Contingent Planning via Heuristic Forward Search with Implicit Belief States

Jörg Hoffmann
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany

Ronen I. Brafman
NASA Ames Research Center *and*
Computer Science Department
Stanford University

Abstract

Contingent planning is the task of generating a conditional plan given uncertainty about the initial state and action effects, but with the ability to observe some aspects of the current world state. Contingent planning can be transformed into an And-Or search problem in belief space, the space whose elements are sets of possible worlds. In (Brafman & Hoffmann 2004), we introduced a method for implicitly representing a belief state using a propositional formula that describes the sequence of actions leading to that state. This representation trades off space for time and was shown to be quite effective for conformant planning within a heuristic forward-search planner based on the FF system. In this paper we apply the same architecture to contingent planning. The changes required to adapt the search space representation are small. More effort is required to adapt the relaxed planning problems whose solution informs the forward search algorithm. We propose the targeted use of an additional relaxation, mapping the relaxed *contingent* problem into a relaxed *conformant* problem. Experimental results show that the resulting planning system, Contingent-FF, is highly competitive with the state-of-the-art contingent planners POND and MBP.

Introduction

Contingent planning is the task of generating conditional plans given uncertainty about initial state and action effects. One has the ability to sense the value of certain world aspects during plan execution, and branch depending on the observed value. The plan should be successful (achieve all goal propositions) regardless of which particular initial world we start from and which action effects occur.

Contingent planning can be transformed into an And-Or search problem in the space of belief states, i.e., the space whose elements are sets of possible world states. (We stick to this distinction between *world* states and *belief* states.) Bonet and Geffner (2000) introduced, in their GPT system, the idea of planning in belief space using heuristic forward search. But the number of possible worlds in a belief state can be large, and so GPT, which explicitly represents these states, usually fails to scale up. In MBP, Bertoli and Cimatti (2001; 2002) tackle this problem by using BDDs to represent the belief states. This often works better, but the size of the constructed BDDs is still often prohibitive for scaling.

In this paper we utilize a third, lazy, approach to represent belief states, introduced for conformant planning in (Brafman & Hoffmann 2004).[1] Belief states are represented implicitly through the action sequences leading to them (plus the initial state description). During search, for every belief state s encountered by action sequence $\bar{a}$, (only) a partial knowledge about s is computed, namely the propositions p that are *known* in s. p is known in s if it is true in the *intersection* of the worlds in s. Obviously, $\bar{a}$ is a conformant plan iff all goal propositions are known in s, and so the knowledge about the known propositions suffices to perform conformant planning. Testing if a proposition is known, given an action sequence $\bar{a}$, is co-NP complete, and is done by reasoning about a CNF formula $\phi(\bar{a})$ that captures the semantics of $\bar{a}$.

The same idea can be utilized for contingent planning. The main difference is that we now obtain an And-Or search process, rather than the previous Or search process. The And aspect comes from the multiple possible values of observations for sensing actions. The plan, which is now a *tree* rather than a sequence of actions, has to treat every possible outcome. That is, the search space is an And-Or tree, and the plan is a sub-tree where all leaves are goal (belief) states. Each node in the And-Or search space can be represented using a straightforward extension of the techniques described above. Belief states s are now represented implicitly through action-observation sequences $\overline{ao}$. Testing for known propositions is still co-NP complete, and is still done by CNF reasoning. The construction of the formulas $\phi(\overline{ao})$ needs to be changed only slightly.

As before, in comparison to the belief state representations used in GPT and MBP, our approach trades space for time. Given a state s and an applicable action a, in order to compute the known propositions in the result state s', we need to reason about the entire action-observation sequence leading to s', *including those actions/observations that lead to* s. With complete knowledge of s in memory, we would not need to do the latter. Our intuition, and empirical experience, is that even a naive SAT solver has little trouble reasoning about the effects of an action sequence if the possible interactions are not overly complex.

A blind traversal of the search space is infeasible due to the combinatorial explosion in possible action choices. In

[1] Conformant planning is the special case of contingent planning where no observations are possible.

classical and in conformant planning, this problem has been tackled most successfully by using heuristic functions based on a relaxation of the planning task. In Conformant-FF this relaxation is obtained by assuming empty delete lists and by performing "relaxed" reasoning about the known facts. In a nutshell, the latter is done by transforming the clauses of the original CNF into binary clauses, and embedding the reasoning about this 2-CNF projection naturally in the planning-graph structure used to perform relaxed planning. The length of the generated relaxed plan is used to guide (Hill-Climbing or A*) forward search.

To adapt Conformant-FF's relaxed planning process to contingent planning, we propose to apply another relaxation. We observe that, to any contingent task without delete lists, there is a plan without observations – a conformant plan – if one assumes what we call the *generous* action execution semantics: assume that, during plan execution, actions with unsatisfied preconditions are ignored instead of making the plan fail. A task with generous execution semantics can be obtained by moving, for each action, the preconditions into the conditions of the action's effects. Applying this additional relaxation *as a pre-process* of Conformant-FF's relaxed planning process, that process can in principle be reused for contingent planning with no modifications whatsoever. However, the resulting heuristic function is often very uninformative because moving all action preconditions into effect conditions renders Conformant-FF's 2-CNF projection a much cruder approximation. We therefore develop technology that applies the generous execution semantics in a targeted way *inside* the relaxed planning process, making use of that semantics only where possible and necessary.

Beside not treating partial observability, in Conformant-FF we did not treat non-deterministic action effects. In our new system, we do handle a simple form of such effects. We describe the respective adaptations together with the other algorithmic techniques. We discuss extensions to richer forms of non-deterministic effects.

Our techniques are implemented in a system we call Contingent-FF. The system is based on the Conformant-FF code, and does a standard weighted AO* search. We empirically compare Contingent-FF with the state-of-the-art contingent planners MBP (Bertoli *et al.* 2001) and POND (Bryce, Kambhampati, & Smith 2004). Our experiments cover a number of traditional contingent planning benchmarks as well as more recent contingent versions of classical planning domains. Our planner is generally competitive with MBP and POND, sometimes outperforming both of them. The generated plans are, in most cases, similar in terms of quality to those generated by POND, and better than those generated by MBP.

The paper is organized as follows. Section 2 briefly describes the planning framework we consider. Section 3 describes the search space representation and the computation of known propositions. Section 4 explains our heuristic function. Section 5 outlines the limited form of repeated states checking we currently use. Section 6 gives some implementation details and our empirical results, Section 7 concludes the paper with a brief discussion of related and future work.

Planning Framework

The contingent planning framework we consider adds observations and uncertainty to (sequential) STRIPS with conditional effects. In the latter formalism, planning tasks are triples (A, I, G) of *action set*, *initial world state*, and *goal world state*. World states w are sets of propositions (those satisfied in them). Actions a are pairs $(pre(a), E(a))$ of the *precondition* – a set of propositions – and the *effects* – a set of conditional effects. A conditional effect e is a triple $(con(e), add(e), del(e))$ of proposition sets, corresponding to the effect's *condition*, *add*, and *delete* lists respectively. An action a is *applicable* in a world state w if $w \supseteq pre(a)$. The result of executing a in w is denoted as $a(w)$. If a is not applicable in w, then $a(w)$ is undefined. If a is applicable in w, then all conditional effects $e \in E(a)$ whose condition is satisfied in w, i.e., $w \supseteq con(e)$, are executed; these effects are said to *occur* (unconditional effects have $con(e) = \emptyset$). Executing a conditional effect e in w results in the world state $w - del(e) + add(e)$. An action sequence $\langle a_1, \dots, a_n \rangle$ is a *plan* if $a_n(\dots a_1(I)\dots) \supseteq G$.

We extend the above with uncertainty about the initial state, non-deterministic action effects, and observations. Belief states s are sets of world states. An action a is applicable in s if $\forall w \in s : pre(a) \subseteq w$. In that case, $a(s)$ is defined as $\{a(w) \mid w \in s\}$. The initial state is a belief state represented by a propositional CNF formula $\mathcal{I}$. The possible initial world states are those that satisfy that formula. We denote that set of world states with $2^{\mathcal{I}}$.

Observations are encoded as follows. Beside the *normal actions* as above, there are now special *observation actions*. These are pairs $(pre(a), o(a))$ where $pre(a)$ is a set of propositions as before, and $o(a)$ is a proposition. Such actions observe the value of $o(a)$, thereby splitting the belief state at hand and introducing two branches into the plan, one marked with $o(a)$ and one marked with $\neg o(a)$. More formally, given a belief state s, an observation action a, and a literal $l(a) \in \{o(a), \neg o(a)\}$, we define $a(s, l(a))$ as $\{w \mid w \in s, w \models l(a)\}$ if a is applicable in s; otherwise, $a(s, l(a))$ is undefined. Plans are now trees of actions in the obvious way. The semantics are defined as follows. An action tree T *solves* a belief state s if (1) T is empty and $\forall w \in s : w \models G$; or (2) the root of T, a, is applicable in s and either (3a) a is a normal action, and removing a from T yields a tree that solves $a(w)$; or (3b) a is an observation action and, for every $l(a) \in \{o(a), \neg o(a)\}$, the tree rooted at a's child marked with $l(a)$ solves $a(s, l(a))$. A plan for a task $(A, \mathcal{I}, G)$ is a tree of actions in A that solves $2^{\mathcal{I}}$.

Normal actions a can have *non-deterministic* effects. These are effects $e \in E(a)$ that may or may not occur. When applying a to a world state w, and $con(e) \subseteq w$, then $a(w)$ is a set of two states: one where e occurred, and one where e did not occur. Extension to belief states in the obvious way, the definition of the semantics remains unchanged. Our current implementation is restricted to a single *un*conditional non-deterministic effect per action. Handling more, or more general, non-deterministic effects is *conceptually* not different; the algorithms as written up here treat multiple conditional non-deterministic effects. Implementing the more general techniques, and determining how well they behave in practice, is an important open topic. By $detE(a)$ we denote the deterministic effects of an action.

In our input format, we also allow *negations* in front of propositions in preconditions, effect conditions, and the goal. Such negations are compiled away in a simple preprocess. For every proposition p we introduce a new inverse proposition called $not\text{-}p$. This is introduced into the action effects in an inverse manner (adds and deletes are swapped), and (the obvious two) clauses encoding $p \leftrightarrow not\text{-}p$ are inserted into the initial state formula. This way $not\text{-}p$ will always be true iff p is false, and $not\text{-}p$ can be used in the conditions instead of $\neg p$. In difference to the STRIPS context, a slight complication arises from the observation actions: such actions a now observe the value of *two* propositions, $o(a)$ and $not\text{-}o(a)$. At some points, this requires an extra treatment, which we will explain. For convenience, by $O(a)$ we denote the set $\{o(a), not\text{-}o(a)\}$. Also, we use $E(a)$ for observation actions, and $O(a)$ for normal actions, to avoid clumsy case distinctions; both sets are empty. If we write $a(s, l(a))$ for a normal action then $l(a)$ is assumed as the (dummy) literal $TRUE$, and $a(s, l(a)) = a(s)$.

Search Space

We perform a standard AO* forward search in belief space. The search space is an And-Or tree whose nodes are, alternatingly, belief states (Or nodes) and actions (And nodes). The Or children of a belief state correspond to the different applicable actions. The And children of an observation action correspond to its two different outcomes. For normal actions (even if they have a non-deterministic effect), there is just one child so the And node trivializes.

Each belief state s is represented only by the initial state formula $\mathcal{I}$, and the action-observation sequence (*ao-sequence* for short) $\overline{ao} = \langle a_1, \dots, a_n \rangle$ leading to s. $\overline{ao}$ corresponds to a path in the And-Or search space. Each a_i may be either a normal action, or an observation. In the latter case, a_i is marked with *the observed literal* $l(a_i)$ (the branch the path takes). The belief state represented by $\overline{ao}$ is $s = a_n(\dots a_1(2^{\mathcal{I}}, l(a_1)), \dots, l(a_n))$.

During search, for each belief state encountered, we compute the sets of *known* and *negatively known* propositions. A proposition p is known in belief state s if p holds in every $w \in s$. We say that p is negatively known in s if p does *not* hold in any $w \in s$. A proposition that is neither known nor negatively known is *unknown*. Deciding about whether a proposition is known or not is co-NP complete (hardness by the same result for the conformant case (Brafman & Hoffmann 2004), membership by the reduction to the complement of SAT outlined below).

In our implementation, we compute the sets of known and negatively known propositions in a belief state by using a CNF corresponding to the semantics of the respective ao-sequence as follows. We use a time index to differentiate between values of propositions at different points along the execution of the ao-sequence. Say the ao-sequence is $\overline{ao} = \langle a_1, \dots, a_n \rangle$. We obtain our CNF $\phi(\overline{ao})$ as follows. We initialize $\phi(\overline{ao})$ as $\mathcal{I}$ indexed with time 0 (i.e., for each clause $l_1 \vee \dots \vee l_k \in \mathcal{I}$ we add $l_1(0) \vee \dots \vee l_k(0)$ into $\phi(\overline{ao})$). We then use a_1 to extend $\phi(\overline{ao})$. If a_1 is a normal action:

- *Effect Axioms*: for every deterministic effect e of a_1, $con(e) = \{p_1, \dots, p_k\}$, and every proposition $p \in add(e)$, we insert the clause $\neg p_1(0) \vee \dots \vee \neg p_k(0) \vee p(1)$; for every proposition $p \in del(e)$, we insert the clause $\neg p_1(0) \vee \dots \vee \neg p_k(0) \vee \neg p(1)$.
- *Frame Axioms*: for every proposition p, let $e_1, \dots, e_n$ be the effects of a_1, including non-deterministic effects, such that $p \in del(e_i)$; for every tuple $p_1, \dots, p_n$ such that $p_i \in con(e_i)$ we insert the clause $\neg p(0) \vee p_1(0) \vee \dots \vee p_n(0) \vee p(1)$ (read this clause as an implication: if p was true before and has not been deleted by either of e_i, it is still true after a_1). Symmetrically, when $e_1, \dots, e_n$ are the effects of a_1 such that $p \in add(e_i)$, we insert for every tuple $p_1, \dots, p_n$ with $p_i \in con(e_i)$ the clause $p(0) \vee p_1(0) \vee \dots \vee p_n(0) \vee \neg p(1)$ (if p was false before and has not been added, it is still false after a_1).[2]

If a_1 is an observation action, we insert only the frame axioms. (Since the action has no effects, these clauses simplify to $\neg p(0) \vee p(1)$ and $p(0) \vee \neg p(1)$ for each p.) We also insert the *Observation Axiom*, i.e. the unit clause $l(a_1)(1)$, thereby imposing the constraint given by the observation. For example, if a_1 observed p to be false, we add the clause $\neg p(1)$.

In the same fashion, we use a_2 to further extend the formula and so on until the axioms for a_n have been inserted. For the resulting CNF, the following holds.

Proposition 1 *Given a contingent planning task* $(A, \mathcal{I}, G)$, *a belief state* s *given by an* n*-step ao-sequence* $\overline{ao} \in A^*$, *and a proposition* p. *Then* p *is known in* s *iff* $\phi(\overline{ao})$ *implies* $p(n)$.

Proof: For every initial world state $I \in 2^{\mathcal{I}}$, the following holds. Denote by $\phi(\overline{ao})_I$ the formula that results from inserting the values given by I into $\phi(\overline{ao})$ at time 0. Then the satisfying variable assignments σ to $\phi(\overline{ao})_I$ – the values of propositions/variables at time steps in $\overline{ao}/\phi(\overline{ao})$ – correspond exactly to all executions of $\overline{ao}$ in I, with different outcomes of non-deterministic effects, that end up in s. (If there is no such execution of $\overline{ao}$ in I, then $\phi(\overline{ao})_I$ is unsatisfiable.) Thus $\phi(\overline{ao}) \wedge \neg p(n)$ is unsatisfiable iff there is no initial state $I \in 2^{\mathcal{I}}$, and no non-deterministic effect outcome, such that p does not hold upon executing $\overline{ao}$ in I. ■

We use Proposition 1 to compute the set of known propositions as follows. Start with the empty set. Then, for each proposition p, hand $\phi(\overline{ao}) \wedge \neg p(n)$ over to the underlying SAT solver. If the result is "unsat" then add p to the known propositions. If the result is "sat", do nothing. Symmetrically, we compute the set of negatively known propositions by handing the formulas $\phi(\overline{ao}) \wedge p(n)$ to the SAT solver.

By Proposition 1, instead of enumerating all initial world states and non-deterministic effect outcomes for computing

[2]Note that, if there is an unconditional delete (add), no positive (negative) frame axiom is generated. Note also that the number of frame axioms is exponential in the number of distinct conditional effects of a single action that can add/delete the same proposition. One could avoid this by introducing additional variables into the CNF. We have not done so because in practice actions seldom affect the same proposition with different effects.

whether a proposition is known or not, we can reason about the formula $\phi(\overline{ao})$. As indicated before, this reasoning appears to be cheap in practice, when interactions between action effects are not overly complex. Note that one can apply several significant reductions to the number of SAT calls made, and the size of the CNF formulas looked at. In our current implementation, these are:

- Simplify $\phi(\overline{ao})$ by inserting the values of propositions at times $i < n$ which are known to be true or false.
- Make SAT calls only on propositions p such that p is affected by a conditional effect that possibly occurs (all conditions are either known or unknown, and at least one of them is unknown).

Once the known propositions in s are computed, the actions applicable to s are those whose preconditions are all known in s. A belief state satisfies the goal iff all goal propositions are known.

Heuristic Function

In classical planning, a successful idea has been to guide (forward, e.g.) search by heuristic functions based on a relaxation of the planning task, where the relaxation is to assume that all delete lists are empty. We adapted this idea to the conformant setting in the Conformant-FF planner, using ideas introduced in FF (Hoffmann & Nebel 2001). We now explain how this idea can be farther adapted to suit the contingent planning setting. We start by explaining the essential ideas behind FF's relaxed planning approach.

To each world state during a forward search, FF computes a relaxed plan – a plan that achieves the goals when all delete lists are assumed empty – and takes the length of the relaxed plan as the state's heuristic value. Relaxed plans are computed as follows. Starting from a world state w, build a *relaxed planning graph* as a sequence of alternating proposition layers $P(t)$ and action layers $A(t)$, where $P(0)$ is the same as w, $A(t)$ is the set of all actions whose preconditions are contained in $P(t)$, and $P(t+1)$ is $P(t)$ plus the add effects (with fulfilled conditions) of the actions in $A(t)$. That is, $P(t)$ always contains those facts that would be true if one executed (the relaxed versions of) all actions at the earlier layers up to $A(t-1)$. From a proposition layer $P(m)$ in which the goals are contained one can easily extract a *relaxed plan*, i.e a subset of the earlier actions that suffices to achieve the goals when ignoring the delete lists. One can use the following simple backchaining loop: select achieving actions at layers $t < m$ for all goals in $P(m)$, insert those actions' preconditions and the respective effect conditions as new subgoals (which by construction are at earlier layers the respective actions), then step backwards and select achievers for the subgoals. The heuristic value $h(w)$ for w then is the number of actions selected in backchaining – the length of the relaxed plan. If (and only if) there is no relaxed plan then the planning graph will reach a fixpoint $P(t) = P(t+1)$ without reaching the goals. $h(w)$ is then set to ∞, excluding the state from the search space – if there is no relaxed plan from w then there is no real plan either.

In the conformant setting, we extended the above with additional fact layers $uP(t)$ containing the facts *unknown* at time i. We introduced reasoning about the unknown facts, i.e., reasoning about when such facts become known in the relaxed planning graph, when assuming that (the relaxed versions of) all actions in the earlier layers are executed. In order to make the reasoning efficient, we made another relaxation to the planning task, on top of ignoring the delete lists. We ignored all but one of the unknown conditions of those effects that were unknown to occur. That is, if an action a appeared in layer $A(t)$, and for effect e of a we had $con(e) \subseteq P(t) \cup uP(t), con(e) \cap uP(t) \neq \emptyset$ – e may occur depending on the initial world state – then we assumed that $|con(e) \cap uP(t)| = 1$. We arbitrarily selected one $c \in con(e) \cap uP(t)$, and reasoned as if the only elements in $con(e)$ were $(con(e) \cap P(t)) \cup \{c\}$. With this simplifying assumption, the implications that the (unknown) action effects induce between unknown propositions come down to binary implications of the form $c(t) \to p(t+1)$. The set of all these implications forms a tree. The reasoning needed to find out if a proposition p becomes known at i can be done by a simple backward chaining over the tree edges that end in $p(t)$, followed by a SAT check to see if the initial state formula implies the disjunction of the reachable tree leafs.

The above processes will be explained in detail below, with pseudo-code, together with their extension to the contingent setting. Note that the processes constitute a stronger (complete but not sound) form of the real reasoning, i.e. we over-approximate the sets of propositions that become known at any point in time. (It becomes easier to achieve a fact, in line with the "relaxed" planning approach.) Note also that the implications $c(t) \to p(t+1)$ are a 2-projection of the real implications $\bigwedge_{c \in con(e) \cap uP(t)} c(t) \to p(t+1)$. Note finally that, in principle, it would be possible to do the full SAT checks, without any 2-projection, to see if a proposition becomes known in a layer. However, doing such a full check for every unknown proposition at every level of the relaxed planning graph for every search state would very likely be too expensive, computationally.

To handle contingent relaxed planning, we need to take into account the And-branches potentially introduced by observation actions. Given the exponential number of possible branches, it seems likely that inside a heuristic computation – that must be repeated in every search state – one has to sacrifice precision for computational efficiency. Our core observation is that, given we ignore the delete lists anyway, we can map the relaxed *contingent* planning problem into a relaxed *conformant* planning problem without losing completeness, i.e. preserving relaxed plan existence (details below). The mapping is to include, for each action a and every $e \in E(a)$, $pre(a)$ into $con(e)$, and then assume $pre(a) = \emptyset$. We call this the *generous execution semantics*. It can be viewed as, during plan execution, simply ignoring actions whose preconditions are not satisfied. When we combine this with ignoring delete lists, we get the following.

Proposition 2 *Given a contingent planning task* $(A, \mathcal{I}, G)$. *If there is a plan – an action-observation tree –* T *for the task, then there is a sequence* $\bar{a}$ *of normal actions such that* $\bar{a}$ *is a plan when assuming the generous execution semantics and empty delete lists.*

Proof: Let $\bar{a}$ be the normal actions in T, in some order respecting the edges in T (e.g., breadth-first). Then under the simplifying assumptions $\bar{a}$ is a plan. Every ao-path $\overline{ao}$ in T (every potential plan execution) is a subsequence of $\bar{a}$. Since we ignore the delete lists, every fact that is achieved by $\overline{ao}$ is also achieved by $\bar{a}$. Since we assume the generous execution semantics, every action in $\bar{a}$ is applicable in the state of its execution. For the actions in the $\overline{ao}$-subsequence of $\bar{a}$, the effects occur (by a simple inductive argument). ■

Without assuming the generous execution semantics, the proof argument does not work because actions belonging to different T paths than $\overline{ao}$ may have unsatisfied preconditions, and therefore result in undefined states.

Appreciate that it is important that the problem we solve inside the heuristic function is complete relative to the real problem, i.e. existence of a real solution implies existence of a relaxed solution. Only then can we safely skip states for which there is no relaxed solution. Proposition 2 tells us that, by ignoring delete lists and assuming the generous execution semantics, we obtain such a complete relaxation even if we totally ignore the observation actions in a contingent task. A straightforward idea is therefore to simply re-use our conformant relaxed planning machinery, with the generous execution semantics.

We implemented the above approach, and found it to work well in some cases, but to behave very badly in several purely conformant tasks that could be solved efficiently beforehand. The reason for the new inefficiency lies in that our conformant relaxed planning machinery ignores all but one of the unknown conditions of any effect. Recall that, to obtain the generous execution semantics, we have to move all preconditions into the effect conditions. With more effect conditions, our machinery is likely to result in a cruder approximation, yielding worse heuristic values and bad overall performance. The approach we have taken to ameliorate this difficulty is, in a nutshell, to apply the generous execution semantics, within the relaxed problem solving mechanism, only where *possible* and *needed*. Moving a precondition $p \in pre(a)$ into the effect conditions of the effects $e \in E(a)$ is considered possible if p is observed by an earlier action in the relaxed planning graph. The move is needed if one of the effects contributes to the shortest way (in the relaxation) of making some unknown proposition become known – otherwise, the move does not affect the relaxed planning process at all. Note that without observation actions the relaxation is exactly the same as before.

We now describe our implemented algorithms in detail. A lot is inherited from Conformant-FF. The new parts are those concerned with non-deterministic effects and observation actions. These new parts are not understandable without presenting the algorithms in total. Figure 1 shows the overall procedure that builds a contingent relaxed planning graph.

In Figure 1, layers $-n$ to -1 of the relaxed planning graph correspond to the ao-sequence $\overline{ao}$ leading to the considered belief state s. The need for this is a consequence of our decision to compute and store only partial knowledge about the belief states. To make our relaxed planning algorithm complete relative to the relaxation, we have to let it reason about

procedure build-CRPG($\langle a_{-n}, \dots, a_{-1} \rangle, A, \mathcal{I}, G$)
1 *Imp* $:= \emptyset$, $P(-n) := \{p \mid p$ is known in $\mathcal{I}\}$
2 $uP(-n) := \{p \mid p$ is unknown in $\mathcal{I}\}$
3 **for** $t := -n \dots -1$ **do**
4 $P(t+1) := \{p \mid p$ is known after $a_t\}$
5 $uP(t+1) := \{p \mid p$ is unknown after $a_t\}$
6 *Imp* $\cup = \{(p(t), p(t+1)) \mid p \in uP(t) \cap uP(t+1)\}$
7 **for** all $e \in detE(a_t)$,
$con(e) \subseteq P(t) \cup uP(t), con(e) \cap uP(t) \neq \emptyset$ **do**
8 select $c \in con(e) \cap uP(t)$
9 *Imp* $\cup = \{(c(t), p(t+1)) \mid p \in add(e) \cap uP(t+1)\}$
10 **endfor**
11 **endfor**
12 propagate-observations-to-$\mathcal{I}$($\overline{ao}, P, uP$)
13 $t := 0$, $oP(0) := \emptyset$
14 **while** $G \not\subseteq P(t)$ **do**
15 $A(t) := \{a \mid a \in A, pre(a) \subseteq P(t) \cup oP(t)\}$
16 build-timestep($t, A(t)$)
17 **if** $P(t+1) = P(t)$ and
18 $uP(t+1) = uP(t)$ and
19 $oP(t+1) = oP(t)$ and
20 $\forall p \in uP(t+1)$: *Impleafs*$(p(t+1))$ = *Impleafs*$(p(t))$ **then**
21 **return** FALSE
22 **endif**
23 $t := t+1$
24 **endwhile**
25 **return** TRUE

Figure 1: Building a contingent relaxed planning graph (CRPG).

what conditional effects may occur along $\overline{ao}$, depending on the initial world state. The negative indices of the actions in $\overline{ao}$ are chosen to simplify the presentation.

As said, the sets $P(t)$, $uP(t)$, and $A(t)$ are taken to contain the propositions that are known to hold at t, the propositions that are unknown at t, and the actions that are known to be applicable at t, respectively. The binary implications $c(t) \rightarrow p(t+1)$, induced by action effects with unknown conditions or between propositions that are unknown at both t and $t+1$, are stored in an implication set *Imp*. Non-deterministic effects need not be considered as these can not be used to establish a proposition (recall that $detE(a)$ denotes the deterministic effects of an action). Let us read Figure 1 from top to bottom. Lines 1 and 2 do obvious initialization steps. Lines 3 to 11 create the sets $P(-n+1), uP(-n+1), \dots, P(0), uP(0)$, and insert the (approximated) implications between unknown propositions: line 6 inserts a *NOOP* implication if p is unknown at two adjacent layers, lines 7 to 10 insert effect-induced implications as explained above. In line 12, we call an algorithm that restricts the set of initial world states considered: only a subset $2^{\mathcal{I}}|_s$ of $2^{\mathcal{I}}$ leads to the observations in $\overline{ao}$. The propagate-observations-to-$\mathcal{I}$ procedure extracts a sufficient condition for $2^{\mathcal{I}}|_s$, i.e., it strengthens the initial state formula with a conjunction of propositions encoding a subset of $2^{\mathcal{I}}|_s$. We will consider that procedure in detail below; now, let's proceed in Figure 1. Lines 13 to 24 construct the relaxed planning graph from layer 0 onwards. In that procedure, on top of the sets $P(t)$ and $uP(t)$, we use sets $oP(t)$. These sets always contain those propositions in $uP(t)$ that can be observed by an earlier action. Line 13 is an initialization step, lines 14 to 24 repeatedly call the build-timestep procedure, that increments the planning graph by one level. There are two termination tests: line 14, if the goal condition is known in the new layer, we can extract a relaxed plan, see below; lines 17 to 22, if the graph reaches a fix point, we know a relaxed plan does not exist, therefore a real plan does not exist, and we can skip the search state. The action sets $A(t)$ are set to those actions whose preconditions are either achievable or observable with earlier actions. Note that, by allowing actions in $A(t)$ with preconditions that are only observed, not known to be true, we apply the generous execution semantics to these actions/preconditions. Pseudo-code for the build-timestep procedure is given in Figure 2.

procedure build-timestep(t, A)
1 $P(t+1) := P(t), uP(t+1) := uP(t)$
2 *Imp* $\cup = \{(p(t), p(t+1)) \mid p \in uP(t)\}$
3 **for** all $a \in A$ with $pre(a) \subseteq P(t)$, all $e \in detE(a)$ **do**
4   **if** $con(e) \subseteq P(t)$ **then** $P(t+1) \cup = add(e)$ **endif**
5   **if** $con(e) \subseteq P(t) \cup uP(t)$ and $con(e) \cap uP(t) \neq \emptyset$ **then**
6         $uP(t+1) \cup = add(e)$
7   select $c \in con(e) \cap uP(t)$
8   *Imp* $\cup = \{(c(t), p(t+1)) \mid p \in add(e)\}$
9   **endif**
10 **endfor**
11 **for** all $a \in A$ with $pre(a) \not\subseteq P(t)$, all $e \in detE(a)$ **do**
12     $uP(t+1) \cup = add(e)$
13   select $o \in pre(a) \cap oP(t)$
14   *Imp* $\cup = \{(o(t), p(t+1)) \mid p \in add(e)\}$
15 **endfor**
16 **for** all $p \in uP(t+1) \setminus P(t+1)$ **do**
17   **if** $\mathcal{I} \rightarrow \bigvee_{l \in \mathit{Impleafs}(p(t+1))} l$ **then** $P(t+1) \cup = \{p\}$ **endif**
18 **endfor**
19 $uP(t+1) \setminus = P(t+1)$
20 $oP(t+1) := uP(t+1) \cap (\{p \mid \exists a \in A(t) : p \in O(a)\})$

Figure 2: Building a new layer in the CRPG.

From top to bottom, the procedure does the following. Line 1 says that every proposition that will always be true at t, or that may be true at t, will a priori have the same property at $t+1$ (unknown propositions at $t+1$ may later be inferred to become known, see below). Line 2 inserts the NOOP implications between unknown propositions. Lines 3 to 10 treat those actions whose preconditions are known at t. The effects known to occur are handled in line 4, those that may occur depending on the initial world are handled in lines 5 to 9; the latter is done in a manner similar to what we have explained above. Lines 11 to 15 treat those actions with a precondition that is only observable at t. For these, implications are inserted between *an observed precondition* $o \in pre(a)$ and the added propositions of the (deterministic) effects. This corresponds to moving o – which is unknown at t – into the effect conditions, and selecting it as the only considered unknown one of these. That is, here we apply the generous execution semantics together with the relaxation that ignores all but one unknown effect condition.

Lines 16 to 18 contain the reasoning that is done to check whether an unknown proposition becomes known at $t+1$. For the respective p, it is checked whether the initial state formula implies the disjunction of the leaves of the *Imp* tree that are reachable from $p(t+1)$. *Impleafs*$(p(t'))$, for any t', is defined as

$$\mathit{Impleafs}(p(t')) := \{l \mid \exists \text{a path in } \mathit{Imp} \text{ from } l(-n) \text{ to } p(t')\}$$

The implication $\mathcal{I} \rightarrow \bigvee_{l \in \mathit{Impleafs}(p(t+1))} l$ is checked by a call to the SAT solver, and if the implication holds, p is inserted into the known propositions at $t+1$.[3]

In line 19 the procedure removes from the set of unknown propositions those that are now known. In line 20, finally, the observable propositions are set to those that are unknown and that are observed by an action in $A(t)$ (note that these include the observed propositions' inverse counterparts).

We will see below how a relaxed plan can be extracted from a successfully built CRPG. Let us first consider the procedure that computes (a sufficient condition for) the constraints imposed on the initial state by the observations made in $\overline{ao}$. The pseudo-code is provided in Figure 3.

procedure propagate-observations-to-$\mathcal{I}$($\langle a_{-n}, \ldots, a_{-1}\rangle$, P, uP)
1 $\Phi(0) := \emptyset$
2 **for** $t := -1 \ldots -n$ **do**
3     $\Phi(t+1) \cup = \{p \mid p \in o(a_t), p \in P(t+1)\}$
4     $\Phi(t) := \emptyset$
5   **for** all $p \in \Phi(t+1)$ **do**
6   **if** $p \in uP(t)$ **then** $\Phi(t) \cup = \{p\}$ **endif**
7   **if** $p \in uP(t) \cup P(t)$ **then**
8   **for** all $e \in E(a_t), con(e) \subseteq P(t) \cup uP(t), p \in del(e)$ **do**
9   select a fact $c \in con(e) \cap uP(t)$
10                 $\Phi(t) \cup = \{not\text{-}c\}$
11   **endfor**
12   **endif**
13   **if** $p \notin uP(t) \cup P(t)$ and
            $\exists e \in detE(a_t) : p \in add(e), con(e) \subseteq P(t) \cup uP(t)$ **then**
14   select one such e
15             $\Phi(t) \cup = \{con(e) \cap uP(t)\}$
16   **endif**
17   **endfor**
18 **endfor**
19 $\mathcal{I} := \mathcal{I} \wedge \bigwedge_{p \in \Phi(-n)} p$

Figure 3: Propagating (a sufficient condition for) the constraints given by the observations into the initial state formula.

The motivation for using an approximative procedure, computing a sufficient condition in the form of a simple conjunction of propositions, is that the real constraints imposed by the observations can be an arbitrarily complex formula. Our approximation proceeds as follows. Lines 1 to 18 implement a simple backwards loop over the actions in $\overline{ao}$. The loop invariant is that, after an iteration t has terminated, if all propositions in $\Phi(t)$ are true at t, then the observations made by actions at $t' \geq t$ will turn out as $l(a_{t'})$, i.e., as required to reach the considered belief state. In line 3, the observations made by a_t are inserted into $\Phi(t+1)$. The new set $\Phi(t)$ is initialized as empty. For each proposition p in $\Phi(t+1)$, we then insert into $\Phi(t)$ a sufficient condition for p to hold at $t+1$. p was either kept true from

[3]It is relatively easy to see that this method is complete and sound relative to our relaxation: the paths in *Imp* from a leaf to $p(t+1)$ correspond to all possible ways of making p true at $t+1$, and p will always be true at $t+1$ iff one of these paths will be executed from every initial world state.

t, or added by a_t. For our sufficient condition, we consider just one of these cases, with a preference on the first, NOOP, case. This is available if $p \in uP(t) \cup P(t)$. The real NOOP implication for p to hold at $t+1$ has the form $p(t) \wedge (\neg c_1(t) \vee \ldots \vee \neg c_k(t)) \wedge \ldots \wedge (\neg c'_1(t) \vee \ldots \vee \neg c'_{k'}(t)) \rightarrow p(t+1)$ where the c_i and c'_i are the conditions of the effects of a_t that delete p. Lines 6 to 12 select, for each such effect, just one of its (negated) conditions into the created conjunction. Note that propositions in $P(t)$ need not be selected since these will be true anyway. If the NOOP case is *not* available, i.e. if p was false at t, then a_t must have a possibly occurring add effect on p. If the only such effect is non-deterministic, there is no condition at t that guarantees p to hold at $t+1$, and no constraint is propagated. Otherwise, an implication for p to hold at $t+1$ is $(c_1(t) \wedge \ldots \wedge c_k(t)) \vee \ldots \vee (c'_1(t) \wedge \ldots \wedge c'_{k'}(t)) \rightarrow p(t+1)$ where the c_i and c'_i are the conditions of the deterministic effects of a_t that add p. Lines 13 to 16 select the condition of only one such effect into the created conjunction. Upon termination, in line 19 the initial state formula is strengthened with the conjunction created at time 0. This stronger formula will be used in the implication checks during the rest of the relaxed planning process, thereby taking account of the smaller set of initial worlds relevant for the belief state at hand. Of course, after relaxed planning has finished, $\bigwedge_{p \in \Phi(-n)} p$ is removed from the initial state formula again.

procedure extract-CRPlan($CRPG(\langle a_{-n}, \ldots, a_{-1} \rangle, A, \mathcal{I}, G), G$)
1 sub-goal(G)
2 **for** $t := Glayer, \ldots, 1$ **do**
3   **for** all $g \in G(t)$ **do**
4   **if** $\exists a \in A(t-1), pre(a) \subseteq P(t-1)$,
         $e \in detE(a), con(e) \subseteq P(t-1), g \in add(e)$ **then**
5   select one such a /* and e */ at $t-1$
6   sub-goal($pre(a) \cup con(e)$)
7   **else**
8   L is a minimal subset of *Impleafs*$(g(t))$ s.t. $\mathcal{I} \rightarrow \bigvee_{l \in L} l$
9   **for** all $i \geq 0$, actions $a \in A(i)$, and effects
10             $e \in detE(a)$ s.t. e is responsible for an edge in
11   a path from $l(-n), l \in L$, to $g(t)$ **do**
12   select a at i
13   sub-goal($(pre(a) \cup con(e)) \cap P(i)$)
14   o-sub-goal($pre(a) \cap oP(i)$)
15   **endfor**
16   **endif**
17   **endfor**
18   **for** all $g \in oG(t)$ **do**
19   select $a \in A(t-1), g \in o(a)$
20   sub-goal($pre(a) \cap P(t-1)$)
21   o-sub-goal($pre(a) \cap oP(t-1)$)
22   **endfor**
23 **endfor**

procedure sub-goal(P): for all $p \in P$, $G(min\{t \mid p \in P(t)\}) \cup= \{p\}$

procedure o-sub-goal(P): for all $p \in P$, $oG(min\{t \mid p \in oP(t)\}) \cup= \{p\}$

Figure 4: Extracting a conformant relaxed plan.

Extraction of a relaxed plan is done when the CRPG was built successfully, reaching the goals. The process is described in the pseudo-code in Figure 4. It makes use of sub-goal (proposition) sets $G(1), \ldots, G(Glayer)$, storing the goals and sub-goals arising at layers $1 \leq t \leq Glayer$, where $Glayer := max_{g \in G} min\{t \mid g \in P(t)\}$ is the layer where all goals were reached. The procedure additionally uses "observation goal" sets $oG(1), \ldots, oG(Glayer)$, which keep track of the propositions that are needed to be moved from action preconditions into effect conditions (as made possible by earlier observations). The observation goals construction is not a genuine part of the relaxed planning process, it is rather an ad-hoc enhancement we added in order to get an estimate of how many observation actions are necessary in reality, and how many actions are needed to achieve their preconditions. More on this below. Now, let's explain Figure 4 from top to bottom. By line 1, all goals are inserted into the G sets at their respective first layers of appearance. Lines 2 to 23 implement a backwards loop over the CRPG layers. In each layer t, lines 3 to 17 select supporting actions for the goals $g \in G(t)$. If there is an action (and effect) a at layer $t-1$ that guarantees to always achieve g, then a is selected at $t-1$. Otherwise, a minimal subset L of *Impleafs*$(g(t))$ is determined such that $\mathcal{I} \rightarrow \bigvee_{l \in L} l$, and all actions are selected, at the respective times, that are responsible for the implication paths from L at $-n$ to g at t. Selection of actions is to be understood as a set union operation, i.e. at each time step each action is selected at most once.

Without the constructs dealing with observation goals, the actions selected by the algorithm form a plan that works under the following relaxations: (1) All delete lists are empty; (2) There is only a single (the selected) unknown condition per action effect; (3) Where needed, preconditions are moved into effect conditions (i.e. the generous execution semantics is applied to the observation goal preconditions). As said, the actions selected for achievement of observation goals, and the resulting sub-goals, are not necessary for a relaxed plan and only form an ad-hoc extension to have some estimation of the observations necessary in reality. The heuristic value to the belief state is the total number of selected actions. For example, consider the initial state of a Bomb-in-the-toilet problem where the bomb b may be **in**(b, p_i) one of three packages p_1, p_2, p_3, p_i can be **flush**ed down the toilet only if it is known that **in**(b, p_i), and one can **observe** if **in**(b, p_i). Then the found relaxed plan is **flush**(b, p_1), **flush**(b, p_2), **flush**(b, p_3) where, with the relaxed execution semantics, the **in**(b, p_i) preconditions have become effect conditions. The observation actions that made that possible are **observe**(b, p_1), **observe**(b, p_2), **observe**(b, p_3), and the overall heuristic value is 6 – in this case, the true optimal plan length.

Stagnating States

In our current implementation, we do only a limited form of repeated states checking. If a belief state s' is generated that descends from a belief state s on a path of normal actions, and we can prove that s' is equal to s, then we say that s' *stagnates* and prune it from the search space – if there is a plan tree that goes through s' then one can skip the bit of the tree between s and s'. The motivation for the limited test is that, to check for stagnating states, every generated state s' has to be compared only with (part of) its ancestors. Our experience from conformant planning is that comparing s' with *all* seen states can be very expensive, given the effort involved in doing a single such comparison based on our belief state representation.

Without non-deterministic effects, belief state comparison is very similar to what is done in Conformant-FF. We did not find a way to test equality based on our belief state representation. Instead, we test a stronger criterion which we call belief state *equivalence*. Let s and s' be two belief states reached via the action sequences $\overline{ao}$ and $\overline{ao}'$, respectively. We say that s is equivalent to s' iff $\forall I \in 2^{\mathcal{I}} : \overline{ao}(\{I\}) = \overline{ao}'(\{I\})$, i.e. from every initial world both ao-sequences result in the same world state (in particular, s and s' are reached from the same initial world states). When s' is a descendant of s on a path of non-observation actions (thus $\overline{ao}$ is a prefix of $\overline{ao}'$), equivalence can be tested based on satisfiability checks as follows. For a proposition p, let v and v' denote p's value (the respective CNF variable) following $\overline{ao}$ and $\overline{ao}'$, respectively. Then s is equivalent to s' iff the formulas $\phi(\overline{ao}') \wedge v \wedge \neg v'$ and $\phi(\overline{ao}') \wedge \neg v \wedge v'$ are unsatisfiable for all p. I.e., we can re-use the formula generated for s', and see whether all propositions have the same value, from all initial world states, in s and s'. [4]

In the presence of non-deterministic effects, one can in principle use the same SAT tests to prune stagnating states. The problem is that they now encode too strict a sufficient criterion: they ask if it is the case that, for *every* initial world and for *every* outcome of the non-deterministic effects, the resulting world state in s is the same as that in s'. Consider, e.g., a single action a that non-deterministically adds a proposition p that is initially false. If s results from applying a once, and s' results from s by applying a another time, then of course the two belief states are equal (namely, $s = s' = \{\emptyset, \{p\}\}$). But p may be false after $\langle a \rangle$ and true after $\langle a, a \rangle$ and so the above test fails. To handle such cases, in our current implementation we weakened our equivalence criterion as follows: we test if it is the case that, *for every outcome of the non-deterministic effects up to s, there exists an outcome of the non-deterministic effects between s and s', so that for every initial world the resulting proposition values in s and s' are the same.* If so, $s \subseteq s'$ is proved and s' can be pruned. Note that the test holds in the above $\langle a \rangle$/$\langle a, a \rangle$ example. The test is done, in our current code, by a naive enumeration of the outcomes nds of the non-deterministic effects up to s, and of the outcomes nds' of the non-deterministic effects between s and s'. At the bottom of this enumeration, for all (unknown) propositions p the same SAT test as before is used: we see whether $\phi(\overline{ao}', nds, nds') \wedge v \wedge \neg v'$ and the inverse are unsatisfiable, where $\phi(\overline{ao}', nds, nds')$ is the formula corresponding to $\overline{ao}'$ with the effects nds, nds'. Note that, for a single proposition, this is basically a naive QBF solver on a formula of the form $\forall nds : \exists nds' : \forall I : \neg(\phi(\overline{ao}', nds, nds') \wedge v \wedge \neg v')$. Indeed, solving an $\exists\forall\exists$-QBF can be easily reduced to the complement of this equivalence test for a single proposition, implying that the test is Π^p_3-complete.

We remark that our current equivalence test is stronger than necessary. To prove $s \subseteq s'$, it would suffice to check if it is the case that, for *every* initial world and for *every* outcome of the non-deterministic effects up to s, there *exists* an outcome of the non-deterministic effects between s and s' so that the resulting proposition values in s and s' are the same. The problem with this better test is that, when implemented naively, it involves enumerating *the initial worlds*. Seeing if the test can be made efficient by using a non-naive QBF-solver is a topic for future work.

[4] Of course, s and s' need only be compared if they agree on the sets of known/negatively known/unknown propositions; then, the SAT tests need be done only for the unknown p.

Results

We implemented the techniques described in the previous sections in C, starting from the Conformant-FF code. We call the implemented system Contingent-FF. The system's overall architecture is that of Conformant-FF, except that we now use a weighted AO* search. In the experiments reported below, the weight was set to 5 (like in POND). The cost of an observation node is computed as (one plus) the *maximum* of the costs of its sons – that is, the search tries to minimize the depth of the solution tree, corresponding to the worst-case execution time of the plan. Taking the maximum, rather than the sum as we did in an earlier version of our system, also has an important impact on search performance; we get back to this below. The underlying SAT solver in Contingent-FF is a naive DPLL implementation. Contingent-FF is given as input a PDDL-like file describing the domain and the task, with obvious modifications for describing uncertainty about the initial state and observation actions.

We experimentally compare Contingent-FF to the state-of-the-art contingent planners POND (Bryce, Kambhampati, & Smith 2004) and MBP (Cimatti & Roveri 2000; Bertoli *et al.* 2001; Bertoli & Cimatti 2002). We use a testbed of 11 domains, of which 6 are taken from the testbed provided with the POND planner, and 5 we created ourselves.[5] The experiments were run on a PC running at 1.2GHz with 1GB main memory and 1024KB cache running Linux, with a runtime cutoff of 900 seconds. The POND domains are the following: BTS and BTCS, two variants of the Bomb-in-the-toilet domain, with a single toilet, without and with clogging of the toilet, respectively. Medical and MedPKS, which require treating a disease based on the results of diagnostic tests; MedPKS (introduced with the PKS (Petrick & Bacchus 2002) system) is a simplified and scalable form of Medical. Logistics and Rovers, which are classical benchmarks enriched with (observable) uncertainty about the initial positions of the packages and the positions of rock/soil/images, respectively. In all POND domains we run a subset of the instances provided with the POND collection, except in Logistics were we also run one (larger) instance we generated ourselves. Our own domains are: BTND, a non-deterministic version of BTCS where flushing a package non-deterministically deletes a new fact ("unstuck(toilet)") that then must be re-achieved. Omlette, the well known problem that is unsolvable due to a non-deterministic effect spoiling the bowl when breaking an (a bad) egg into it. Unix, motivated by the Unix domain

[5] The entire set of test examples can be downloaded from http://www.mpi-sb.mpg.de/∼hoffmann/contff-tests.tgz

of (Petrick & Bacchus 2002), in which one must copy or move a file from one directory to another. Blocks, where there is uncertainty about the initial arrangement of the top 2 blocks on each stack, and one must observe the block positions before proceeding. Grid, where there is (observable) uncertainty about the shapes of the locked positions. Table 1 gives the results for all the domains.

	FF	FF+H	POND	MBP
Instance	$t/S/D$	$t/S/D$	$t/S/D$	$t/S/D$
BTS10	0.10/19/10	0.01/10/10	0.37/19/10	0.00/10/10
BTS30	3.92/59/30	2.53/30/30	11.86/59/30	0.01/30/30
BTS50	45.04/99/50	40.17/50/50	154.25/99/50	0.06/50/50
BTS70	266.13/139/70	317.10/70/70	–	0.16/70/70
BTCS10	0.04/19/10	0.01/19/19	0.23/19/10	0.01/20/20
BTCS30	4.56/59/30	5.74/59/59	13.00/59/30	0.27/60/60
BTCS50	57.45/99/50	111.87/99/99	168.16/99/50	0.84/100/100
BTCS70	342.82/139/70	895.91/139/139	–	1.73/140/140
Medical2	0.00/4/3	0.00/NS	0.12/4/3	0.00/5/4
Medical3	0.00/8/6	0.00/NS	0.14/7/5	0.00/7/5
Medical4	0.00/12/6	0.00/NS	0.14/10/5	0.01/10/5
Medical5	0.00/15/8	0.00/NS	0.15/12/6	0.01/12/6
MedPKS10	0.14/20/11	0.00/NS	0.55/20/11	8.71/20/11
MedPKS30	20.82/60/31	0.00/NS	14.86/60/31	–
MedPKS50	350.09/100/51	0.00/NS	192.52/100/51	–
MedPKS70	–	0.00/NS	–	–
Logistics1	0.01/10/7	0.01/11/11	0.20/10/7	0.21/29/29
Logistics3	0.01/21/11	0.01/16/16	0.86/18/9	–
Logistics5	0.53/160/23	0.11/29/29	18.45/130/23	–
Logistics7	0.79/228/24	0.39/36/36	21.32/178/23	–
LogisticsL	43.60/2401/78	2.04/126/126	–	–
Rovers2	0.00/ 11/7	0.00/9/9	0.81/30/10	–
Rovers4	0.04/21 /11	0.01/15/15	0.53/20/10	–
Rovers6	0.49/181/28	0.03/31/31	11.10/107/21	–
Rovers8	0.12/62 /25	0.02/25/25	4.95/48/21	–
BTND10	0.40/53/13	51.40/28/28	–	0.01/32/32
BTND30	12.22/173/33	–	–	0.50/92/92
BTND50	136.66/293/53	–	–	0.96/152/152
BTND70	887.72/413/73	–	–	2.60/212/212
Omlette1	0.00/UNS	0.00/NS	SF	0.01/UNS
Omlette2	0.20/UNS	0.00/NS	SF	0.02/UNS
Omlette3	–	0.13/NS	SF	0.02/UNS
Unix1	0.02/17/14	0.00/17/14	5.10/26/21	11.58/21/18
Unix2	0.41/48/37	0.18/48/37	MEM	–
Unix3	10.08/111/84	7.58/111/84	MEM	–
Unix4	525.02/238/179	395.50/238/179	MEM	–
Blocks3	0.00/7/5	0.01/8/6	0.20/10/6	0.33/7/5
Blocks7	0.36/56/10	0.51/56/10	–	–
Blocks11	3.20/116/19	1.67/111/17	–	–
Blocks15	43.20/327/85	11.86/241/54	MEM	–
Grid2	0.35/ 47/15	0.02/13/13	MEM	–
Grid3	1.33/269/39	0.06/23/23	MEM	–
Grid4	2.17/853/80	0.40/49/49	MEM	–
Grid5	2.70/1095/231	0.63/68/68	MEM	–

Table 1: t is runtime (in sec.), S/D is size/depth of the plan tree, dashes indicate time-outs, MEM: out of memory, NS: planner output "no solution found", UNS: proved unsolvable, SF: segmentation fault.

We ran Contingent-FF with (FF+H) and without (FF) a straightforward adaptation of FF's helpful actions technique (which prunes successors of a state s that are not generated by actions considered useful by the relaxed plan for s). Since the generated plans are trees of actions, to indicate their quality we provide both tree size and depth.

In the BTS and BTCS domains, Contingent-FF is a little more efficient than POND; MBP is a lot faster but finds non-branching plans in all cases. Medical is not a challenge to any of the planners, except for FF+H where the helpful actions pruning cuts out the observation actions and, with that, all solutions. This makes the planner stop unsuccessfully after exploring just a few search states. In MedPKS, POND scales somewhat better than FF, MBP is very bad, and FF+H behaves exactly like in Medical. In Logistics and Rovers, MBP does not scale, and FF scales better than MBP. Regarding plan quality, we observe that POND is in most cases somewhat better than FF, and that FF+H finds non-branching plans with worse depth values. We will come back to the latter below. In BTND, POND behaves very badly, and MBP scales by far best, again at the cost of bad-quality plans. FF+H behaves very badly because, like in Medical and MedPKS, the observation actions are cut out; in difference to Medical and MedPKS, however, this does not lead to an empty search space quickly but makes the planner search large fruitless regions of the belief space. We could not get POND to run in our (unsolvable) Omlette domain; MBP scales far better here than FF, which spends too much time in the tests for repeated states. For Unix, FF is the only reasonably efficient solver; note that the tasks (their underlying directory structure) scale exponentially in the size parameter. In Blocks, FF is clearly better than POND and MBP, scaling up quite comfortably and finding branching plans. Interestingly, here helpful actions pruning improves both runtime and plan quality significantly. In Grid, POND and MBP can't even solve our smallest instance, while again FF scales relatively well. Similarly to Logistics and Rovers, FF+H is faster and finds non-branching plans; these plans are, however, smaller in depth than the branching plans found by FF. The latter contain a lot of superfluous actions. It is unclear to us what the reason for this odd behavior is.

It is important to note that, in all examples where POND ran out of memory, it did so during the building of its Labeled Uncertainty Graph (the basis for its heuristic computation). If that is due to details in our encoding of the examples, or to inefficiencies in POND's implementation, or to real weaknesses of POND's approach, is hard to tell.

One interesting observation in the above concerns the precise encoding of the interplay between observation actions and conditional/unconditional action effects. In conformant test domains, actions can handle different world states by effects that occur only when a condition c is satisfied. E.g., in BTS, dunking a package into a toilet disarms the bomb only if the bomb is in the package. In the POND contingent test suites, and in our Grid suite, the conformant benchmarks were modified only by the introduction of additional observation actions. This gives planners the opportunity to do observations, but it does not *enforce* them. An alternative encoding is to move the conditions c into the preconditions of the respective actions, making it impossible to apply such an action without knowing if or if not the relevant effect will occur. The affected domains – the domains whose version treated above does not enforce observations – are BTS, BTCS, BTND, Logistics, Rovers, and Grid. Enforcing observations in these domains often changes planner performance considerably. Table 2 shows the results for the modified domains, indicated by a leading "e" in their names.

The most striking observation in eBTS, eBTCS, and eBTND, as compared to BTS, BTCS, and BTND, is that MBP completely loses its ability to scale up. In eBTS and eBTCS, Contingent-FF (in both versions) and POND become somewhat better, in eBTND it is only FF+H that im-

	FF	FF+H	POND	MBP
Instance	$t/S/D$	$t/S/D$	$t/S/D$	$t/S/D$
eBTS10	0.06/19/10	0.00/19/10	0.16/19/10	0.46/56/10
eBTS30	2.23/59/30	2.11/59/30	5.72/59/30	–
eBTS50	28.80/99/50	26.67/99/50	76.41/99/50	–
eBTS70	166.89/139/70	160.75/139/70	474.94/139/70	–
eBTCS10	0.04/19/10	0.02/19/10	0.17/19/10	1.07/30/11
eBTCS30	3.71/59/30	2.19/59/30	5.99/59/30	–
eBTCS50	30.05/99/50	27.78/99/50	78.56/99/50	–
eBTCS70	174.76/139/70	165.03/139/70	488.43/139/70	–
eBTND10	0.23/28/12	0.02/28/12	–	–
eBTND30	9.56/88/32	0.82/88/32	–	–
eBTND50	132.65/148/52	7.90/148/52	–	–
eBTND70	860.26/208/72	39.88/208/72	–	–
eLogistics1	0.00/10/7	0.00/10/7	0.20/10/7	15.31/20/15
eLogistics3	0.01/21/11	0.00/19/9	0.86/18/9	–
eLogistics5	0.42/160/23	0.14/156/23	18.44/130/23	–
eLogistics7	0.66/228/24	0.21/223/23	21.14/178/23	–
eLogisticsL	44.68/3389/78	25.95/6590/138	–	–
eRovers2	0.00/11/7	0.00/11/8	0.79/30/10	–
eRovers4	0.02/21/11	0.00/21/11	0.53/20/10	–
eRovers6	0.15/169/23	0.08/157/21	10.95/107/21	–
eRovers8	0.03/59/22	0.02/57/21	4.79/48/21	–
eGrid2	0.10/43/14	0.01/13/13	MEM	–
eGrid3	0.53/283/39	0.12/65/29	MEM	–
eGrid4	1.91/893/67	0.85/335/47	MEM	–
eGrid5	2.39/1089/231	1.46/460/137	MEM	–

Table 2: Results with enforced observations.

proves, dramatically in this case. In the classical domains enriched with uncertainty, the main difference under enforced observations lies in the different behavior of FF+H. Whereas, previously, FF+H found non-branching plans, it now behaves more similarly to FF, improving on runtime in all cases, and on plan quality in all cases except the large Logistics example.

In an earlier version of Contingent-FF, we ran an AO* search that tried to minimize solution tree *size* rather than depth, taking the cost of a node to be the *sum* of the costs of its children, rather than the maximum as we do now. With enforced observations, the earlier version of Contingent-FF behaved a lot worse, being able to solve only the smallest examples in Blocks, eLogistics, eRovers, and eGrid. The reason for this is that, in these domains, many actions have to be *re-used* in all branches of the solution. This is not taken account of in our heuristic function, introducing, under the *sum* operator, a strong bias to not do observations.

Discussion

We described a contingent planner, Contingent-FF, that builds on our conformant planner, Conformant-FF, extending its sparse belief space representation, and the heuristic techniques building on it, to handle observation actions and non-deterministic action effects. Contingent-FF is competitive with POND and MBP.

Our planner is unique in its belief state representation, but is only one among a number of contingent planners, many of which are quite recent. One of the first contingent planners is CNLP (Peot & Smith 1992), which used non-linear planning techniques. More related are Graphplan-based planners, the first being SGP (Weld, Anderson, & Smith 1998), which used multiple planning graphs, one for each possible initial world, keeping track of the relationship between these graphs. POND (Bryce, Kambhampati, & Smith 2004) is a more recent descendant of this lineage which replaced the use of multiple planning graphs with a single planning graph in which propositions/actions are labeled with formulas describing the initial worlds under which they can be true/can be applied. Contingent-FF is a forward-search planner in the space of belief-states, and was preceded by GPT (Bonet & Geffner 2000) and MBP (Bertoli *et al.* 2001), both of which use the same idea. The main differences between these planners are the belief state representations used and the particular techniques used for generating a heuristic.

The most important open topic for Contingent-FF is, in our opinion, to extend the system to richer forms of non-deterministic effects. In particular, faster and/or more powerful techniques for repeated states checking in the presence of such effects must be developed. Note that, *conceptually*, it is not difficult to modify our approach to non-deterministic effects that result in a list of arbitrary alternative outcomes, rather than occurring or not occurring individually. One just needs to encode the possible outcomes in CNF format, and include that into the formula describing the action sequence. Note also that one can, *conceptually*, easily integrate observation actions with multiple outcomes, including cases where an observation failed (to make this make sense, one needs a slightly more powerful search algorithm able to construct loops in the plan). Of course, it is not a priori clear how well such approaches would work in practice.

References

Bertoli, P., and Cimatti, A. 2002. Improving heuristics for planning as search in belief space. In *Proc. AIPS'02*, 143–152.

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. MBP: a model based planner. In *Proc. IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information, Seattle, August 2001.*

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS'00*, 52–61.

Brafman, R. I., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *Proc. ICAPS'04*, 355–364.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2004. Planning in belief space with a labelled uncertainty graph. In *Proc. AAAI'04 Workshop on Learning and Planning in Markov Decision Processes*.

Cimatti, A., and Roveri, M. 2000. Conformant planning via symbolic model checking. *JAIR* 13:305–338.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Peot, M., and Smith, D. E. 1992. Conditional non-linear planning. In *Proc. AIPS'92*, 189–197.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. AIPS'02*, 212–221.

Weld, D. S.; Anderson, C.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proc. AAAI'98*, 189–197.