recursively constructed from the initial belief state, expanding each encountered belief state by every possible combination of applicable actions and observations. The graph is possibly cyclic, i.e. it is possible to return in the same belief state, representing the same condition of uncertainty. In order to rule out cyclic behaviors, however, the exploration can be limited to the acyclic prefix of the graph. Figure 1 depicts the finite prefix of the search space for the problem $\{\mathcal{I}, \mathcal{G}\}$, where $\mathcal{I}$ is $\{NW, SW\}$

and $\mathcal{G}$ is $\{SW\}$. Each node in the prefix is associated with a path, describing how the node has been reached, and with a corresponding belief state. The path is basically a sequence of actions, and assignments to observation variables. For instance, node N2 is associated with the path `GoEast`, and with the belief state Bs2, that is the result of the execution of `GoEast` in the belief state Bs1 associated with N1. Node N5 is associated with the path `GoEast;(WallN =` $\perp$`)`. The associated belief state Bs3 is the result of the fact that, in N2, `WallN` is observed, and it has value $\perp$. Bs3 is indeed Bs2 $\cap\, \mathcal{X}_{\texttt{WallN}}(\perp)$. Similarly, node N5 is associated with path `GoEast;(WallN =` $\top$`)`.

The prefix is constructed by expanding each node in all possible ways, each represented by an outgoing arc. Single-outcome arcs correspond to simple actions (action execution is deterministic in belief space). For instance, N4 expands into N7 and N8; the corresponding paths are obtained by appending the actions `GoSouth` and `GoWest` to path of N4. In general, consider a node N associated with path ρ and belief state Bs. N can be expanded with an action α if α is applicable in Bs, and results in a node associated with the path $\rho; \alpha$ and with the belief state $\mathit{Exec}[\alpha](Bs)$. Multiple outcome arcs correspond to observations. For instance, node N2 results in nodes N4 and N5, corresponding to the observation of `WallN`. The application of an observation is what gives the "and" component in the search space: we have a solution for (the belief state associated with) N2 if we have a solution for both N4 and N5. In Node N1, the arc labeled as `WallW?`, stands for a non-informative observation: it leads to node N3, with the same belief states as N1 (and to a cross representing the emptyset). This is due to the fact that the states of Bs1 are indistinguishable via observation, i.e. the observation variable `WallW?` has the same value in the different states of Bs1. Actually, none of the variables in $\mathcal{O}$ is informative, and therefore the other links are not explicitly reported. If we expand node N by considering the observation variable o, we obtain the two nodes associated with paths $\rho; (o = \top)$ and $\rho; (o = \perp)$, and the belief states $(Bs \cap \mathcal{X}_o[\top])$ and $(Bs \cap \mathcal{X}_o[\perp])$, respectively. The argument can be generalized to the case where k observations are taken into account at once, and result in (at most) 2^k paths. (In the case of action-dependent observations, the last action of the path of the expanded node must be taken into account to guarantee that meaningful information is acquired.)

In the following, we use the notions of father and ancestor node, and the dual notions of son and descendent, in the obvious way. For instance, N2 is the father of N4, N5 and N6, while N1 is ancestor for all nodes. We call "brothers" all the nodes that result from the same observation expansion of the same node. For instance, N4 and N5 are brothers, but N4 and N6 are not.

The expansion of nodes is halted under the following conditions. First, a node is a associated with a belief state contained in the goal. Second, a node is a loop back, i.e. it has an ancestor node with the same belief state. For instance, node N6 loops back onto node N1, while node N11 loops back onto node N4. Node N9 and N10 are associated with the goal belief state.

3.2 The Planning Algorithm

The planning algorithm for conditional planning under partial observability is described in Figure 2. It takes in input the initial belief state and the goal belief state, while the domain representation is assumed to be globally available to the subroutines. The algorithm proceeds by incrementally constructing the finite acyclic prefix described above.

The algorithm relies on an extended data structure, stored in the $graph$ variable, that is a direct representation of the prefix. Each node is associated with a belief state and a path. Furthermore, the graph has a tree component that, for each node, allows to retrieve the sons, brothers, and father

```
HEURSYMCONDPLAN(I, G)
1   graph := MKINITIALGRAPH(I, G);
2   while (GRAPHROOTMARK(graph) ∉ {Success, Failure})
3       node := EXTRACTNODEFROMFRONTIER(graph);
4       if (SUCCESSPOOLYIELDSSUCCESS(node, graph))
5           MARKNODEASSUCCESS(node);
6           NODESETPLAN(node,RETRIEVEPLAN(node, graph));
7           PROPAGATESUCCESSONTREE(node,graph);
8           PROPAGATESUCCESSONEQCLASS(node,graph);
9       else
10          orexp := EXPANDNODEWITHACTIONS(node);
11          andexp := EXPANDNODEWITHOBSERVATIONS(node);
12          EXTENDGRAPHOR(orexp, node, graph);
13          EXTENDGRAPHAND(andexp, node, graph);
14          if (SONSYIELDSUCCESS(node))
15              MARKNODEASSUCCESS(node);
16              NODESETPLAN(node,BUILDPLAN(node));
17              PROPAGATESUCCESSONTREE(node, graph);
18              PROPAGATESUCCESSONEQCLASS(node, graph);
19          else if (SONSYIELDFAILURE(node))
20              MARKNODEASFAILURE(node, graph);
21              NODEBUILDFAILUREREASON(node, graph);
22              PROPAGATEFAILUREONTREE(node, graph);
23              PROPAGATEFAILUREONEQCLASS(node, graph);
24  end while
25  if (GRAPHROOTMARK(graph) = Success)
26      return GRAPHROOTPLAN(graph);
27  else
28      return Failure;
```

Fig. 2. The planning algorithm

nodes. The graph also contains a frontier of the nodes that have not yet been expanded. Finally, it contains the equivalence classes of nodes that share the same belief state. The graph is also annotated with a success pool, that contains the belief states for which a plan guaranteed to reach the goal has been found.

At line 1, the algorithm initializes the graph, by constructing the root node corresponding to the initial belief state, and the success pool with the goal belief state. Then, at lines 2-24, the iteration proceeds by selecting a node and expanding it, until a solution is found or the absence of a solution is detected. After the main loop, either a plan or a failure are returned (lines 25-28).

With the first step in the loop, at line 3, a node is selected for expansion, and is extracted from the frontier. The EXTRACTNODEFROMFRONTIER primitive embodies the selection criterion and is responsible for the style (and the effectiveness) of the search being carried out. Then, at line 4, we check whether the belief state associated to the selected node is entailed by the pool of previously solved belief states. The primitive SUCCESSPOOLYIELDSSUCCESS checks if there exists a node n_s, stored in the success pool, such that the associated belief state contains the belief state of the node being analyzed. If so, the algorithm takes care of the newly solved node. In particular, (the node stored in) $node$ is marked as success, and inserted in the success pool (line 5). Then, it is associated with the plan that has been previously computed for n_s (line 6). (As a consequence, the plan constructed by the algorithm is a DAG rather than a tree.) At line 7, the PROPAGATESUCCESSONTREE primitive propagates success backwards. If $node$ is the result of

an action expansion, the father node n_f is marked as success, and the open descendents of n_f are marked as removed and deleted from the frontier, since their expansion is no longer necessary. If $node$ is the result of an observation, the above process is carried out only if all the brothers of $node$ are already marked as success. In this case, n_f is associated with a conditional plan. If n_f is marked as success, the success propagation process is recursively iterated, until either an observation branching with at least one non-success node is reached, or the root of the graph is marked as success, in which case the problem is solved. Then, the success of $node$ is propagated to the equivalence class, i.e. to any other node n_e having the same belief state of $node$. This recursive success propagation is activated for each node in the equivalence class. This simplification can be seen as an optimization of the conceptually simple exploration of the tree-shaped search space.

If the success of $node$ is not entailed by the success pool, then the expansion of $node$ is attempted, computing the nodes resulting from possible actions (line 10) and observations (line 11). Let Bs be the belief state associated with $node$. Then, the result of EXPANDNODEWITHACTIONS, stored in variable $orexp$, is a list of belief state-action pairs $\langle Bs_\alpha \,.\, \alpha \rangle$, where Bs_α is the result of the execution of α in Bs. Similarly, EXPANDNODEWITHOBSERVATIONS returns a list of observation results, each composed of an observation mask, describing what variables are being observed, and a list of the resulting belief states, corresponding to the associated observation results. The graph extension steps, at lines 12-13, construct the nodes associated to the expansion, and add them to the graph, also doing the bookkeeping operations needed to update the frontier and the links between nodes. In particular, for each node, the associated status is computed. For instance, if a newly constructed node has a belief state that is already associated with a plan, then the node is marked as success[1]. Newly constructed nodes are also checked for loops, i.e. if they have an ancestor node with the same belief state then they are marked as failure.

If it is possible to state the success of $node$ based on the status of the newly introduced sons (primitive SONSYIELDSUCCESS at line 14), then the same operations at line 5-8 for success propagation are executed. At line 19, the SONSYIELDFAILURE primitive tries to state if a $node$ is a failure. This can happen, for instance, if no action is applicable in the associated belief state, and observations are non-informative (i.e. both $orexp$ and $endexp$ are empty). In this case, it is possible to store the failure of the node in such a way that it can be reused in the following search attempts. Notice however that, differently from success, a failure can not be associated with the belief state of the node, but it depends on the node path. For instance, in a subsequent search attempt it could be possible to reach a belief state with a non-cyclic path. Therefore, each belief state is associated with a set of belief states representing the failure reason. Intuitively, the failure reason contains the sets of belief states that caused a loop in all the search attempts originating from the belief state marked with failure.

Figure 3 describes a possible behavior of the algorithm for the example of Figure 1 as a sequence of tree expansion steps. Each line represents a (meaningful) step in the algorithm. The first column in the table contains the set of nodes in the frontier. The highlighted node is the one extracted for expansion. The second column contains a description of the nodes resulting from the expansion of the selected node. For instance, the expansion of node N2 gives either N4 and N5, or N6. The other columns contain the marking of the nodes, highlighted when they are introduced

[1] This operation is weaker than checking success entailment from the success pool. Although in principle it could be possible to access the success pool for each newly created node, we perform this operation only for the node being expanded since entailment is rather expensive.

Frontier	Sons	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	Expl.
N1	N2	**C**												
N2	(N4,N5),N6	C	**C**		**O**	**O**	**F**(1)							LD
N4, N5	N7,N8	C	C		**C**	O	F	**O**	**O**					
N5, N7, **N8**	N11	C	C		C	O	F	O	O			**F**(4)		LD
		C	C		C	O	F	O	**F**(4)			F		FPT
N5, **N7**	N10, N12	C	C		C	O	F	**C**	F		**S**	F	**O**	Goal
		C	C		**S**	O	F	**S**	F		S	F	**R**	SPT
		C	C		S	**S**	F	S	F		S	F	R	SPEC
		S	**S**		S	S	F	S	F		S	F	R	SPT

Fig. 3. The behavior of the algorithm for the example.

or changed, where C stands for “computing”, O for “open”, F for “failure”, R for “removed”, and S for “success”. Failure nodes are also annotated with the failure reason(s) associated with their belief state. The last column contains an explanation for the new marking of nodes. For instance, N6 is marked as failure because of loop detection (LD) with reason belief state 1. N8 are marked as failure because of a loop detection on belief state 4, while N4 is marked as failure during the subsequent failure propagation on tree (FPT), inheriting the same failure reason; N10 is marked success because it has a goal belief state; N7 and N4 are marked success during propagation of success on tree (SPT); N5 is marked success for propagation of success over equivalence class (SPEC); finally, N2 and N1 are marked as success by the last SPT. Notice that the algorithm does not expand non-informative observations (therefore N3 is not generated), and that the information propagation phases (either over the search tree or over equivalence classes) may prevent certain frontier nodes from being expanded (thus, N9 is not expanded). Plan reconstruction is left to the reader.

The algorithm presented above is guaranteed to terminate, based on the fact that the explored search space is finite. Upon termination, it either returns a solution plan, if the problem admits a solution, or a failure. Depending on the selection function, it is possible for the algorithm to explore the search space with different search styles, e.g. depth-first, breadth-first, or with other more sophisticated search strategies. By maintaining a frontier, search can be switched from one direction to another, considering, at each time, the most promising node among the ones on the frontier. The problem of finding an effective selection functions appears to be very hard. Given that the search is an and-or, and is carried out in belief space, several parameters could be used to determine what is a promising node, or a more promising expansion. A “bold” strategy, favoring actions over observations, would limit the number of subgoals resulting from “and” branching, but would give larger belief states to deal with. On the other hand, a “coward” strategy, observing whenever possible, may give smaller belief states, but generates a larger number of and nodes. The distance from the goal of each state in a belief state could also be of help. We have currently experimented with a simple structural selection function, that gives high scores to nodes having an equivalence class with a large intersection on the frontier. In the following we call HSEQC the search algorithm controlled by that selection function. Although the heuristic is extremely simple, it appears to work quite well.

4 BDD-based Implementation within the MBP Planner

The algorithm described above is integrated in MBP [CRT98; BCP+01]. MBP is a general planner for nondeterministic domains, based on a two-stages architecture. In the first stage, a general de-

	DFS							HSEQC						
	ST	PL	NBS	APL	TRB	BF	UvR	ST	PL	NBS	APL	TRB	BF	UvR
OER(2)	0.000	2	6	1.000	4	1.692	1 / 12	0.010	2	6	1.000	4	1.692	1 / 12
OER(3)	0.000	6	15	2.667	9	1.351	6 / 21	0.010	7	16	3.556	9	1.289	7 / 21
OER(4)	0.010	16	33	10.000	16	1.117	13 / 39	0.050	10	27	6.125	16	1.190	11 / 35
OER(5)	0.020	23	46	15.360	25	1.078	17 / 54	0.160	15	49	10.960	25	1.122	23 / 55
OER(6)	0.020	31	60	23.000	36	1.070	29 / 68	0.350	19	58	13.556	36	1.103	21 / 70
OER(7)	0.030	47	82	35.306	49	1.033	27 / 94	0.700	24	93	17.020	49	1.099	46 / 97
OER(8)	0.030	57	98	46.781	64	1.037	43 / 110	1.290	31	124	21.062	64	1.082	70 / 116
OER(9)	0.070	79	126	63.259	81	1.018	37 / 142	2.280	34	154	23.506	81	1.077	92 / 149
OER(10)	0.090	91	144	78.640	100	1.023	57 / 160	3.420	41	181	27.720	100	1.068	119 / 173
OER(11)	0.080	119	178	99.223	121	1.011	47 / 198	4.520	45	202	31.736	121	1.093	158 / 185
OER(12)	0.120	133	198	118.542	144	1.015	71 / 218	6.390	49	241	33.458	144	1.063	183 / 225
OER(13)	0.130	167	238	143.195	169	1.008	57 / 262	8.160	56	261	38.343	169	1.063	240 / 235
OER(14)	0.160	183	260	166.469	196	1.011	85 / 284	11.090	64	297	41.827	196	1.095	253 / 297
OER(15)	0.190	223	306	195.173	225	1.006	67 / 334	14.650	66	349	44.907	225	1.061	320 / 293
OER(16)	0.210	241	330	222.414	256	1.008	99 / 358	17.110	69	371	44.508	256	1.065	355 / 373
OER(17)	0.260	287	382	255.156	289	1.004	77 / 414	21.930	74	402	50.567	289	1.053	429 / 354
OER(18)	0.300	307	408	286.370	324	1.007	113 / 440	27.360	80	460	54.790	324	1.051	539 / 374
OER(19)	0.340	359	466	323.141	361	1.003	87 / 502	33.030	84	447	55.424	361	1.064	567 / 431
OER(20)	0.380	381	494	358.335	400	1.005	127 / 530	102.040	89	559	61.115	400	1.046	699 / 427

Table 1. Results for the Original Empty Room

scription of a nondeterministic domain is processed, and the corresponding internal representation is built. Then, for the domain description being processed, MBP allows for conditional planning under full observability [CRT98], also considering temporally extended goals [PT01], for conformant planning [CR00; BCR01], and for partial observability based on DFS [BCRT01]. MBP is based on the use of symbolic model checking techniques [McM93]. In particular, we rely on Binary Decision Diagrams (BDD) [Bry92], that allow for a compact representation of sets of states, and an efficient analysis of the search space. For lack of space, we do not describe BDDs, and the use of symbolic model checking techniques to planning (see [CR00] for an introduction). We simply remark that the problem of searching in the space of beliefs requires a significant departure from the standard symbolic model checking techniques, where the search space is $\mathcal{S}$, i.e. the set of states, and one single BDD is used to represent a portion of the visited search space. Here, each visited belief state is represented by a unique BDD, while a hashing structure is used to efficiently implement the marking mechanism described in previous section. The machinery used to tackle partially observable planning is a generalization of the one used to do conformant planning [BCR01] as search in the space of belief states: the main difference is that deterministic search is enough for conformant planning, while partially observable planning requires and-or search.

The primitives for the expansion of nodes by means of actions and of observations used by the algorithm presented in previous section are based on the same primitives of the implementation of the DFS algorithm, and take full advantage of BDDs (see [BCRT01]). With respect to the algorithm in Figure 2, that is presented at a certain level of detail for the sake of clarity, additional mechanisms can be activated to increase efficiency. For instance, a preliminary step of conformant planning can be carried out backwards from $\mathcal{G}$ towards $\mathcal{I}$, in order to enlarge the pool of target belief states. This is sometimes convenient given that conformant planning can be carried out very efficiently in this framework (see [CR00; BCR01]). The drawback, however, is that a long sequential plan can be obtained where a "smart" conditional plan is available. Furthermore, the success pool is maintained in form of a list of *maximal* successful belief states. This avoids to test for inclusion with all the successful visited belief states, and minimizes the size of the list. This idea is based on the lattice structure induced by set inclusion. Finally, when a success plan is found, it is sometimes very useful to backward simulate the plan, in order to identify the maximal belief state where it is applicable and for which it guarantees success. This operation tries to propagate the results of a successful search branch.

	DFS							HSEQC						
	ST	PL	NBS	APL	TRB	BF	UvR	ST	PL	NBS	APL	TRB	BF	UvR
VER(2)	0.000	2	2	2.000	1	1.000	0 / 2	0.010	2	2	2.000	1	1.000	0 / 2
VER(3)	0.000	7	8	4.500	2	1.111	1 / 8	0.000	3	4	2.500	2	1.200	1 / 4
VER(4)	0.000	8	9	7.500	2	1.067	1 / 9	0.010	5	6	4.500	2	1.143	1 / 6
VER(5)	0.010	18	19	17.500	2	1.029	1 / 19	0.030	5	6	4.500	2	1.143	1 / 6
VER(6)	0.010	22	23	21.500	2	1.023	1 / 23	0.070	9	10	8.500	2	1.062	1 / 10
VER(7)	0.080	26	27	25.500	2	1.020	1 / 27	0.200	11	12	10.500	2	1.053	1 / 12
VER(8)	0.110	30	31	29.500	2	1.017	1 / 31	0.430	13	14	12.500	2	1.043	1 / 14
VER(9)	4.210	34	35	33.500	2	1.015	1 / 35	0.220	17	18	16.500	2	1.033	1 / 18
VER(10)	14.310	56	57	55.500	2	1.009	1 / 57	0.260	19	20	18.500	2	1.029	1 / 20
VER(11)	M.O.							0.530	19	20	18.500	2	1.030	1 / 20
VER(12)								0.530	21	22	20.500	2	1.027	1 / 22
VER(19)								2.470	35	36	34.500	2	1.016	1 / 36
VER(20)								3.190	37	38	36.500	2	1.015	1 / 38

Table 2. Results for the Variated Empty Room

	DFS							HSEQC						
	ST	PL	NBS	APL	TRB	BF	UvR	ST	PL	NBS	APL	TRB	BF	UvR
ERS(6,7)	0.010	20	22	19	4	1.041	2 / 22	0.050	8	16	7	4	1.097	3 / 16
ERS(6,8)	0.460	20	22	19	4	1.041	2 / 22	0.050	8	16	7	4	1.097	3 / 16
ERS(6,9)	T.O.							0.050	8	16	7	4	1.097	3 / 16
ERS(6,10)								0.040	8	16	7	4	1.097	3 / 16
ERS(6,800)								0.050	8	16	7	4	1.097	3 / 16
ERS(6,1000)								0.050	8	16	7	4	1.097	3 / 16
ERS(20,20)	0.080	174	176	173	4	1.004	2 / 176	0.190	20	28	19	4	1.038	3 / 28
ERS(20,21)	0.120	174	176	173	4	1.004	2 / 176	0.190	20	28	19	4	1.038	3 / 28
ERS(20,22)	T.O.							0.190	20	28	19	4	1.038	3 / 28
ERS(20,25)								0.190	20	28	19	4	1.038	3 / 28
ERS(20,800)								0.210	20	28	19	4	1.038	3 / 28
ERS(20,1000)								0.210	20	28	19	4	1.038	3 / 28

Table 3. Results for the Empty Room with Sink

5 Experimental Evaluation

We compare the heuristic-symbolic HS approach described in this paper (HSEQC) with the DFS approach of [BCRT01]. The comparison with other systems for planning under partial observability is discussed in next section. The experiments were run on a Pentium II 300MHz with 512Mb of memory running Linux, fixing a memory limit of 450Mb and timeout to 1 hour CPU. When reporting the results, we write T.O. for "time out" and M.O. for "memory out". We consider several parameterized problem classes. In order to focus on the properties of the algorithms, we report the search time (ST, in seconds), and not the time needed to preprocess the domain descriptions. We also report information on the DAG structure of the returned plan: the maximum and the average plan length (MPL and APL); the number of belief states associated with the execution of the plan (NBS); the branching factor (BF); the total number of Relevant Branches (TRB) associated with the plan (reuse of plans can result in branches that are never followed for some belief states); the Unreused vs Reused subplans (UvR), that gives an idea of the "remerging" in the plan.

The problems considered in the comparison are the original empty room (OER) and the MAZE navigation problems, and RING problem, from [BCRT01]. Furthermore, we consider two variations of the empty room domain, called VER and ERS. Notice that, in the experimental evaluation, the same algorithm configuration was used for the all the problems (e.g. the preliminary call to conformant planning was not activated, a structural, problem independent selection function was used). Performance and results can be greatly improved by "tuning" the search on the problem at hand. In the OER, the problem is moving to a given corner position from *any* position in a $n \times n$ empty room. The results are reported in Table 1. DFS is extremely fast. This can be explained

	DFS							HSEQC						
	ST	PL	NBS	APL	TRB	BF	UvR	ST	PL	NBS	APL	TRB	BF	UvR
MAZE(3)	0.000	6	9	3.857	7	1.429	5 / 21	0.000	6	9	3.857	7	1.429	5 / 21
MAZE(5)	0.010	12	9	7.529	17	1.265	26 / 47	0.040	12	9	7.529	17	1.265	26 / 47
MAZE(7)	0.040	27	36	14.226	31	1.191	38 / 91	0.140	25	27	13.581	31	1.272	52 / 84
MAZE(9)	0.060	41	29	23.388	49	1.148	117 / 128	0.240	45	32	24.776	49	1.204	119 / 112
MAZE(11)	0.070	55	65	29.310	71	1.096	128 / 179	0.880	55	84	29.479	71	1.071	75 / 211
MAZE(13)	0.080	75	55	41.794	97	1.104	173 / 228	1.320	75	64	42.103	97	1.126	266 / 205
MAZE(15)	0.140	87	143	53.504	127	1.076	205 / 298	5.280	89	156	52.260	127	1.055	140 / 313
MAZE(17)	0.220	114	212	62.398	161	1.049	268 / 339	2.760	114	171	63.453	161	1.056	333 / 326
MAZE(19)	0.270	169	202	79.623	199	1.043	383 / 449	11.880	171	160	79.935	199	1.050	261 / 493
MAZE(21)	0.350	157	345	91.983	241	1.036	459 / 488	14.730	161	351	92.017	241	1.028	265 / 512
MAZE(23)	0.410	184	401	99.314	287	1.036	528 / 556	17.390	184	301	99.167	287	1.028	305 / 597
MAZE(25)	0.580	194	386	104.522	337	1.028	588 / 653	22.190	199	295	104.617	337	1.055	387 / 653
MAZE(27)	0.740	204	476	116.780	391	1.028	655 / 719	26.690	205	431	116.703	391	1.032	530 / 759
MAZE(29)	0.840	266	581	153.189	449	1.023	886 / 836	34.210	267	549	153.024	449	1.024	690 / 865
MAZE(31)	1.100	292	499	168.706	511	1.024	1098 / 89	46.620	293	552	168.945	511	1.024	787 / 951
MAZE(33)	1.580	324	659	177.764	577	1.019	1152 / 10	63.610	344	654	179.047	577	1.021	849 / 116
MAZE(35)	1.560	352	636	197.689	647	1.019	1273 / 11	64.420	343	565	197.427	647	1.018	969 / 120
MAZE(37)	1.850	381	939	215.448	721	1.017	1374 / 12	81.070	379	761	215.933	721	1.019	1188 / 13
MAZE(39)	2.010	363	933	213.203	799	1.020	1671 / 12	113.550	367	747	213.488	799	1.018	990 / 145
MAZE(41)	2.490	480	970	257.140	881	1.015	1840 / 14	106.880	481	810	258.059	881	1.015	1249 / 15
MAZE(43)	2.960	501	836	264.820	967	1.015	2010 / 15	121.670	493	743	266.681	967	1.016	1729 / 17
MAZE(45)	4.000	549	969	290.155	1057	1.014	2146 / 17	130.340	567	518	292.384	1057	1.017	1812 / 19
MAZE(47)	4.550	544	1130	330.248	1151	1.013	2297 / 19	175.570	563	1232	330.619	1151	1.013	1635 / 19
MAZE(49)	4.540	682	1319	374.328	1249	1.011	2397 / 20	181.870	697	1309	376.051	1249	1.012	2031 / 20
MAZE(51)	4.990	760	1509	408.448	1351	1.011	3117 / 22	189.680	761	1447	409.754	1351	1.012	2198 / 23

Table 4. Results for the Maze

	DFS							HSEQC						
	ST	PL	NBS	APL	TRB	BF	UvR	ST	PL	NBS	APL	TRB	BF	UvR
RING(2)	0.010	7	1	4.333	9	1.343	5 / 18	0.010	6	14	3.889	9	1.341	9 / 16
RING(3)	0.020	15	1	7.815	27	1.335	25 / 41	0.010	8	14	5.000	27	1.433	5 / 17
RING(4)	0.060	24	1	11.580	81	1.282	64 / 82	0.020	11	19	7.000	81	1.430	7 / 23
RING(5)	0.150	32	1	16.872	243	1.297	161 / 146	0.040	15	25	10.000	243	1.425	9 / 30
RING(6)	0.470	53	1	22.372	729	1.251	359 / 300	0.060	18	30	12.000	729	1.427	11 / 36
RING(7)	1.210	64	1	29.031	2187	1.276	801 / 519	0.070	21	35	14.000	2187	1.428	13 / 42
RING(8)	4.330	87	1	36.152	6561	1.242	1718 / 10	0.110	24	40	16.000	6561	1.428	15 / 48
RING(9)	18.290	100	1	45.170	19683	1.245	3604 / 19	0.150	27	45	18.000	19683	1.429	17 / 54
RING(10)	68.510	129	1	53.154	59049	1.227	7528 / 38	0.190	30	50	20.000	59049	1.429	19 / 60
RING(11)	273.800	155	1	63.739	177147	1.229	15535 / 7	0.250	33	55	22.000	177147	1.429	21 / 66
RING(12)	1118.520	182	1	75.083	531441	1.207	31852 / 1	0.300	36	60	24.000	531441	1.429	23 / 72
RING(13)	4415.530	219	1	86.010	1594323	1.204	65034 / 2	0.390	39	65	26.000	1594323	1.429	25 / 78
RING(14)	T.O.							0.480	42	70	28.000	4782969	1.429	27 / 84
RING(16)								0.670	48	80	32.000	43046721	1.429	31 / 96
RING(48)								18.480	144	240	96.000	797664430768725143060 48	1.429	95 / 288
RING(50)								20.810	150	250	100.000	717897987691852578422784	1.429	99 / 300

Table 5. Results for the Ring

considering that (the implementation of) DFS tends to privilege observations over actions. Given that the starting point is completely unspecified, in most cases observations are informative. The maximal length of the returned plan, however, is up to five times longer than the maximal length of the plan constructed by HSEQC. In the VER, the problem is moving to the same given position from an initial belief state containing two states in the center of the room. Table 2 shows that HSEQC significantly outperforms DFS. Differently from DFS, the search strategy of HSEQC does not try to re-enter the same portion of the search space over and over from different paths. The plans returned by HSEQC are also significantly shorter. In Table 3 we report the results for the ERS(g,n) (ER with a Sink). The idea is that, in a room of size n, a portion of the room (size $n - g$) is a sink, i.e. once entered, it can not be left. The tackled problem is solvable. While HSEQC is able to reconsider potentially bad initial choices (e.g. entering the sink), DFS is bound to the complete exploration of the sink, once it is entered. Table 4 shows that DFS is extremely effective with the MAZE problem. Most likely, this is due to the nature of the problem, that severely constraints the search space. The plans constructed by the two algorithms are indeed very similar. As reported in [BCRT01], the RING problem is hard for DFS. Table 5 shows that HSEQC is able to tackle very large instances of the problem, since it focuses on remerging attempts. Notice also that the plans

constructed by HSEQC are extremely close to optimal, while the plans constructed by DFS are very far from that.

In summary, DFS appears to behave extremely well with highly constrained problems, but suffers from the inability to deal with bad initial choices. Heuristic-symbolic search, on the other hand, even with a simple selection function, is able to solve problems that are out of reach for DFS.

6 Related Work and Conclusions

In this paper we have presented a new algorithm for conditional planning under partial observability. The planning algorithm is based on the exploration of a (possibly cyclic) and-or graph induced by the domain. The algorithm departs from heuristic search algorithms like AO*, that are based on the assumption that and-or search graphs are acyclic. Given the exhaustive style of the exploration, the algorithm can decide whether the problem admits an acyclic solution, i.e. a plan guaranteed to reach the goal in a finite number of steps. The algorithm takes full advantage from the combination of BDD-based symbolic model checking techniques with the heuristic style of the search. The experimental evaluation shows that, even with a problem independent, structural selection function, the HS approach can outperform the DFS approach both in terms of efficiency and in the quality of the returned plans.

A comparison against other conditional planners follows from the experimental evaluation in [BCRT01], where the DFS algorithm outperforms the SGP and GPT conditional planners. Another interesting system is QBFPLAN [Rin99a], that extends the SAT-based approach to planning to the case of nondeterministic domains. The planning problem is reduced to a QBF satisfiability problem, that is then given in input to an efficient solver [Rin99b]. QBFPLAN relies on a symbolic representation, but the approach seems to be limited to plans with a bounded execution length. The search space is significantly reduced by providing the branching structure of the plan as an input to the planner. The problem of planning under partial observability has been deeply investigated in the framework of Partially Observable MDP (see, e.g., [CKL94; HZ98; PB00]). Methods that interleave planning and execution [KS98; GN93] can be considered alternative (and orthogonal) approaches to the problem of planning off-line with large state spaces. However, these methods cannot guarantee to find a solution, unless assumptions are made about the domain. For instance, [KS98] assumes “safely explorable domains” without cycles. [GN93] describes an off-line planning algorithm based on a breadth-first search on an and-or graph. The paper shows that the version of the algorithm that interleaves planning and execution is more efficient than the off-line version, both theoretically and experimentally.

Future research will be directed to the definition of more effective preprocessing techniques and heuristic functions, with the goal to obtain “smarter” behaviors from the heuristic-symbolic algorithm. For instance, an accurate selection function should be able to take into account, in addition to the distance from the goal, the degree of uncertainty in the belief state being analyzed, and also the point in the and-or search. Another direction of future research is the extension of the partially observable approach presented in this paper to strong cyclic solutions [CRT98], and for temporally extended goals [KBSD97; PT01].

References

[BCP+01] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *Proc. of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, Seattle, August 2001. To appear.

[BCR01] P. Bertoli, A. Cimatti, and M. Roveri. Heuristic Search + Symbolic Model Checking = Efficient Conformant Planning. In *Proc. 7^{th} International Joint Conference on Artificial Intelligence (IJCAI-01)*. AAAI Press, August 2001.

[BCRT01] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc. 7^{th} International Joint Conference on Artificial Intelligence (IJCAI-01)*. AAAI Press, August 2001.

[BG00] B. Bonet and H. Geffner. Planning with Incomplete Information as Heuristic Search in Belief Space. In S. Chien, S. Kambhampati, and C.A. Knoblock, editors, 5^{th} *International Conference on Artificial Intelligence Planning and Scheduling*, pages 52–61. AAAI-Press, April 2000.

[Bry92] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[CKL94] A. Cassandra, L. Kaelbling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proc. of AAAI-94*. AAAI-Press, 1994.

[CR00] A. Cimatti and M. Roveri. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research (JAIR)*, 2000. Accepted for publication. To appear.

[CRT98] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceeding of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, 1998. AAAI-Press. Also IRST-Technical Report 9801-10, Trento, Italy.

[GN93] M. Genesereth and I. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*, 1993.

[HZ98] E. A. Hansen and S. Zilberstein. Heuristic search in cyclic and-or graphs. In *Proceedings of the National Conference on Artificial Intelligence*, 1998.

[KBSD97] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.

[KS98] S. Koenig and R. Simmons. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling*, 1998.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.

[PB00] P. Poupart and C. Boutilier. Value-directed belief state approximation for pomdps. In *UAI-2000*, 2000.

[PC96] L. Pryor and G. Collins. Planning for Contingency: a Decision Based Approach. *J. of Artificial Intelligence Research*, 4:81–120, 1996.

[PT01] M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proc. 7^{th} International Joint Conference on Artificial Intelligence (IJCAI-01)*. AAAI Press, August 2001.

[Rin99a] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intellegence Research*, 10:323–352, 1999.

[Rin99b] J. Rintanen. Improvements to the Evaluation of Quantified Boolean Formulae. In T. Dean, editor, *16th Iinternational Joint Conference on Artificial Intelligence*, pages 1192–1197. Morgan Kaufmann Publishers, August 1999.

[SW98] David E. Smith and Daniel S. Weld. Conformant graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 889–896, Menlo Park, July 26–30 1998. AAAI Press.

[TK00] C. Tovey and S. Koenig. Gridworlds as testbeds for planning with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.

[WAS98] Daniel S. Weld, Corin R. Anderson, and David E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 897–904, Menlo Park, July 26–30 1998. AAAI Press.