Conformant Planning via Heuristic Forward Search

Ronen I. Brafman
Dept. of Computer Science
Ben-Gurion University
Beer-Sheva 84105, Israel

Jörg Hoffmann
Dept. of Computer Science
Albert-Ludwigs University
79110 Freiburg, Germany

Abstract

Conformant planning is the task of generating plans given uncertainty about the initial state and action effects, and without any sensing capabilities during plan execution. The plan should be successful regardless of which particular initial state we are in. This paper is motivated by the observation that (1) Conformant planning without conditional effects can be handled easily by any forward search planner; (2) To handle conditional effects, one can reason about the set of known facts following each sequence of actions, rather than explicitly enumerating the sets of possible worlds. Using this computation of known facts, we extend the classical planning system FF to the conformant setting. Our experimental evaluation shows Conformant-FF to be superior to the state-of-the-art conformant planner MBP in a variety of benchmark domains.

Introduction

Conformant planning is the task of generating plans given uncertainty about the initial state and action effects, and without any sensing capabilities during plan execution. The plan should be successful regardless of which particular initial state we are in and which action effects occur.

It is well known that conformant planning can be transformed into a search problem in belief space (Bonet & Geffner 2000), i.e., the space whose elements are sets of possible worlds. This way, our uncertainty about the true current state is modeled via the set of states that we consider possible at this time. Unfortunately, the number of states in belief space is doubly exponential in the set of propositions, and so a naive implementation of this idea is bound to fail. Yet, by carefully representing this set of states using BDDs, and by carefully generating a heuristic function that takes into account levels of uncertainty, the state-of-the-art conformant planner MBP (Bertoli & Cimatti 2002) provides a more practical implementation of this idea.

Our work, too, uses the idea of search in belief space, but is based on a somewhat different representation and builds on the ideas of the classical planning system FF (Hoffmann & Nebel 2001). To get an idea of our approach, consider the problem of conformant planning with actions that have no conditional effects. In that case, one needs only pay attention to those facts that hold in all possible worlds, i.e., the *known* facts. Since each action has to be applicable in all the possible worlds, its preconditions must be known prior to its execution, and its effects will necessarily become known following its execution. Thus, using any forward search planner, we can perform conformant planning with unconditional effects: Instead of representing the current state of the world using a set of propositions, the planner uses such a set to represent the set of facts that hold in *all* currently possible worlds. Indeed, an initial exercise we conducted to test this idea showed that FF with *no* modifications what-so-ever solves the infamous Bomb-in-the-toilet problem in no time at all (roughly, linear in the number of bombs and toilets).

Conformant planning becomes truly challenging when actions with conditional and non-deterministic effects are introduced. To handle this case, our forward-chaining approach introduces an ability to reason about the set of facts that are known to hold upon executing a sequence of actions. This knowlede is reminiscent of the use of epistemic knowledge by Petric and Bacchus in their contingent planner (Petrick & Bacchus 2002), but whereas Petric and Bacchus feed the knowledge to their planner in their domain descriptions, we infer the knowledge *automatically*. The corresponding decision problem is hard – we show it to be co-NP complete. In fact, our implementation performs this reasoning by deciding solvability of a CNF formula that captures the semantics of the action sequence. While this may sound impractical, our intuition is that modern SAT solvers, which can easily handle problems with tens of thousands of variables and clauses, will have little trouble reasoning about the effects of an action sequence if the possible interactions are not overly complex. Indeed, this intuition is supported quite impressively by excellent results we obtained in a variety of benchmark domains using a *completely naive* DPLL implementation as the underlying SAT solver.

The symbolic encoding of the action sequence semantics we use is relatively efficient, but is far from sufficient on its own. One of the major strengths of the FF system in the classical (STRIPS and ADL) setting is its efficiently computable and highly informative heuristic function. This function is obtained by observing the length of a *relaxed plan* generated from the original problem by ignoring the delete lists. One of the main contributions of this work is the adaptation of this approach to the conformant setting. Basically, we enrich the relaxed planning process by an efficiently handable reasoning technique that uses a stronger version of the propositional formula that captures the true state of knowledge. The stronger formula can be thought of as a 2-CNF projection of the original formula. The implication graph (Aspvall, Plass, & Tarjan 1979) of this 2-CNF theory supports linear time reasoning, and is naturally embedded into the relaxed planning graph used by FF to compute its heuristic function.

Our experimental results in a number of traditional conformant benchmarks show that our implementation, which we call Conformant-FF, is competitive with the state-of-the-art conformant planner MBP in most of these domains, scaling in fact much better in some of them. Moreover, we show that our approach has the potential to *combine* the strengths of FF with conformant abilities in domains that combine classical and conformant aspects. In a number of classical planning benchmarks enriched with uncertainty, Conformant-FF shows fine scalability, dramatically outperforming MBP.

The paper is organized as follows. The next section briefly describes the planning framework we consider. Section then explains FF's algorithms for STRIPS and ADL, as well as their extension to the conformant setting. Section gives our empirical results, Section discusses related work, and Section contains a discussion of our contributions, and of future research directions. A longer version of this paper containing proofs and additional technical details is available at http://www.informatik.uni-freiburg.de/∼hoffmann/cff-report.ps.gz.

Planning Background

The conformant planning framework we consider is an extension to the classical STRIPS (Fikes & Nilsson 1971) and ADL (Pednault 1989) languages. Propositional planning tasks are triples (A, I, G) corresponding to the *action set, initial state,* and *goal state.* World states w are represented using the sets of propositions satisfied in them. In STRIPS, actions a are simple triples $(pre(a), add(a), del(a))$ of proposition sets, corresponding to the *precondition, add,* and *delete* lists. An action a is *applicable* in a world state w if $w \supseteq pre(a)$. The result of applying a in w is the state $w - del(a) + add(a)$. We assume that $del(a) \cap add(a) = \emptyset$.[1] ADL is more complex, allowing for action preconditions containing closed first-order logical formulae, conditional effects (whose conditions are, again, logical formulae), and a formula as the goal condition. We denote conditional effects e as triples $(con(e), add(e), del(e))$. An action a is applicable in a state w if $w \models pre(a)$ holds. Applying a in w results in updating the state with all effects that *occur*, i.e., whose condition holds in w. Again, we postulate that there be no intersection of added and deleted propositions. In both frameworks, an action sequence is a *plan* if the result of its execution fulfills the goal.

The conformant setting we consider is that of ADL planning tasks, with additional uncertainty about the initial state. We allow to express such uncertainty by specifying propositions p as *unknown* initially, meaning that both truth values of p yield a possible initial state. We also allow dependencies between sets of propositions, in the form of *initial disjunctions* $l_1 \vee \ldots \vee l_k$ of literals (where l_i is an initially unknown proposition or its negation), meaning that all possible initial states satisfy this disjunction. We also allow *non-deterministic effects*, meaning that one does not know whether the respective effect will occur or not. Our implementation does yet not support non-deterministic effects, but integrating them is conceptually easy.

We denote the *description* of the set of possible initial states with $\mathcal{I}$. This contains propositions that are known to true, $p \in \mathcal{I}$, and propositions known to be false, $\neg p \in \mathcal{I}$, which we refer to as *known* and *negatively known* propositions, respectively. In the input files, the negatively known propositions are implicitly given as the set of all propositions in the task except those that are known or unknown. In addition, our initial states description contains disjunctive constraints $l_1 \vee \ldots \vee l_k \in \mathcal{I}$. Slightly abusing the powerset notation, the (possibly exponentially larger) *set* of possible initial states is denoted with $2^{\mathcal{I}}$. An action sequence is a plan if, for any possible initial world state $I \in 2^{\mathcal{I}}$, executing the sequence in I results in a world state that fulfills the goal. Note that a plan can succeed in different initial world states because actions with conditional effects can produce different results in different situations.

Conformant-FF

FF is a classical planning system based on a simple heuristic search approach (first proposed by Bonet and Geffner (1998; 2001)). Search takes place forward in the state space, guided by a heuristic function that estimates the distance to the goal by the length of a *relaxed plan*: an action sequence that achieves the goal when assuming the delete lists are all empty. With empty delete lists, the planning task is simpler and can be solved efficiently. FF does this in every search state (Hoffmann & Nebel 2001). We denote the heuristic value of a state w by $h(w)$. This is the number of actions in a relaxed plan from w. The set of actions that can start such a plan (details below in Section) is denoted by $H(w)$.

The overall architecture of Conformant-FF is virtually the same as FF's. An overview is this:

1. Compile away in a preprocess all ADL constructs except conditional effects.
2. Do one trial of "enforced hill-climbing": set the current search sate s to $s := I$; while $h(s) \neq 0$ do:
   (a) Starting from s, perform breadth-first search for a state s' such that $h(s') < h(s)$; during search, avoid repeated states; cut out states s' with $h(s') = \infty$; and expand only the successors of s' generated by the actions in $H(s')$.
   (b) If no s' with $h(s') < h(s)$ has been found then fail, else $s := s'$;
3. If enforced hill-climbing failed, envoke complete best-first search, i.e., starting from the initial state expand all states in order of increasing h value, avoiding repeated states.

To adopt this approach, we need to redefine the following key techniques in the context of conformant planning: ADL compilation, search states and state transitions (which are non-trivial in the conformant setting); relaxed plans; and repeated states checking. The other parts of the architecture remain exactly the same as in the classical planner. The following four subsections correspond in turn to the four techniques we need to extend.

[1]This conservative approach is required to make the compilation of actions in ADL correct, see Section .

ADL Compilation

ADL formulae can contain negations. In the context of negations, the technique of ignoring delete lists does not yield a simplified planning problem. In the classical setting, FF compiles negations away following Gazen and Knoblock (1997). First, all formulae are brought into negation normal form (negations only in front of propositions). Then, for an occurence of $\neg p$ a new proposition $not\text{-}p$ is introduced and inserted in an inverse manner to p into the action effects (for adds on p a new delete on $not\text{-}p$ is inserted into the same effect, and vice versa), and the initial state ($not\text{-}p$ is in I iff p is not); $not\text{-}p$ is then true in a reachable state iff p is false.[2] One can thus replace occurences of $\neg p$ with $not\text{-}p$. The same methodology works in the conformant setting, except for the way we treat the initial state: we now insert $not\text{-}p$ into $\mathcal{I}$ if $\neg p \in \mathcal{I}$, and we insert $\neg not\text{-}p$ into $\mathcal{I}$ if $p \in \mathcal{I}$. If neither case holds, i.e., if p is initially unkown, we leave $not\text{-}p$ initially unkown, too, and insert the initial disjunctions $p \vee not\text{-}p$ as well as $\neg p \vee \neg not\text{-}p$.

With other simple compilation techniques, one can compile away quantifiers and disjunctions in formulae (Gazen & Knoblock 1997). These techniques remain the same in the conformant setting. We end up with an action and goal description that differs from STRIPS only in that action effects can have proposition sets as their individual effect condition (plus, in the conformant case, the possibly non-deterministic effect occurences).

States, and State Transitions

Search states s in our framework are the endpoints of action sequences (to be executed in the initial state). As opposed to *world* states w which are simply the sets of propositions that are true, a search state represents a *set* of possible world states: the action sequence can have different outcomes depending on the initial world state. For each search state, we compute the sets of known and negatively known propositions, which are defined as follows. Given a conformant planning task $(A, \mathcal{I}, G)$, a search state s corresponding to an action sequence $P \in A^*$, and a proposition p, we say that p is *known* in s if, for all $I \in 2^{\mathcal{I}}$, executing P in I results in a world state that contains p. We say that p is *negatively known* in s if for all $I \in 2^{\mathcal{I}}$ executing P in I results in a world state that does not contain p. A proposition that is neither known nor negatively known is *unknown*.

Deciding about whether a proposition is known or not is co-NP complete.

Theorem 1 *Given a conformant planning task $(A, \mathcal{I}, G)$, a state s corresponding to an action sequence $P \in A^*$, and a proposition p. Deciding whether p is known in s is co-NP complete.*

The planning task in the proof to Theorem 1 makes use of no syntactical constructs other than what we are actually using in the internal representation after preprocessing. Note also that the task makes no use of delete lists so the decision problem is hard even in the relaxed case.[3] With little changes to the proof one can show that deciding whether p is negatively known is co-NP complete, too, and that deciding whether p is unknown is NP-complete.

In our implementation, we compute the sets of known and negatively known propositions in a state by using a CNF corresponding to the semantics of the respective action sequence as follows (for now we assume that all action effects are deterministic; extension to non-determinism will be explained at the end of this subsection). We use a time index to differentiate between values of propositions at different points along the execution of the action sequence. The temporal index 0 stands for proposition values at the initial state. Say we have an action sequence $P = \langle a_1, \ldots, a_n \rangle$. We obtain our CNF $\phi(P)$ as follows. We initialize $\phi(P)$ as the constraints given by $\mathcal{I}$ (i.e., we insert a unit clause $p(0)$ for all $p \in \mathcal{I}$, a unit clause $\neg p(0)$ for all $\neg p \in \mathcal{I}$, and a clause $l_1(0) \vee \ldots \vee l_k(0)$ for all $l_1 \vee \ldots \vee l_k \in \mathcal{I}$). We then use a_1 to extend $\phi(P)$:

- *Effect Axioms*: for every effect e of a_1, $con(e) = \{p_1, \ldots, p_k\}$, and every proposition $p \in add(e)$, we insert the clause $\neg p_1(0) \vee \ldots \vee \neg p_k(0) \vee p(1)$; for every proposition $p \in del(e)$, we insert the clause $\neg p_1(0) \vee \ldots \vee \neg p_k(0) \vee \neg p(1)$.[4] Note that con is empty in the case of unconditional effects.
- *Frame Axioms*: for every proposition p, let $e_1, \ldots, e_n$ be the effects of a_1 such that $p \in del(e_i)$; for every tuple $p_1, \ldots, p_n$ such that $p_i \in con(e_i)$ we insert the clause $\neg p(0) \vee p_1(0) \vee \ldots \vee p_n(0) \vee p(1)$ (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 $e_1, \ldots, e_n$ are the effects of a_1 such that $p \in add(e_i)$, we insert for every tuple $p_1, \ldots, p_n$ with $p_i \in con(e_i)$ the clause $p(0) \vee p_1(0) \vee \ldots \vee p_n(0) \vee \neg p(1)$ (if p was false before and has not been added, it is still false after a_1).

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. The resulting CNF $\phi(P)$ captures the semantics of P in the following sense.

Proposition 1 *Given a conformant planning task $(A, \mathcal{I}, G)$, and an n-step action sequence $P \in A^*$. Say we have a possible initial state $I \in 2^{\mathcal{I}}$ and a proposition p. Then there is exactly one satisfying assingment σ to $\phi(P)_I$ ($\phi(P)$ where all variables at time 0 have been set to their values in I), and p holds upon execution of P in I iff $\sigma(p(n)) = TRUE$.*

Note that, in essence, $\phi(P)$ is a description of P's semantics in the situation calculus (McCarthy & Hayes 1969).

[2] Given the occuring add effects and delete effects of each action application are non-intersecting: if an action occurence both adds and deletes p, then the same will happen with $not\text{-}p$ and no matter whether one applies the adds or the deletes first both propositions will have the same truth value afterwards.

[3] The task in the proof *does* make use of non-unary effect antecedants (unary effect antecedants are a special case relevant for our current heuristic function implementation, see below). It is easy to come up with a reduction that proves the decision problem hard with unary effect antecedants in the presence of delete lists; whether the problem remains hard with unary effect antecedants *and* no delete lists is an open question.

[4] Due to our preprocessing, the conditions are all (positive) propositions; at this point (in difference to the heuristic function which computes relaxed plans) this is not important and we could as well deal with literals.

Proposition 1 immediately gives us the following corollary which we use to compute our sets of known facts.

Corollary 1 *Given a conformant planning task $(A, \mathcal{I}, G)$, a state s corresponding to an n-step action sequence $P \in A^*$, and a proposition p. Then p is known in s iff $\phi(P)$ implies $p(n)$.*

We use Corollary 1 to compute the set of known propositions as follows. For each proposition p, hand $\phi(P) \wedge \neg p(n)$ 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 propostions by handing the formulae $\phi(P) \wedge p(n)$ to the SAT solver.

At this point, let us consider a small illustrative example. Say we have a robot that is initially at one out of two locations, modeled as $\mathcal{I} = \{at\text{-}L_1 \vee at\text{-}L_2, \neg at\text{-}L_1 \vee \neg at\text{-}L_2\}$ (both propositions are unknown initially which is specified implicitly – no truth value for the propositions is given in $\mathcal{I}$). Our goal is to be at L_2, and we have a *move-right* action that has an empty precondition, and the conditional effect $(con = \{at\text{-}L_1\}, add = \{at\text{-}L_2\}, del = \{at\text{-}L_1\})$. The known propositions in the search state s corresponding to the sequence $P = \langle$*move-right*$\rangle$ are computed as follows. The formula $\phi(P)$ consists of the clauses $at\text{-}L_1(0) \vee at\text{-}L_2(0)$ and $\neg at\text{-}L_1(0) \vee \neg at\text{-}L_2(0)$ (initial disjunctions), $\neg at\text{-}L_1(0) \vee at\text{-}L_2(1)$ (add effect axiom for *move-right*) and $\neg at\text{-}L_1(0) \vee \neg at\text{-}L_1(1)$ (delete effect axiom for *move-right*), as well as $\neg at\text{-}L_1(0) \vee at\text{-}L_1(0) \vee at\text{-}L_1(1)$ (positive frame axiom for $at\text{-}L_1$; note that this can be skipped), $\neg at\text{-}L_2(0) \vee at\text{-}L_2(1)$ (positive frame axiom for $at\text{-}L_2$), $at\text{-}L_1(0) \vee \neg at\text{-}L_1(1)$ (negative frame axiom for $at\text{-}L_1$), and $at\text{-}L_2(0) \vee at\text{-}L_1(0) \vee \neg at\text{-}L_2(1)$ (negative frame axiom for $at\text{-}L_2$). To check whether $at\text{-}L_1$ is known in s, a satisfiability test is made on $\phi(P) \wedge \neg at\text{-}L_1(1)$. The result is "sat": a satisfying assignment σ is, e.g., that corresponding to $I = \{at\text{-}L_2\}$, i.e., $\sigma(at\text{-}L_2(0)) = TRUE$, $\sigma(at\text{-}L_1(0)) = FALSE$, $\sigma(at\text{-}L_2(1)) = TRUE$, $\sigma(at\text{-}L_1(1)) = FALSE$. Checking whether $at\text{-}L_2$ is known in s succeeds, however: $\phi(P) \wedge \neg at\text{-}L_2(1)$ is unsatisfiable. Inserting $\neg at\text{-}L_2(1)$ into the positive frame axiom for $at\text{-}L_2$ we get $\neg at\text{-}L_2(0)$, inserting $\neg at\text{-}L_2(1)$ into the effect axiom for *move-right* we get $\neg at\text{-}L_1(0)$, in consequence the initial disjunction clause $at\text{-}L_1(0) \vee at\text{-}L_2(0)$ becomes empty. Similarly, one can find out that $at\text{-}L_1$ is negatively known in s.

The observation made in Corollary 1 gets us around enumerating all possible initial states for computing whether a given proposition is known upon execution of P or not. While we *do* need to perform worst-case exponential reasoning about the formula $\phi(P)$, our empirical results show that this reasoning is feasible, as the interactions between action effects in practice (at least as reflected by our benchmark domains) 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 formulae looked at. In our current implementation, these are:

- Simplify $\phi(P)$ by inserting the values of propositions at times $i < n$ which are known to be true or false – these values are stored in the respective search states along the path corresponding to P. In effect, $\phi(P)$ only contains variables whose value is unknown at the respective points of P's execution.
- Make SAT calls only on propositions p that fulfill the following. First, p is not affected by an effect of a_n which is known to occur (namely, a_n's unconditional effects and its conditional effects whose condition is known to hold at the preceeding state). Second, p *might be inverted* by an effect of a_n which is *unknown*. The latter effects are those of a_n's effects such that all condition propositions are, at the preceeding state, either known to be true or unknown, and there is at least one condition that is unknown. Then, p might be inverted if it was unknown previously and is either added or deleted by an unknown effect, or if p was known to be true previously and is deleted by an unknown effect, or if p was known to be false previously and is added by an unknown effect.

Once the known propositions in s are computed, the actions applicable to s are those whose preconditions are all known in s, and whose effects can not be *self-contradictory* in s. Obviously, only actions fulfilling the first condition can be applied in s no matter what the real initial state is. To explain the second condition, it can be that an action a has *potentially* self-contradictory effects, i.e., effects e and e' such that $add(e) \cap del(e') \neq \emptyset$ or $add(e') \cap del(e) \neq \emptyset$. For all such pairs e and e' of potentially self-contradictory effects, we check whether they can occur *together* in s for a possible initial state. This check is made by a SAT call on the formula $\phi(P) \wedge \bigwedge_{c \in con(e) \cup con(e')} c(n)$. If the result is "sat" then e and e' can occur together and we skip the action. This conservative approach is necessary to ensure that our translation of negated propositions works correctly (if adds and deletes intersect then p and *not-p* will undergo the same updates, as also explained above in Section). Of course, one can avoid unnecessary SAT calls when the known and negatively known propositions in s already imply that a pair e and e' of effects will or will not occur together.

Relaxed Plans

Let us focus on the computation of the heuristic function. As said, this function is defined as the length of a relaxed plan, where the relaxation is to assume that all delete lists are empty (plus, in the conformant setting, a stronger form of reasoning about known propositions, as explained below). In the classical setting, relaxed plans are computed in the following Graphplan-style manner (Blum & Furst 1997; Hoffmann & Nebel 2001). Starting from the world state w, build a *relaxed planning graph* as a sequence of alternating proposition layers P_i and action layers A_i, where P_0 is the same as w, A_i is the set of all actions whose preconditions are contained in P_i, and P_{i+1} is P_i plus the add effects (with fulfilled conditions) of the actions in A_i. From a proposition layer P_m in which the goals are contained one can find a relaxed plan by a simple backchaining loop: select achieving actions at layers $i < m$ for all goals in P_m, insert those actions' preconditions and the respective effect conditions as new subgoals (which by construction are at layers below 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. Actions that achieve a sub-goal at P_1 –

actions that could be selected to *start* the relaxed plan – are called *helpful actions* and denoted by $H(w)$. As described earlier these are used by FF to prune (unpromising branches of) the search space. If there is no relaxed plan then the planning graph will reach a fixpoint $P_i = P_{i+1}$; $h(w)$ is then set to ∞, excluding the state from the search space – if there is no relaxed plan from w then there does not exist a real plan either.

In the conformant setting, we extend this machinery by sets pP_i of propositions that *possibly* hold at step i. Starting from the search state s, similar to before P_0 are the propositions that are known in s; pP_0 are the propositions that are unknown in s. To step from proposition layer i to layer $i+1$, we first proceed exactly as before, i.e., we set A_i to those actions whose preconditions are in P_i, then we set P_{i+1} to the union of P_i with the add effects (with fulfilled conditions) of the actions in A_i. Thereafter, we set pP_{i+1} to $pP_i \setminus P_{i+1}$ — the previously possibly true propositions minus those that now *are* true. Then new possibly true propositions are inserted: the added propositions of effects that *possibly occur*. These effects are those of actions in A_i such that their condition propositions are all either in P_i or in pP_i, and at least one condition is in pP_i. The added propositions of such effects, if they are not already in P_{i+1}, are inserted into pP_{i+1}. When this process is finished, for each proposition in pP_{i+1} a check is made whether it can be *inferred* from an *implication graph* which we maintain in parallel to the relaxed plan graph. If the check succeeds, the respective proposition is removed from pP_{i+1} and inserted into P_{i+1} instead (with a flag identifying it as an inferred proposition).

The implication graph we maintain captures a stronger form of the inference process that we do for computing state transitions. The idea is to look at the relaxed problem, i.e., to ignore delete lists, and to consider only 2 literals out of each clause of the CNF that captures the true state semantics: the implications induced by these literals are kept as edges in the graph, and the inference process is done as known from 2-CNF reasoning (Aspvall, Plass, & Tarjan 1979). In more detail, the implication graph is built and used as follows. Let P be the action sequence leading to the state s, n be the length of P. Then the implication graph contains literals with a temporal index t ranging from $-n$ to m where m is the current top layer of the relaxed plan graph. Time values $t \geq 0$ correspond to the respective plan graph layer, values $t < 0$ correspond to the states along the execution of P ($t = -n$ thus corresponds to the initial state). The implication graph is initialized by inserting edges for the initially valid disjunctions, and for the add effect axioms of the actions in P (the latter edges are needed in order to take into account what might already have been achieved on the way to s). First, for all $t \leq 0$ a node $p(t)$ is inserted for all propositions which are unknown in the respective state; for $t = -n$, nodes $\neg p(t)$ are also created (while everywhere else all nodes in the implications are positive, initial disjunctions might yield negative nodes, see below). For propositions p that are unknown both at t and $t+1$ (for $t < 0$) an edge $(p(t), p(t+1))$ is inserted: edges (l, l') are to be read as "l implies l'", and $(p(t), p(t+1))$ is the positive frame axiom for p (as there are no delete lists in the relaxation a positive value of p will be preserved). Then, for each initial disjunction $l_1 \vee \ldots \vee l_k$ the edges $(\overline{l_1}(-n), l_2(-n))$ and $(\overline{l_2}(-n), l_1(-n))$ are inserted — i.e., we arbitrarily pick two literals in the disjunction and insert the respective implication (and its contraposition), which is stronger than the original disjunction. For each action in P, executed at time point t, that has an unknown effect with conditions $p_1, \ldots, p_k$ (all of which are either known or unknown in the state at t, at least one being unknown), for all added propositions p of the effect (where p is unknown at $t+1$) the edge $(p_j(t), p(t+1))$ is inserted where j is the lowest index such that p_j is unknown at t. Note that picking one single arbitrary p_j for the implication results in a stronger form of reasoning as really *all* conditions are needed to imply $p(t+1)$. We make this simplification for computational efficiency (some more on this in the outlook). When all the edges capturing initial disjunctions and P effects have been inserted, relaxed plan graph construction starts. Say the construction steps from layer i to layer $i+1$. Reconsider the process described above, computing the possibly true propositions pP_{i+1}. First, $pP_i \setminus P_{i+1}$ is inserted. For all these propositions p an implication graph node $p(i+1)$ is created, as well as the frame axiom edge $(p(i), p(i+1))$. Then, we consider the effects that possibly occur, see above. If such an effect with conditions $p_1, \ldots, p_k$ adds a proposition $p \notin P_{i+1}$ then a node $p(i+1)$ is created (unless that node is already there), and the effect edge $(p_j(i), p(i+1))$ is inserted (where, similar to before, j is the lowest index such that p_j is in pP_i). When all possibly occuring effects have been processed, as said above the graph is used to perform inference checks which might move propositions out of pP_{i+1} into P_{i+1}. The check for a proposition p proceeds by seeing whether there are (backwards) paths in the implication graph from $p(i+1)$ to a proposition and its negation (i.e., $p'(i')$ and $\neg p'(i'')$ for $i', i'' < i$), or to the negation of p itself (i.e., $\neg p(i')$ for $i' < i$). If that is the case, the check succeeds, otherwise it fails. This sort of reasoning is a quite standard technique for reasoning with the implication graph of a 2-CNF formula (Aspvall, Plass, & Tarjan 1979). During relaxed plan extraction, if a sub-goal is encountered where the implication graph check succeeded and that has thus been inferred, then we consider the paths in the implication graph that were responsible for this inference: the actions constituting the add effect edges on this paths are selected into the relaxed plan. If these actions are executable in s they are also put into the set of helpful actions $H(s)$.

Note that the implication graph we use is a complete, but not necessarily sound, approximation of the set of known propositions. Therefore, if no plan exists in the modified relaxed planning graph (if relaxed plan graph construction fails to reach the goals), no plan exists in practice. On the other hand, as the implication graph might infer propositions that can't be inferred, it can happen that the relaxed plan is empty but no goal state is reached yet. As FF terminates when $h(s) = 0$, we thus set $h(s)$ to 0 (only) if all goal propositions are known in s; otherwise we set $h(s)$ to the number of actions in the relaxed plan plus 1.

To illustrate the relaxed planning process, let us consider the example we also used above in Section . We have a robot that is initially at one out of two locations, $\mathcal{I} = \{at\text{-}L_1 \vee at\text{-}L_2, \neg at\text{-}L_1 \vee \neg at\text{-}L_2\}$. The goal is to be at L_2, and we have a *move-right* action that has an empty precondition, and the conditional effect

$(\{at\text{-}L_1\}, \{at\text{-}L_2\}, \{at\text{-}L_1\})$. When we build the relaxed plan graph to the initial state (i.e., to the search state corresponding to the empty action sequence), we first initialize the implication graph by inserting the nodes $at\text{-}L_1(0)$, $\neg at\text{-}L_1(0)$, $at\text{-}L_2(0)$, and $\neg at\text{-}L_2(0)$; we also insert the edges $(\neg at\text{-}L_1(0), at\text{-}L_2(0))$ and $(\neg at\text{-}L_2(0), at\text{-}L_1(0))$ (for the initial disjunction $at\text{-}L_1 \vee at\text{-}L_2$), as well as $(at\text{-}L_1(0), \neg at\text{-}L_2(0))$ and $(at\text{-}L_2(0), \neg at\text{-}L_1(0))$ (for the initial disjunction $\neg at\text{-}L_1 \vee \neg at\text{-}L_2$). Starting the graph building procedure, we then get $P_0 = \emptyset$, $pP_0 = \{at\text{-}L_1, at\text{-}L_2\}$, and $A_0 = \{$*move-right*$\}$. Thereafter we initialize $P_1 = \emptyset$ and $pP_1 = \{at\text{-}L_1, at\text{-}L_2\}$, inserting the implication graph nodes $at\text{-}L_1(1)$ and $at\text{-}L_2(1)$ as well as the frame axiom edges $(at\text{-}L_1(0), at\text{-}L_1(1))$ and $(at\text{-}L_2(0), at\text{-}L_2(1))$. We then look at all effects of the actions in A_0, and find that the effect of *move-right* possibly occurs – its single condition $at\text{-}L_1$ is in pP_0. Thus, we insert the add effect edge $(at\text{-}L_1(0), at\text{-}L_2(1))$. This finishes the loop over the A_0 effects, and we proceed to checking whether the propositions in $pP(1)$ can be inferred from the implication graph. For $at\text{-}L_1$ this check fails: the only implication graph nodes reachable by following edges backwards from $at\text{-}L_1(1)$ are $at\text{-}L_1(0)$ and $\neg at\text{-}L_2(0)$. For $at\text{-}L_2$ however the check succeeds: we can reach $\neg at\text{-}L_2(0)$ via the add effect edge $(at\text{-}L_1(0), at\text{-}L_2(1))$ and the initial disjunction edge $(\neg at\text{-}L_2(0), at\text{-}L_1(0))$. In effect, $at\text{-}L_2$ is moved from pP_1 into P_1, yielding $P_1 = \{at\text{-}L_2\}$ and $pP_1 = \{at\text{-}L_1\}$. The goals are reached. Relaxed plan extraction selects the actions constituting the path that was responsible for the inference of $at\text{-}L_2(1)$; this is the single action *move-right* which forms the extracted relaxed plan to the initial state.

Repeated States

As the final one of FF's algorithmic techniques that we need to extend to the conformant setting, let us now consider how we can avoid exploring the same, or more generally *dominated* (see below), search states over again. In the classical setting, we say that world state w *dominates* world state w' if the set of true propositions in w contains the set of true propositions in w'. This simple containment test is cheap to perform. When classical FF performs search, it checks via a simple hash table lookup whether the new state is dominated by some previous state (in the same hash entry). If so, the new state is pruned.

In the conformant setting, we use the following definition. Let s and s' be two search states reached via the action sequences P and P', respectively. We say that s *dominates* s' if for every possible initial state $I \in 2^{\mathcal{I}}$ (and every outcome of the non-deterministic effects, if such effects are present), the world state upon execution of P in I dominates, in the classical sense, the world state upon execution of P' in I (for every outcome of the non-deterministic effects). We can prune a search state s' if s is an earlier seen search state that dominates s' (no matter what the initial state is, P will lead to a world state at least as good as the one where P' leads to). The domination relation between s and s' can be tested by checking for each proposition p whether there exists a possible initial state given which p does not hold in s, but does hold in s'. One such check requires a single satisfiability test using the formulae we already generated for s and s'. Let $\phi(P)$ and $\phi(P')$ be the formulae constructed for the plans P and P', where $\phi(P)$ and $\phi(P')$ share the same propositional variables for time 0 propositions but have different propositional variables for all times > 0 (i.e., these formulae capture the semantics of executing P and P' in the *same* initial world state). Let $p(P)$ and $p(P')$ denote p's value (the respective propositional variable) following P and P', respectively. If $\phi(P) \wedge \phi(P') \wedge \neg p(P) \wedge p(P')$ is satisfiable, we know that there is an initial state (and non-deterministic effects outcome) from which P leads into a state where p does not hold but P' leads into a state where p does hold. In that case, s does not dominate s'. If $\phi(P) \wedge \phi(P') \wedge \neg p(P) \wedge p(P')$ is unsatisfiable for all propositions p, then s dominates s' and s' can be pruned.

In our implementation we use the following variation of the above technique, avoiding some unnecessary computations. Like in the classical case we keep the seen search states in a hash table, where the hash key is a function of the union of the known and the unknown propositions in a state. Any new state s' is then compared to the states s in the same hash entry. The union of known and unknown propositions is, by virtue of the hash key function, the same in both s and s'. If a proposition is not known in s that is known in s' then the comparison fails. Otherwise, the above described satisfiability test is invoked for all propositions that are unknown in both s and s'; if the result is "unsat" for all these propositions, then s' is pruned.

Results

We performed experimental evaluation on a variety of domains. These include the traditional conformant benchmark domains Bomb-in-the-toilet, Ring, Cube, Omlette, and Safe; see (Cimatti & Roveri 2000) and (Petrick & Bacchus 2002) for a detailed description of these domains. We also performed tests on variants of the classical benchmarks Blocksworld, Logistics, and Grid, into which we introduced uncertainty about the initial state. In the Blocksworld, the stacking order of a number of the top blocks on the stacks was unknown; in the Logistics domain the initial locations of packages within a city were unknown; and in the Grid domain, the shapes of some of the locks were unknown. All testing examples are available for download from http://www.informatik.uni-freiburg.de/∼hoffmann/cff-tests.tgz. Conformant-FF is written in C and the experiments were conducted on a PC with a Pentium 4 2.4 GHz processor with 0.5 Gigabyte memory and 512KB cache running linux. The SAT procedure used within the tested implementation is a very naive standard DPLL solver (Davis & Putnam 1960), performing unit propagation by an iterated naive loop over all clauses: look at all literals in all clauses, if a clause is unit then set the value of the respective literal, continue until an empty clause (all literals set to FALSE) is reached or no more changes occur. We also integrated Chaff (Moskewicz *et al.* 2001) into our code, but decided not to use it as, typically, communicating all the CNFs (thousands, in many examples) to Chaff was *much* more costly – in fact prohibitive for performance – than it was to solve these (typically, a matter of split seconds). So we decided to demonstrate the potential of our approach by running it with an internal, naive, DPLL implementation instead. An obvious option to further improve

Conformant-FF's performance is to implement more sophisticated SAT procedures. In particular improving on the implementation of unit propagation seems worthwhile: we observed that often few iterations of unit propagation were sufficient to complete a SAT call; in Bomb-in-the-toilet, for example, *all* clauses in our formulae are binary. Judging from the success of sophisticated unit propagation procedures in the field of SAT, and considering that usually around 75% of the runtime Conformant-FF requires are spent in that procedure, dramatic runtime improvements are to be expected from better implementations.

In our experiments we compared the performance of Conformant-FF and MBP. Although there is a variety of conformant planners, using diverse techniques, according to (Bertoli & Cimatti 2002) MBP is the best current conformant planner, outperforming all other conformant planners on the traditional domains. We used the newest version of MBP that was downloadable from the web page of MBP's authors. Conformant-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 (namely, initially unknown facts as well as initially valid disjunctions of literals based on such facts). Conformant-FF then generates a set of ground instances from this domain performing some optimizations, such as recognizing static variables and doing an approximate reachability analysis to prune unreachable actions. MBP is given as input an NPDDL file, which is an extension of PDDL allowing for uncertainty, non-deterministic effects, and a rich set of goal conditions. MBP translates this input into a set of ground action instances, as well. Its translation process is naive, and therefore, we made a serious effort to use various NPDDL options that reduce this set of actions. In particular, NPDDL allows for the definition of functions (which allow efficient encoding of multi-valued variables) – a capability that Conformant-FF does not have – and we tried to use this option as much as we could. In each domain, we tested a variety of example tasks, giving the planners at most 25 minutes to solve each task. In Table 1 we provide the results for the Bomb-in-the-toilet, Cube, Omlette, Ring, and Safe domains.

In the well-known Bomb-in-the-toilet domain, the observation to be made is that MBP is competitive with Conformant-FF when there are very few toilets, but gets outperformed quickly as the number of toilets increases. The data in Table 1 are to be understood as follows. In Bomb-in-the-toilet we used four test suits, each of which contains five example instances. In the "Bomb-x-x" suit instances the number of toilets is the same as the number of bombs. In the "Bomb-x-c" suits the number of bombs varies, but the number of toilets is fixed to c. Across all these test suits, example 1 contains 5 bombs, example 2 contains 10 bombs, example 3 contains 20 bombs, example 4 contains 50 bombs, and example 5 contains 100 bombs. For each test suit, the upper row in the table provides Conformant-FF's runtimes whereas the lower row provides MBP's runtimes. The top right corner of Table 1 thus says that Conformant-FF solves the task with 100 bombs and 100 toilets in no more than 2.39 seconds. Indeed, we believe that one modest contribution of our work might be to lay this annoying domain to rest.

In Cube, one is initially located at any point on a 3-dimensional grid with extension $n \times n \times n$. In each dimension one can move up or down; when moving against a border of the Grid, nothing happens. In the "Cube-corner" test suit the task is to move into a corner of the grid (which can be done by moving, in each dimension, n times into the direction of that corner). In the "Cube-center" test suit the task is to move into the center of the grid (which can be done by moving into a corner to locate oneself, then going to the center). In both test suits, the value of n is 3, 5, 7, 9, and 11 for example 1, 2, 3, 4, and 5, respectively. Conformant-FF is clearly better suited than MBP to find a corner.[5] None of the planners scales very well to the move into the center; MBP is somewhat better at that.

Test suit	exp1	exp2	exp3	exp4	exp5
Bomb-x-x	0.00	0.00	0.00	0.24	2.39
Bomb-x-x	0.08	1.82	50.54	-	-
Bomb-x-1	0.00	0.00	0.07	4.69	113.70
Bomb-x-1	0.01	0.02	0.09	1.18	11.35
Bomb-x-5	0.00	0.01	0.10	4.68	113.24
Bomb-x-5	0.08	0.39	2.67	35.24	327.92
Bomb-x-10	0.00	0.00	0.04	3.32	97.50
Bomb-x-10	0.39	1.82	10.20	130.90	1279.07
Cube-corner	0.00	0.06	0.48	1.80	5.66
Cube-corner	0.04	2.94	-	-	-
Cube-center	0.19	-	-	-	-
Cube-center	2.82	2.74	-	-	-
Omlette	0.02	0.03	0.30	1.14	3.74
Omlette	1.33	13.32	556.66	-	-
Ring	0.02	1.35	50.61	-	-
Ring	0.00	0.01	0.02	0.03	0.07
Safe	0.00	0.05	151.29	788.93	907.38
Safe	0.00	0.06	144.45	573.98	1355.86

Table 1: Conformant-FF runtime (upper rows) vs. MBP runtime (lower rows) in our purely conformant testing suits. Times are in seconds, dashes indicate time-outs. See test suit explanation in the text.

In the Omlette domain, n eggs must be broken into a bowl without spoiling the bowl. Breaking an egg into the bowl spoils the bowl by a non-deterministic effect (namely, when the egg is bad, which one does not know without breaking it). As said, we haven't yet implemented support for such effects so we have modeled that effect as a conditional effect that has a new, unknown, "dummy" proposition as its condition. There is no conformant plan to the Omlette tasks. In our test suit, the value of n is 3, 5, 10, 15, and 20 for the five examples respectively. In difference to MBP, Conformant-FF proves these tasks unsolvable quite efficiently (by exhausting the space of reachable search states).

The Ring domain is the only domain that we found really problematic for Conformant-FF. There are n rooms through which one can move in a cyclic fashion. The initial location is unkown. Each room has a window which is either

[5] In both planners, there were performance differences depending on *which* corner the planner was supposed to move into. For both planners we chose the corner that yielded the best performance.

open or closed or locked. The initial states of the windows are unknown. One can apply an action “close” which closes the window of the room in which one is currently located, and one can apply an action “lock”which locks the window of the room in which one is currently located, *given the window is already closed* (note that this is a binary effect condition: current location, and closed window). A solution plan moves once through the ring and applies the “close” and “lock” actions after each move. Our test suit contains the tasks with n value 2, 3, 4, 5, and 6, respectively. Judging from what we observed in our experiments, Conformant-FF’s lack of performance is due to two reasons. First, the heuristic function is not very informative as it takes into account only one of the two conditions of the effects of “lock”. Second, until up to the point where a plan has been found not a single proposition becomes known, and thus SAT calls for repeated states checking are made for *every* pair of new and seen states – all states are in the same hash table entry, and 90% of the runtime is spent in SAT calls for checking state domination. Indeed, when turning the repeated states check off, Conformant-FF becomes more efficient and solves the five examples in 0.00, 0.29, 4.76, 50.76, and 445.50 seconds respectively. We conclude that, in Ring, repeated states checking causes more overhead than pruning benefits. On the other hand, even without the repeated states check Conformant-FF does not scale well. We attribute this to the first weakness mentioned above, lack of heuristic quality. Ring is the only one of our testing domains where there are non-unary unknown effect conditions. We consider the lack of heuristic quality in the presence of such conditions as the most important weakness of our current Conformant-FF implementation. In the discussion, Section , we outline algorithmic methods that have the potential to overcome this weakness. We also outline enhancements to Conformant-FF that have the potential to avoid the overhead that can be caused by repeated states checking.

In the Safe domain, a safe has one out of n possible combinations, which one must all try in order to open the safe. Our test suit contains the examples with n values 5, 10, 30, 50, and 70, respectively. As can be seen, Conformant-FF and MBP scale roughly similar here. We remark at this point that, similar to what we observed in Ring, all search states are in the same hash entry so that most of Conformant-FF’s runtime (80%, roughly) is spent in SAT calls for checking state domination. Turning the repeated states check off, the runtimes we get are 0.01, 0.01, 0.61, 22.73, and 156.27 seconds for our five examples, respectively.

Overall, we see that in the traditional conformant benchmarks Conformant-FF is very competitive, being weak only in Ring and Cube-center while outperforming MBP in Bomb-in-the-toilet (when there are more than 5 toilets), Omlette, and Cube-corner.

In Table 2 we provide the results for our “mixed” benchmark domains, enriching classical planning benchmarks with uncertainty.

The Blocksworld domain we use is the variant with three operators to put a block x from another block y onto the table, to put a block x from a block y onto a different block z, and to put a block x from the table onto a block y. As said, the uncertainty in our test suits is that the top blocks on each initial stack are arranged in an unknown order. In the test

Test suit	exp1	exp2	exp3	exp4	exp5
Blocksworld-2	0.00	0.00	0.00	1.08	3.53
Blocksworld-2	6.17	13.18	102.50	-	-
Blocksworld-3	0.00	0.00	0.02	1.64	210.84
Blocksworld-3	0.62	13.32	251.34	-	-
Blocksworld-4	0.03	0.07	0.14	6.53	93.97
Blocksworld-4	0.62	13.46	240.63	-	-
Logistics-2	0.00	0.01	0.01	0.02	3.33
Logistics-2	1.63	16.36	35.54	-	-
Logistics-3	0.00	0.02	0.03	0.06	41.80
Logistics-3	23.13	508.24	428.94	-	-
Logistics-4	0.01	0.12	0.01	0.07	71.20
Logistics-4	122.66	1091.85	1187.00	-	-
Grid-2	0.03	0.17	1.46	3.13	80.87
Grid-2	-	-	-	-	-
Grid-3	0.05	0.95	1.39	6.81	136.79
Grid-3	-	-	-	-	-
Grid-4	0.05	2.54	5.78	12.53	512.30
Grid-4	-	-	-	-	-

Table 2: Conformant-FF runtime (upper rows) vs. MBP runtime (lower rows) in our mixed classical and conformant testing suits. Times are in seconds, dashes indicate timeouts. See test suit explanation in the text.

suits “Blocksworld-k” the order of the top k blocks on each stack (which might be the whole stack) is unknown (the unknown arrangement is accomplished by specifying rather a lot of initial disjunctions for the respective “on”, “on-table”, and “clear” relations). Putting a block x from a block y onto the table has conditional effects: (only) if x is located on y in the state of execution, x is put onto the table, and y is cleared. That is, (only) if x is on y then x, including the whole stack of blocks on top of it, is moved to the table. To make sure that a block x out of an unknown stack is on the table, one must thus try to put x to the table from all other blocks y in the stack – if n blocks are in the unknown stack then $n * (n - 1)$ actions are needed to be sure where all the blocks are. Across our three test suits, the examples are generated with the software by Slaney and Thiebaux (Slaney & Thiebaux 2001), and contain 5, 6, 7, 13, and 20 blocks respectively. As the behavior of Conformant-FF differs quite considerably on individual Blocksworld instances, we generated 25 random examples per size, and provide average runtime values. Conformant-FF scales well and outperforms MBP dramatically (though we improved the performance of MBP by using a more concise domain description based on functions). MBP did not solve any instance with 13 or 20 blocks. We note that in some of these cases, MBP stopped before our self-imposed time-limit because it reached its maximal number of search states.

Our Logistics domain is the following modification of the well-known standard encoding. The uncertainty we introduced lies in that the initial position of each package *within its origin city* is unknown. Loading a package onto a truck has a conditional effect that only occurs when the package is at the same location as the truck. One must thus try the loading action at all locations within a package’s origin city in order to be sure the package is loaded. The amount of uncertainty increases with the size of the cities. In our test suits "Logistics-k" the city size (which is the same for each city) is k. Across all suits, the instances are randomly generated ones with the following parameters. Each city contains one truck. Example 1 has 2 cities, 2 packages, and 1 airplane; example 2 has 2 cities, 4 packages, and 1 airplane; example 3 has 3 cities, 2 packages, and 1 airplane; example 4 has 3 cities, 3 packages, and 2 airplanes; example 5 has 10 cities, 10 packages, and 10 airplanes. While MBP consumes a lot of time to solve even the smaller instances, Conformant-FF comfortably scales up to the largest instances (which are quite challenging: the plan found for the largest "Logistics-4" example, e.g., has 120 actions in it).

Our final testing domain is a variant of the Grid domain as used in the AIPS-98 planning competition. In Grid, a robot moves on a 2-dimensional grid on which positions can be locked and, to be accessible, must be opened with a key of a matching shape; the robot can hold one key at a time, and the goal is to transport certain keys to certain positions. We introduced uncertainty about (some of the, see below) locked positions on the grid. The shape of these locks was specified to be an unknown one out of a number of possible shapes. Opening a lock with a key has a conditional effect that occurs only if the key is of the same shape as the lock. One must thus try all possible keys in order to make sure that a lock is opened. In our test suits "Grid-k" the unknown locks have k possible shapes. Across the suits, the instances are the original five examples "prob01" … "prob05" as used in the AIPS-98 competition. The locks for which we have introduced uncertainty are those that *must be opened in order to solve the respective task*. In examples 1, 3, and 4 this is a single lock, in examples 2 and 5 it are two locks. While MBP fails to solve even the modified "prob01" tasks, Conformant-FF scales up to "prob05".[6] Note that these instances are hard to solve even without uncertainty. In fact, FF is the only domain-independent planner the authors are aware of which has been reported to solve all the original AIPS-98 instances. With uncertainty, the tasks become considerably more difficult, as several keys instead of one key must be transported to the locked grid position(s) (remember that the robot can transport only one key at a time). In fact, while FF finds a 174-steps plan for "prob05" without uncertainty, the plan Conformant-FF finds for "prob05" with $k = 2$ has 194 steps, the one found with $k = 3$ has 214 steps, and the one found with $k = 4$ is 254 steps long.

We also conducted a number of experiments with GPT on a sample of instances from the mixed domains in order to ensure that we were indeed comparing Conformant-FF with the best conformant planner for these domains. Indeed, in these domains, MBP outperforms GPT.

From our results in the "mixed" domains, we conclude that our approach *does* have the potential to combine the strengths of FF with conformant abilities in domains that combine classical and conformant aspects. In this respect, Conformant-FF is superior to MBP (and, presumably, to GPT and other conformant planners). We consider this an important advantage as, in a real-world domain, one would certainly expect only *parts* of the problem to be uncertain. This, we find, is reflected quite nicely in, e.g., our modified Grid encoding.

[6]We have also run Conformant-FF on versions of "prob01" … "prob05" where the shapes of *all* locks (8, 8, 10, 9, and 19 respectively) were unknown. With that, Conformant-FF still scaled up to "prob04" in our experiments – it solved that task with $k = 4$ in 44.52 seconds.

Related Work

Conformant-FF shares various ideas and techniques with other conformant planning algorithms. CGP (Smith & Weld 1998) was the first specialized conformant planner, based on the Graphplan algorithm, which Conformant-FF uses to generate its heuristic function. However, while Conformant-FF uses a single planning graph, CGP used one planning graph for each possible initial state, an approach that does not scale up well. GPT (Bonet & Geffner 2000) is a general platform for planning with uncertainty, including, but not limited to, conformant planning. GPT introduced the idea of planning in belief space using heuristic search. Conformant-FF is based on this idea, though it represents the set of possible worlds in a different manner, and it computes its heuristic estimate in a very different manner. Petrick and Bacchus (Petrick & Bacchus 2002) describe a contingent planner that uses logical formulas to represent the current state of *knowledge* of the agent. Conformant-FF adopted this idea, but using a much simpler representation that is adequate for the simpler, conformant planning problem. The major difference between the two approaches is that Petrick and Bacchus *model* planning tasks at the knowledge level (which loses expressive power (Petrick & Bacchus 2002)) while we *infer* our knowledge from a standard task description.

The use of SAT solvers plays an important part in QbfPlan (Rintanen 1999) and Cplan (Castellini, Giunchiglia, & Tacchella 2001), which are conformant planners based on the planning as satisfiability approach. QbfPlan reduces this problem to the problem of satisfaction of a quantified boolean formula, whereas Cplan generates candidate solutions to the conformant planning problem and tests their adequacy via a reduction to SAT. However, aside for the use of SAT (or QBNF) solvers, these approaches differ considerably from Conformant-FF.

Finally, the approach to conformant planning that has been most successful so far, employed in MBP (Bertoli & Cimatti 2002), is based on heuristic forward search in belief space, where states in belief-space are represented using BDDs. It shares Conformant-FF's use of forward search in belief-space, but differs considerably in the representation of the search space and in the computation of the heuristic function. It should be noted that MBP is a more general planning system, supporting a rich set of goal conditions and run-time sensing capabilities.

Discussion

We described Conformant-FF, a heuristic forward-search planner, which extends the classical planner FF (Hoffmann & Nebel 2001) into conformant planning. Conformant-FF uses the same architecture as FF, but instead of representing the current world state, it computes and represents the current state of knowledge. The relaxed planning problem FF uses is modified accordingly using the idea of an implication graph which supports linear time reasoning about an approximate description of the knowledge state. The resulting planner is competitive with the state-of-the-art conformant planner MBP in the traditional conformant benchmark domains. Moreover, the planner shows the potential to combine the strength of FF with conformant abilities, by scaling well (in particular, much better than MBP) in a number of classical benchmark domains enriched with uncertainty. Our results are obtained using a *completely naive* SAT solver as the underlying reasoning technique.

As we have seen in the example of the Ring domain, c.f. Section , major inefficiencies in our current code can arise when 1. there are non-unary effect conditions (all but one of which are ignored by the heuristic function) and / or when 2. there is a lack of different known propositions in the search states, and thus repeated states checking has to test too many pairs of new and seen search states for domination. The second phenomenon appears to be a pathological property of certain conformant benchmarks like the Ring or the Safe domain. Even so, a strategy that might be able to overcome the difficulty is to introduce incomplete pre-tests to avoid full domination tests of state pairs s and s'. For example, say ϕ is the formula whose satisfiability must (repeatedly) be tested for deciding whether s dominates s', c.f. Section . If even a 2-projection of ϕ is solvable, then ϕ is solvable, and s does not dominate s'. As another option one can test conditions such as "does the path to s' contain an action that the path to s does not contain?" or "does the implication graph corresponding to s' contain an edge that the implication graph to s does not contain?", and, if one test succeeds, assume that s does not dominate s'. Note that the latter option might lose pruning power as the tested conditions are not sufficient for non-domination. Exploring these options is a topic for future work.

As for the first inefficiency mentioned above, possibly arising when there are non-unary effect conditions, we remark the following. First, the inefficiency only arises when more than one of the effect conditions in question is unknown in the state of the respective action's execution – otherwise the single unknown condition will be adequately dealt with by the implication graph. Our second remark is that one might be able to better adjust the heuristic function to non-unary effect conditions. One option is to intoduce a special case treatment for these, similar to our treatment of non-binary initial disjunctions: an additional possibility for inferring a proposition p could be to find implication graph paths to an effect condition $p_1 \wedge \ldots \wedge p_n$ as well as to all of $\neg p_1, \ldots, \neg p_n$. Another option is, even, to introduce SAT reasoning into the heuristic computation: in our experiments the single SAT calls were always very cheap, and the number of expanded search states rather low.[7] So it is worth trying to infer propositions in relaxed plan graph layers by doing the full inference process (on the relaxed problem) when non-unary effect conditions are present. While this is likely to slow down heuristic computation it is as likely to result in a very informative heuristic function.

[7]In the Ring instance with 3 rooms, e.g., with repeated states checking only 36 heuristic computations are performed to solve the task; with 4 rooms, it are 66.

Acknowledgement: We are grateful to the MBP and GPT teams who were always happy to answer our annoying questions and fix our faulty domain descriptions.

References

Aspvall, B.; Plass, M.; and Tarjan, R. 1979. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters* 8:121–123.

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

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *AIJ* 90(1-2):279–298.

Bonet, B., and Geffner, H. 1998. HSP: Heuristic search planner. In *AIPS'98 Planning Competition*.

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

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1–2):5–33.

Castellini, C.; Giunchiglia, E.; and Tacchella, A. 2001. Improvements to sat-based conformant planning. In *Proc. ECP'01*, 241–252.

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

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *J. ACM* 7(3):201–215.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *AIJ* 2:189–208.

Gazen, B. C., and Knoblock, C. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. ECP'97*, 221–233.

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

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 4. Edinburgh University Press. 463–502.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proc. 38th Design Automation Conf. (DAC'01)*.

Pednault, E. P. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR'89*, 324–331.

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.

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. *JAIR* 10:323–352.

Slaney, J., and Thiebaux, S. 2001. Blocks world revisited. *AIJ* 125:119–153.

Smith, D. E., and Weld, D. 1998. Conformant graphplan. In *Proc. AAAI'98*, 889–896.