

# Fast and Informed Action Selection for Planning with Sensing

Alexandre Albore<sup>1</sup>, Héctor Palacios<sup>1</sup>, and Hector Geffner<sup>2</sup>

<sup>1</sup> Universitat Pompeu Fabra  
Passeig de Circumvalació 8  
08003 Barcelona Spain

<sup>2</sup> ICREA & Universitat Pompeu Fabra  
Passeig de Circumvalació 8  
08003 Barcelona Spain

**Abstract.** Consider a robot whose task is to pick up some colored balls from a grid, taking the red balls to a red spot, the blue balls to a blue spot and so on, one by one, without knowing either the location or color of the balls but having a sensor that can find out both when a ball is near. This problem is simple and can be solved by a domain-independent contingent planner in principle, but in practice this is not possible: the size of any valid plan constructed by a contingent planner is exponential in the number of observations which in these problems is very large. This doesn't mean that planning techniques are of no use for these problems but that building or verifying complete contingent plans is not feasible in general. In this work, we develop a domain-independent action selection mechanism that does not build full contingent plans but just chooses the action to do next in a closed-loop fashion. For this to work, however, the mechanism must be both fast and informed. We take advantage of recent ideas that allow delete and precondition-free contingent problems to be converted into conformant problems, and conformant problems into classical ones, for mapping the action selection problem in contingent planning into an action selection problem in classical planning that takes sensing actions into account. The formulation is tested over standard contingent planning benchmarks and problems that require plans of exponential size.

## 1 Introduction

Contingent planning is concerned with the problem of achieving goals in the presence of incomplete information and sensing actions [1,2]. This is one of the most general problems considered in the area of planning and one of the hardest [3,4]. In the last few years, significant progress has been achieved resulting in a variety of contingent planners that can solve large and non-trivial problems, usually by casting the contingent planning problem as an AND/OR search over belief space [5] guided by effective heuristics and belief representations [6,7,8].

In spite of this progress, however, a large obstacle remains: there are many problems involving incomplete information and sensing actions whose solutions

have exponential size. Thus constructing or even verifying plans for such problems would take exponential time. This situation is different than in classical or conformant planning where exponential length solutions are the exception. Contingent plans of exponential size follow naturally from situations where the number of observations that needs to be done is linear in the size of the problem.<sup>3</sup>

The goal of this work is to use domain-independent planning techniques for dealing with such problems. However, rather than aiming at constructing full contingent plans, we aim at an effective *action selection mechanism* that chooses the action to do next in a closed-loop fashion. For this, we will move to the ‘knowledge-level’ [9], represent sensing actions as normal deterministic actions and map the *action selection problem in planning with sensing* into an *action selection problem in classical planning*, a problem that has good and well known solutions.

We take advantage of two recent ideas: the reduction of contingent planning into conformant planning that is obtained when deletes are relaxed and preconditions are moved in as conditions [6], and the reduction of conformant into classical planning obtained by the addition of conditionals and simple epistemic formulas represented as literals [10]. The two reductions in a row, however, do not suffice as sensing actions are ignored. We will thus extend the resulting classical encoding of a contingent problem  $P$  with a suitable representation of the sensing actions. On the one hand we define an *execution model*  $X(P)$  where sensing actions are represented as actions with non-deterministic effects  $Kx|K\neg x$ , where  $x$  is the boolean variable being observed and  $KL$  represents that  $L$  is *known*; on the other, we define an *heuristic model*  $H(P)$  where these effects are relaxed into deterministic effects of the form  $Mx \wedge M\neg x$ , where  $ML$  represents that  $L$  *may be known*. In addition, while preconditions  $L$  of  $P$  must be known with certainty in  $X(P)$  and are thus modeled as  $KL$ , in the heuristic model  $H(P)$  they must be contingently known only and are modeled as  $ML$ .

The proposed *Closed-Loop Greedy planner (CLG)* then works as follows. In current state of the execution model  $X(P)$ , which is always fully known, an action in  $X(P)$  is selected by using the heuristic model  $H(P)$  which is a classical planning problem. The selected action is then applied in  $X(P)$ , its effect is observed, and the new state of the execution model is computed, from which the loop resumes until reaching a state that is a goal in  $X(P)$ . In CLG, the execution models keeps track of the belief state in the form of a set of literals at the knowledge level (details below), while the heuristic model selects the action to do next. CLG can be used and we will use it also for computing full contingent plans. For this, all the effects of the non-deterministic (sensing) actions applied need to be considered, and their responses cached.

---

<sup>3</sup> It must be said though that problems such as the one above, where balls in a grid are to be located and placed in their corresponding destination, admit compact solutions in languages, closer to the ones used in programming, that accommodate loops and subroutines. Current languages for contingent planning, however, do not accommodate such constructs. Dealing with such constructs in domain-independent planning is a hard open challenge, as hard indeed as automatic programming.

The rest of the paper is organized as follows: we start with the contingent problem  $P$ , define the translation  $K(P)$  of the conformant fragment of  $P$  (no sensing actions) into classical planning, consider the execution and heuristic models  $X(P)$  and  $H(P)$  that result from adding to  $K(P)$  an encoding of the sensing actions, make the working of the CLG planner precise, and test it over a number of problems.

## 2 The Contingent Planning Problem $P$

We consider a planning language that extends Strips with conditional effects, a possibly uncertain initial situation, and sensing actions. More precisely, a contingent planning problem is a tuple  $P = \langle F, O, I, G \rangle$  where  $F$  stands for the fluent symbols in the problem,  $O$  stands for the set of actions or operators  $a$ ,  $I$  is a set of *clauses* over  $F$  defining the initial situation, and  $G$  is a set of literals over  $F$  defining the goal.

A normal action  $a$  has a precondition given by a set of fluent literals, and a set of conditional effects  $C \rightarrow L$  where  $C$  is a set of fluent literals and  $L$  is a literal. The sensing actions  $a$ , on the other hand, have a single unconditional effect  $obs(x)$  where  $x$  is a fluent symbol, meaning that after doing action  $a$  the truth value of  $x$  will be known. Sensing actions can have preconditions as any other actions but for simplicity we assume that they have no other effects.

We refer to the conditional effects  $C \rightarrow L$  of an action  $a$  as the *rules* associated with  $a$ , and sometimes write them as  $a : C \rightarrow L$ . Also, we use the expression  $C \wedge X \rightarrow L$  to refer to rules with literal  $X$  in their bodies. In both cases,  $C$  may be empty. Last, when  $L$  is a literal, we take  $\neg L$  to denote the complement of  $L$ . The ‘conformant fragment’ of  $P$  will mean the contingent problem  $P$  with the sensing actions removed.

## 3 The Conformant Translation $K(P)$

We have recently shown elsewhere that it is possible to convert conformant problems  $P$  into classical problems  $K(P)$  so that solutions from  $P$  can be extracted from the solutions computed by a classical planner over  $K(P)$  [10]. This translation is not complete but has been shown to be quite effective [11]. More recently, this translation has been simplified and generalized into a translation scheme  $K_{T,M}(P)$  where  $T$  is a set of *tags* and  $M$  is a set of *merges* [12]. A tag  $t$  is set of literals in  $P$  whose status in the initial situation  $I$  of  $P$  is not known. A merge  $m$  is a collection of tags  $t$  such that one of them must be true in  $I$ . The translation that maps the conformant problem  $P$  into a classical problem  $K_{T,M}(P)$  replaces the literals  $L$  in  $P$  by literals  $KL/t$  for each  $t \in T$ , whose intuitive meaning is that ‘if  $t$  is true in the initial situation,  $L$  is true’. In addition, extra actions, called merge actions, allow the derivation of the literal  $KL$ , i.e.  $KL/t$  with the empty tag  $t$ , when  $KL/t'$  has been obtained for each tag  $t'$  in a merge.

If  $P = \langle F, O, I, G \rangle$  is the conformant problem, then the classical problem  $K_{T,M}(P) = \langle F', I', O', G' \rangle$  is given as:

$$\begin{aligned}
F' &= \{KL/t, K\neg L/t \mid L \in F \text{ and } t \in T\} \\
I' &= \{KL/t \mid \text{if } I \models t \supset L\} \\
G' &= \{KL \mid L \in G\} \\
O' &= \{a : KC/t \rightarrow KL/t, a : \neg K\neg C/t \rightarrow \neg K\neg L/t \mid a : C \rightarrow L \text{ in } P\} \cup \\
&\quad \left\{ \bigwedge_{t \in m} KL/t \rightarrow KL \mid L \in F \text{ and } m \in M \right\}
\end{aligned}$$

with  $KL$  a precondition of action  $a$  in  $K_{T,M}(P)$  if  $L$  is a precondition of  $a$  in  $P$ .

The intuition behind the translation is simple: first,  $KL/t$  is true in  $I'$  iff  $t \supset L$  follows from  $I$ . This removes all uncertainty from  $I'$ . Then  $KL$  is a goal in  $G'$  iff  $L$  is a goal in  $G$ . Also, to ensure soundness, each conditional effect  $a : C \rightarrow L$  in  $P$  maps, not only into the **supporting rule**  $a : KC/t \rightarrow KL/t$  but also into the **cancellation rule**  $a : \neg K\neg C/t \rightarrow \neg K\neg L/t$  that guarantees that  $K\neg L/t$  is deleted (prevented to persist) when action  $a$  is applied and  $C/t$  is not known to be false. The expressions  $KC$  and  $\neg K\neg C$  for  $C = L_1 \wedge \dots \wedge L_n$  are used as abbreviation of the formulas  $KL_1 \wedge \dots \wedge KL_n$ , and  $\neg K\neg L_1 \wedge \dots \wedge \neg K\neg L_n$ . Last, the **merge actions** yield  $KL$  when  $KL/t$  is true for each  $t$  in a merge  $m \in M$ .

The translation scheme  $K_{T,M}(P)$  is always sound, meaning that the classical plans that solve  $K_{T,M}(P)$  yield valid conformant plans for  $P$  (by just dropping the merge actions). On the other hand, the complexity and the completeness of the translation depend on the choice of tags and merges  $T$  and  $M$ . The  $K_i(P)$  translation, where  $i$  is a non-negative integer, is a special case of the  $K_{T,M}$  translation where the tags  $t$  are restricted to contain at most  $i$  literals. By a suitable choice of the merges  $M$ , we show in [12] that the  $K_i(P)$  translation for  $i = 1$  is *complete* for almost all of the conformant benchmarks. In this translation,  $t \in T$  iff  $t$  is the empty tag or a singleton  $\{L\}$  for an uncertain literal  $L$  in  $I$ , and  $M$  is the set of non-unit clauses in  $M$ . We assume this translation below and we refer to it as  $K_1(P)$  or simply as  $K(P)$ . This is the translation that underlies the conformant planner  $T_0$ , winner of the Conformant Track of the recent International Planning Competition [11].

For the sake of simplicity, from now on and when  $t$  is the empty tag  $t = \{\}$  and the singleton tag  $t' = \{L'\}$ , we write  $KL/t$  and  $KL/t'$  as  $KL$  and  $KL/L'$  respectively.  $KL$  represents that ‘ $L$  is known to be true with certainty’, while  $KL/L'$ , that ‘it is known with certainty that if  $L'$  is true initially,  $L$  is true’.

## 4 The Execution Model $X(P)$

The execution model  $X(P)$  for the CLG planner is the union of a translation of the ‘conformant fragment’ of  $P$  into a classical problem, and a suitable encoding of the sensing actions. Both parts are expressed in the language of the epistemic conditionals  $K/t$  of the translation above.

#### 4.1 The Classical Part $K^c(P)$

The classical part  $K^c(P)$  in  $X(P)$  is the translation above applied to the ‘conformant fragment’ of  $P$  extended with a set of deductive rules, encoded as actions with no preconditions and unique conditional effects of the form:

1.  $KL/t \wedge K\neg L \rightarrow K\neg t$
2.  $\bigwedge_{t \in m} (KL/t \vee K\neg t) \rightarrow KL$

This extension is needed because, while in conformant planning one reasons only ‘forward’ in time, in a contingent setting one must reason both ‘forward’ and ‘backward’. In particular, if a tag  $t$  cannot be shown to be false in  $I$ , no conformant plan will ever make it false. On the other hand, a tag  $t$  may be inferred to be false or true in contingent planning by simply doing actions and gathering observations. Many ‘identification’ tasks have this form: one needs to act and observe in order to identify a static but hidden state.

In the head  $K\neg t$  of the first deductive rule,  $t$  refers to the value of the tag  $t$  in the *initial situation* only. That is, if the rule is applied in a plan after several actions and  $t = L$ , then the inference that  $L$  is false refers to the initial situation and not to the situation that follows the action sequence. This distinction is irrelevant if  $L$  is a *static* literal whose value in the initial situation cannot change, but is relevant otherwise. With this in mind, we combine the use of these deductive rules implemented as actions, with a simple transformation that makes all literals in tags *static*. If  $L$  is not a static literal, then we create a *static copy*  $L_0$  of  $L$  by adding the equivalence  $L_0 \equiv L$  in  $I$ , so that  $L_0$  has the same value as  $L$  in the initial situation but does not change as it is not affected by action action. The tags are then limited to such static literals.

#### 4.2 The Sensing Part $K^o(P)$

The sensing actions  $a : obs(x)$  in the contingent problem  $P$  are translated into a set  $K^o(P)$  of non-deterministic actions

$$a : \neg Kx \wedge \neg K\neg x \rightarrow Kx \mid K\neg x$$

that capture their effects directly at the ‘knowledge level’ [9] making one of the fluents  $Kx$  or  $K\neg x$  true. We make such effects conditional on not knowing the value of  $x$ , as we do not want these rules to set a true  $KL$  literal into a false one. In addition, for each precondition  $L$  of  $a$  in  $P$ , we set the literal  $KL$  as a precondition of  $a$  in  $K^o(P)$ .

Like  $P$ , the execution model  $X(P) = K^c(P) + K^o(P)$  is a contingent planning problem, and due to the soundness of the translation, solutions to  $X(P)$  encode solutions to  $P$  (although not the other way around, as the translation is not complete). Yet, while  $P$  involves incomplete information and sensing actions,  $X(P)$  being at the ‘knowledge-level’ features full information (all literals are known) and no sensing actions. The model  $X(P)$ , on the other hand, features actions that are non-deterministic. In order to solve  $X(P)$ , and hence  $P$ , we consider a relaxation of  $X(P)$  that removes this non-determinism and results in a classical problem that is used for selecting the actions in the planner.

## 5 Heuristic Model $H(P)$

The basic change in the transition from the execution model  $X(P)$  to the heuristic model  $H(P)$  is the transformation of the *non-deterministic actions*

$$a : \neg Kx \wedge \neg K\neg x \rightarrow Kx | K\neg x$$

that arise from sensing actions into *deterministic actions*:

$$a : \neg Kx \wedge \neg K\neg x \rightarrow Mx \wedge M\neg x$$

where  $ML$  is an ‘epistemic’ literal aimed at expressing *contingent knowledge*: knowledge that may be obtained along some but not necessarily all execution branches, and hence which is weaker than  $KL$ .

By *relaxing* the actions with non-deterministic effects  $Kx | K\neg x$  in  $X(P)$  into actions with deterministic effects  $Mx \wedge M\neg x$  in  $H(P)$ , a *classical problem* is obtained. The rest of heuristic model  $H(P)$  includes *deductive rules* for the  $ML$  literals similar to the rules above for the  $KL$  literals, and the use of such literals in the *action preconditions* in place of the  $KL$  literals.

Deductive rules, similar to the ones for  $K$ , allow us also to expand the literals  $L$  that are assumed to be ‘contingently known’:

1.  $KL \rightarrow ML$
2.  $KL/t \wedge M\neg L \rightarrow M\neg t$
3.  $KL/t \wedge Mt \rightarrow ML$
4.  $\bigwedge_{t' \in m/t} M\neg t' \rightarrow Mt$

In addition, rules  $a : MC \rightarrow ML$  are added to  $H(P)$  for rules  $a : C \rightarrow L$  in  $P$ .

Likewise, every precondition  $L$  of an action  $a$  in  $P$  is *copied* as a condition in the body of  $C$  of every rule  $a : C \rightarrow L'$  before the translation (a change that does not affect the semantics), and while the precondition  $L$  is replaced by  $KL$  in the execution model  $X(P)$ , it is replaced by the weaker condition  $ML$  in the heuristic model  $H(P)$ .

The introduction of the literals  $ML$  ensures that the ‘wishful thinking’ done over the *action preconditions* does not translate into ‘wishful thinking’ about their *effects*. A different situation would arise if the non-deterministic effects  $Kx | K\neg x$  would be relaxed into the deterministic effects  $Kx \wedge K\neg x$ , instead of the weaker  $Mx \wedge M\neg x$ . In the first, a plan for observing  $x$  will be a plan for making  $x$  true (or false), something that does not result from the latter encoding as the  $M$ -literals are used only in action preconditions but not in conditions or goals.

Two reasons explain why the resulting heuristic model  $H(P)$ , which is a classical planning problem, provides a useful heuristic criterion for selecting actions in the contingent planning problem  $P$ . If action preconditions in  $P$  are ignored (after copying them as conditions), the resulting delete-relaxation is a conformant problem [6] whose classical translation is the precondition and delete-free version of  $H(P)$ . The problem with this choice is that sensing actions are ignored. The model  $H(P)$ , on the other hand, does not ignore the action preconditions in  $P$  but relaxes them in terms of the  $M$ -literals and uses the sensing actions along with the rules that propagate the  $M$ -literals for achieving them.

## 6 Action Selection and the CLG Planner

The action selection cycle in the Closed-Loop Greedy planner is based on the execution model  $X(P)$  and the heuristic model  $H(P)$ , relies on the classical FF planner [13], and proceeds as follows:

1. given the current state  $s_x$  in  $X(P)$  (initially  $I'$ ),  $X(P)$  deductively closes it by applying all its deductive rules, passing the resulting state  $s'_x$  to  $H(P)$ ,
2. a modified version of the classical FF planner is called upon  $H(P)$  with  $s'_x$  as the starting state, returning an *improving action sequence*  $\pi$ ,
3. the actions in  $\pi$  are then applied *in the execution model*  $X(P)$ , starting in the state  $s'_x$  and *finishing right after the first non-deterministic action* in a state  $s_y$  with a true condition applied, letting the environment, a simulator, or a 'coin' choose the effect. If a *full contingent plan* is desired, all possibilities must be tried, recording the action sequences leading to the goal along each possible observation sequence,
4. if the resulting state  $s_y$  is a goal state in  $X(P)$ , then the execution (along this branch in the full contingent plan setting) is successfully terminated, else the cycle repeats at 1 with  $s_x := s_y$ .

The 'improving action sequence' in Step 3 refers to the action sequence found by FF after performing a *single enforced hill climbing step*, which –if successful– maps the current state  $s$  into another state  $s'$  that improves the value of the FF heuristic in  $H(P)$ . If this enforced hill climbing fails, the execution (along this branch) is terminated with failure.

It is possible to prove that if FF returns an action sequence that is a classical plan for  $H(P)$  with no actions corresponding to sensing actions, such a plan is a *conformant* plan that solves  $X(P)$  and hence  $P$ . This is due to the soundness of the conformant translation and to the equivalence of the executions of the models  $X(P)$  and  $H(P)$  when no sensing actions are applied, which implies the invariant  $ML = KL$ .

## 7 Preliminary Experimental Results

We tested the Closed-Loop Greedy Planner (CLG) over two sets of problems: a set of existing benchmarks, and a new set of problems of our own. We compare CLG with Contingent-FF, run both with and without the helpful actions pruning mechanism [6]. The experiments are obtained on a Linux machine running at 2.33 Ghz with 8Gb of RAM, with a cutoff of 30mn of time or 1.8Gb of memory. For the implementation, we modified the FF planner [13] so that it accepts one PDDL file where the two models  $X(P)$  and  $H(P)$  are combined, using flags for fixing the right set of actions and fluents, for doing the progression and calculating the heuristic respectively. The actual numbers reported are preliminary as there are a number of aspects in the current implementation that need to be improved. See the discussion below.

problem	Contingent FF		CLG			
	time (s)	nacts	t0 time (s)	pddl size (Mb)	time (s)	nacts
ebtcs-30	0,95	59	0,56	3,19	3,26	89
ebtcs-50	11,9	99	2,04	11,27	22,83	149
ebtcs-70	68,01	139	5,17	26,94	91,06	209
elog-5	0,04	156	0,05	0,29	0,26	130
elog-7	0,07	223	0,05	0,32	0,36	193
elog-huge	> 1.8Gb		0,95	2,39	523,1	43835
medpks-30	11,72	60	1,06	5,35	10,09	61
medpks-50	164,14	100	3,94	19,17	79,17	101
medpks-70	1114,21	140	20,92	109,31	> 1.8Gb	
unix-3	4,02	111	2,41	26,00	52,59	111
unix-4	221,23	238	24,08	226,59	> 1.8Gb	

**Table 1.** Solution times for Contingent-FF and CLG over the first set of domains. 'nacts' stands for the total number of actions on the solution, 't0 time' is translation time to get  $X(P)$  and  $H(P)$  from the original problem, 'pddl size' is their size, and 'time' is total time minus translation time.

Table 1 shows data concerning the first set of problems: *ebtcs-x* stands for *enforced Bomb-in-the-toilet* with  $x$  bombs and a single toilet, *elog-x* for *enforced Logistics*, *medpks-x* is a diagnose-and-treat domain, and *unix-x* is the problem of moving one file to the root node of tree directory with the *ls* action showing the contents of a directory. All these examples are taken from the Contingent-FF distribution.

Table 2 shows the solution times for some new problems. *colorballs-n-x* is the problem of collecting  $x$  colored balls from an  $n \times n$  grid whose location and color are not known but can be observed when agent and ball are in the same cell. *doors-n* is the problem of traversing a square room  $n \times n$ , with walls covering every odd column of the square, except for an open door at an unknown position in every column. The open door can be detected by a sensing action from an adjacent cell.

On the first set of problems, Contingent-FF and CLG are comparable in terms of coverage with the former taking less time. The 'helpful actions' option was not used in order to solve medpks. The number of actions in the table do not measure actually the quality of the contingent plans but the total number of actions along all the branches. For CLG, the size of the domain-pddl file produced by the translation constitutes the bottleneck for solving the instances medpks-70 and unix-4.

On the second set of problems, Contingent-FF solves only the smallest *colorballs* instances, and it fails in the *doors* instances due to a bug in Contingent-FF, confirmed by the authors. In these domains, CLG exhibits a more robust behavior.

In all the cases above, CLG is used for and successfully generates full contingent plans by considering all possible 'contingencies'. An inefficiency in our current implementation for this task consists in that contingent plans are represented as trees rather than graphs, meaning that (belief) states that are reached

problem	Contingent FF		CLG			
	time (s)	nacts	t0 time (s)	pddl size (Mb)	time (s)	nacts
colorballs-4-1	0,27	277	0,14	0,70	0,58	281
colorballs-4-2	36,33	18739	0,27	1,35	39,72	18232
colorballs-4-3	> 30mn		0,41	2,0	> 30mn	
colorballs-5-1	1,83	611	0,44	1,98	2,43	584
colorballs-5-2	867,28	71157	0,82	3,89	307,4	67945
colorballs-5-3	> 30mn		1,28	5,79	> 30mn	
colorballs-6-1	7,43	1091	1,17	5,01	9,48	1021
colorballs-6-2	> 30mn		2,19	9,91	> 30mn	
colorballs-7-1	42,03	1826	2,83	11,38	30,88	1614
colorballs-7-2	> 30mn		5,21	22,60	> 30mn	
colorballs-8-1	> 30mn		6,02	23,62	95,73	2397
colorballs-9-1	> 30mn		12,78	45,53	256,59	3384
colorballs-9-2	> 30mn		23,58	90,79	> 1.8Gb	
doors-7	fail		1,53	4,58	61,89	2357
doors-9	fail		7,22	15,00	> 30mn	

**Table 2.** Solution times for Contingent-FF and CLG over second set of problems. 'nacts' stands for the total number of actions in solution. 't0 time' is translation time to get  $X(P)$  and  $H(P)$  from the original problem, 'pddl size' is their size, and 'time' is total time minus the translation time. 'fail' means that Contingent-FF (incorrectly) reported a problem as unsolvable.

through different execution paths are explored multiple times. This should be easy to fix and should lead to faster run times and more compact plans (with an smaller total number of actions).

A main motivation for this work has been to have a fast but informed Closed-Loop planner that can scale up to problems in which the contingent solutions have exponential size and thus cannot be constructed. For testing this, we generated 25 random executions in instances of *colorballs* and *doors*, finding all executions leading to the goal, even in cases like *colorballs-9-2* and *7-4*, and *doors-9* for which no full contingent plans could be computed due to time or memory limitations.

## 8 Discussion

We have developed a domain-independent action selection mechanism for planning with sensing that can be used as a greedy but informed closed-loop planner or as a contingent planner able to generate full plans. The approach builds on two recent ideas that explain also why the approach works: the first by Hoffmann and Brafman, that states that the delete-relaxation of a precondition-free contingent problem is a conformant problem; the second by Palacios and Geffner, that shows how conformant problems can be translated into classical problem at the 'knowledge level'. Rather than applying the two transformations in a row resulting in a formulation that ignores sensing actions, we have shown however

how preconditions and sensing actions can be brought in the formulation by introducing new literals for modeling ‘contingent knowledge’. We have also tested the action selection mechanism empirically over a number of problems, showing that it compares well with state-of-the-art planners for computing full contingent plans, while being able to scale up better when used in closed-loop fashion.

As future work, we plan to improve the implementation, clean up the formulation by incorporating axioms or ramifications in the target language of the translation, and redefine the ‘enforced hill climbing’ (EHC) step that selects the action sequence to apply next so that the deterministic heuristic model  $H(P)$  is used for computing the heuristic only, while the non-deterministic execution model  $X(P)$  is used in the progression within the EHC. This is needed for ruling out the possibility of loops during the execution.

## Acknowledgments

We thank the anonymous reviewers for useful comments and J. Hoffmann for help with Contingent-FF. H. Geffner is partially supported by Grant TIN2006-15387-C03-03, and H. Palacios by an FPI fellowship, both from MEC/Spain.

## References

1. Peot, M., Smith, D.E.: Conditional nonlinear planning. In Hendler, J., ed.: Proc. 1st Int. Conf. on AI Planning Systems. (1992) 189–197
2. Pryor, L., Collins, G.: Planning for contingencies: A decision-based approach. *Journal of AI Research* **4** (1996) 287–339
3. Haslum, P., Jonsson, P.: Some results on the complexity of planning with incomplete information. In: Proc. ECP-99, Lect. Notes in AI Vol 1809, Springer (1999)
4. Rintanen, J.: Complexity of planning with partial observability. In: Proc. ICAPS-2004. (2004) 345–354
5. Bonet, B., Geffner, H.: Planning with incomplete information as heuristic search in belief space. In: Proc. of AIPS-2000, AAAI Press (2000) 52–61
6. Hoffmann, J., Brafman, R.: Contingent planning via heuristic forward search with implicit belief states. In: Proc. ICAPS 2005. (2005)
7. Bertoli, P., Cimatti, A., Roveri, M., Traverso, P.: Strong planning under partial observability. *Artif. Intell.* **170**(4-5) (2006) 337–384
8. Bryce, D., Kambhampati, S., Smith, D.E.: Planning graph heuristics for belief space search. *Journal of AI Research* **26** (2006) 35–99
9. Petrick, R., Bacchus, F.: A knowledge-based approach to planning with incomplete information and sensing. In: Proc. AIPS’02. (2002) 212–221
10. Palacios, H., Geffner, H.: Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In: Proc. AAAI-06. (2006)
11. Bonet, B., Givan, B.: Results of the conformant track of the 5th int. planning competition. At <http://www ldc.usb.ve/ bonet/ipc5/docs/results-conformant.pdf> (2006)
12. Palacios, H., Geffner, H.: From conformant into classical planning: Efficient translations that may be complete too. In: Proc. ICAPS-07. (2007)
13. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* **14** (2001) 253–302