Conformant Planning via Heuristic Forward Search: A New Approach

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 world we start from. It is well known that conformant planning can be transformed into a search problem in belief space, the space whose elements are sets of possible worlds. We introduce a new representation of that search space, replacing the need to store sets of possible worlds with a need to reason about the effects of action sequences. The reasoning is done by deciding solvability of CNFs that capture the action sequence's semantics. Based on this approach, we extend the classical heuristic planning system FF to the conformant setting. The key to this extension is the introduction of approximative CNF reasoning in FF's heuristic function. Our experimental evaluation shows Conformant-FF to be superior to the state-of-the-art conformant planners MBP, KACMBP, and GPT 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 (achieve all goal propositions) regardless of which particular initial world we start from and which action effects occur.

Conformant planning can be transformed into a search problem in the space of belief states, i.e., the space whose elements are sets of possible world states. This way, our uncertainty about the true current world state is modeled via the set of world states that we consider possible at this time. (Throughout the paper, we stick to this distinction between *world* states and *belief* states.) Bonet and Geffner (2000) introduced the idea of planning in belief space using heuristic forward search. But the number of possible worlds in a belief state is typically large – even in simple examples like the infamous Bomb-in-the-toilet problem, there are exponentially many worlds in the initial belief state – and so Bonet and Geffner's system GPT, which explicitly represents these states, usually fails to scale up. In their MBP system, Bertoli and Cimatti (2000; 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 prohibitive for scaling in many cases.

In this paper we suggest a third, lazy, approach to represent belief states. We observe that, for conformant planning, it suffices to know the propositions that are true in the *intersection* of the worlds contained in a belief state – only these propositions will be true no matter what initial world we start from (it should be noted that GPT and MBP are capable of doing much more than conformant planning). An action sequence is a conformant plan iff it leads to a belief state where the intersection of the worlds contains all goal propositions. In our framework, a belief state s is represented just by the initial belief state representation together with the action sequence P that leads to s. To check for fulfillment of the goal and of action preconditions, we then test, for each proposition p, if it is contained in the intersection of the worlds in s, i.e., if it is always true after executing P. We say that such propositions are *known* in s. Doing the test for a proposition is hard in general; we show the corresponding decision problem to be co-NP complete. Our implementation performs the reasoning by deciding solvability of a CNF formula that captures the semantics of the action sequence P.

In comparison to the belief state representations used in GPT and MBP, our approach trades space for time: assume the planner faces a belief state s and an applicable action a, and wants to know what the outcome s' of applying a to s would be. If, like in GPT and MBP, the planner maintains a complete knowledge of s in memory then the computation can be done based exclusively on that knowledge. In our approach, however, the planner has no knowledge about s in memory other than the known propositions (and the path to s). To determine the known propositions for s', the planner has to reason about the entire action sequence leading to s', *including those actions that lead to* s. Our intuition is that modern SAT solvers, which can easily handle problem instances 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.

While our representation of belief states is relatively efficient, a blind traversal of the search space is still infeasible due to the combinatorial explosion in possible action

sequences. In classical planning (with no uncertainty about the initial state or action effects), this problem has been tackled most successfully by using heuristic functions based on a relaxation of the planning task, where the relaxation is to assume that all delete lists are empty. To each search state during, e.g., a forward search, a relaxed plan is computed. The length of the relaxed plan provides an, often very informative, estimate of the state's goal distance. FF (Hoffmann & Nebel 2001) is one particularly successful system that uses this technique. Beside the search space representation, the main contribution of our work is the adaptation of FF's heuristic function to the conformant setting. We enrich FF's relaxed planning process by an efficient reasoning technique that uses a stronger version of the CNF formula that captures the true semantics of the action sequence leading to the search (belief) state. The stronger formula we use is a 2-CNF projection of the original formula. The implication graph (Aspvall, Plass, & Tarjan 1979) of this 2-CNF formula supports linear time reasoning, and is naturally embedded into the relaxed planning process used by FF to compute its heuristic function.

Our implementation, which we call Conformant-FF, is based on the FF code. Except for the necessary changes to deal with uncertainty, Conformant-FF is identical to FF. Our experimental results in a number of traditional conformant benchmarks show that 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. And while on traditional domains Conformant-FF is typically not as fast as KACMBP, a more recent version of MBP geared specifically to conformant planning (Bertoli & Cimatti 2002), on traditional domains, our approach has the potential to *combine* the strengths of FF with conformant abilities in domains that combine classical and conformant aspects. In fact, in a number of classical planning benchmarks enriched with uncertainty, Conformant-FF shows fine scalability, dramatically outperforming MBP as well as KACMBP (Bertoli & Cimatti 2002) and GPT (Bonet & Geffner 2000).

The paper is organized as follows. Section 2 briefly describes the planning framework we consider. Section 3 describes the search space representation and computation of known propositions. Section 4 explains our extension of FF's heuristic function, Section 5 explains how we avoid repeated search states. Section 6 gives some implementation details and our empirical results, Section 7 concludes the paper with a discussion of related and future work.

In a longer technical report we provide a more detailed description of Conformant-FF, explaining various optimizations, and how Conformant-FF can be extended to deal with non-deterministic effects. Here, we only deal with uncertainty about the initial state. The report is available at http://www.informatik.uni-freiburg.de/∼hoffmann/cff-report.ps.gz.

Planning Background

The conformant planning framework we consider adds uncertainty to a subset of the classical ADL language. The subset of ADL we consider is (sequential) STRIPS with conditional effects, i.e., the following. Propositional planning tasks are triples (A, I, G) corresponding to the *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 (for conformant planning, one needs conditional effects as otherwise the same action sequence can hardly achieve the goal from different initial worlds). An action a is *applicable* in a world state w if $w \supseteq pre(a)$. If a is not applicable in w, then the result of applying a to w is undefined. If a is applicable in w, then all applicable conditional effects $e \in E(a)$ get executed i.e., those conditional effects whose condition satisfies $w \supseteq con(e)$ (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 is a *plan* if the world state that results from iterative execution of the actions, starting in the initial world state, leads to a state that contains the goal world state.

The conformant planning setting we consider extends the above with uncertainty about the initial state (as said before, in the TR we also consider non-deterministic effects). The initial state is now a belief state that is represented by a propositional CNF formula $\mathcal{I}$. The possible initial world states are those that satisfy that formula. Slightly abusing the powerset notation, we denote the (possibly exponentially large) set of the possible initial world states with $2^{\mathcal{I}}$. We refer to non-unary clauses in $\mathcal{I}$ as *initial disjunctions*, and to propositions that do not occur in a unary clause as *unknown*.[1] 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, by saying that the result of applying non-applicable actions is undefined, we require that all actions in the plan must be applicable at their point of execution no matter what the initial world state is.

Search Space

As explained in the introduction, we perform a forward search in belief space. The search states are belief states. A belief state s is represented by the initial state representation $\mathcal{I}$ together with the action sequence P that leads from the initial state to s.

For each belief state encountered during search, we compute the sets of known and negatively known propositions; these are defined as follows. Given a conformant planning task $(A, \mathcal{I}, G)$, a belief 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

[1] Syntactically, $\mathcal{I}$ is given in an abbreviated form using the closed-world assumption, where propositions whose value is not known are explicitly specified as *unknown* (rather than listing all the propositions that are known to be false). As a side remark, note that, theoretically, a proposition can be "known" implicitly due to possible unit propagations in $\mathcal{I}$. But then $\mathcal{I}$, i.e. the planning task description, can be simplified.

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 (knowing the propositions that will always be false helps speed up the reasoning, see below). 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 belief state s represented by an action sequence $P \in A^*$, and a proposition p. Deciding whether p is known in s is co-NP complete.*

Proof: We consider the complementary problem. Membership in NP: non-deterministically guess an initial world state, and check if it satisfies $\mathcal{I}$, and if p does not hold upon execution of P. NP-hardness follows by a reduction from SAT. For a SAT instance with variables $\{v_1, \ldots, v_n\}$, the planning instance has the propositions $\{v_1, not\text{-}v_1, \ldots, v_n, not\text{-}v_n\}$ and p. Set $\mathcal{I}$ to $\neg p \bigwedge_{1 \leq i \leq n}(v_i \vee not\text{-}v_i) \bigwedge_{1 \leq i \leq n}(\neg v_i \vee \neg not\text{-}v_i)$. Create a single action $eval$, with empty precondition and, for each clause $l_1 \vee \ldots \vee l_k$, a conditional effect $(\{\overline{l_1}, \ldots, \overline{l_k}\}, \{p\}, \emptyset)$, where negative literals $\neg v_i$ are encoded via $not\text{-}v_i$. Then, p is not known in the state that results from applying $eval$ iff there is a possible initial state such that p is false upon execution of $eval$, which is the case iff there is a possible initial state such that all clauses are fulfilled. ■

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 action sequence as follows. We use a time index to differentiate between values of propositions at different points along the execution of the action sequence. Say the action sequence is $P = \langle a_1, \ldots, a_n \rangle$. We obtain our CNF $\phi(P)$ as follows. We initialize $\phi(P)$ as $\mathcal{I}$ indexed with time 0 (i.e., for each clause $l_1 \vee \ldots \vee l_k \in \mathcal{I}$ we add $l_1(0) \vee \ldots \vee l_k(0)$ into $\phi(P)$). 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 $\neg p_1(0) \vee \ldots \vee \neg p_k(0) \vee \neg p(1)$.
- *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.[2] For the resulting CNF $\phi(P)$, the following holds.

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

Proof: For all possible initial world states $I \in 2^{\mathcal{I}}$, there is exactly one satisfying variable assignment σ to $\phi(P)_I$ (i.e., $\phi(P)$ where all variables at time 0 have been set to their values in I); the truth values assigned by σ correspond exactly to the values that the respective propositions take on at the respective points in the plan. Thus $\phi(P) \wedge \neg p(n)$ is unsatisfiable iff there is no possible initial state $I \in 2^{\mathcal{I}}$ such that p does not hold upon executing P 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 $\phi(P) \wedge \neg p(n)$ 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 propositions by handing the formulas $\phi(P) \wedge p(n)$ 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.

[2] Note 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 can avoid this by introducing a new proposition $p(e)$ for each conditional effect and ensuring that $p(e)$ is true iff e occurs; single frame axioms of the form $\neg p(0) \vee p(e_1)(0) \vee \ldots \vee p(e_n)(0) \vee p(1)$ then suffice. We have not implemented this because in practice actions seldomly affect the same proposition with different effects.

The observation made in Proposition 1 gets us around enumerating all possible initial world 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 formulas 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 (for the respective belief 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 such that p is affected by a conditional effect that *possibly* occurs (all condition propositions 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. P achieves the goal if 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 now adapt this idea to the conformant setting. Specifically, we adapt the heuristic function used in FF (Hoffmann & Nebel 2001), a descendant of the HSP system by Bonet and Geffner (2001).

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 in the following Graphplan-style manner (Blum & Furst 1997; Hoffmann & Nebel 2001). Starting from a 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. If there is no relaxed plan then the planning graph will reach a fixpoint $P_i = P_{i+1}$ 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 does not exist a real plan either.

In the conformant setting, we extend this machinery by sets uP_i of propositions that are unknown at step i. Starting from a belief state s, similar to before, P_0 contains the propositions that are known in s; uP_0 contains 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 uP_{i+1} to $uP_i \setminus P_{i+1}$ — the previously unknown propositions minus those that are now known to be true. Next, new unknown 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 uP_i, and at least one condition is in uP_i. The added propositions of such effects, if they are not already in P_{i+1}, are inserted into uP_{i+1}. When this process is finished, for each proposition in uP_{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 uP_{i+1} and inserted into P_{i+1} instead (with a flag identifying it as an inferred proposition).

Recall that to recognize whether some proposition is known following some action sequence, we basically check to see whether it is implied by a CNF formula encoding the effects of this action sequence on the initial states. During relaxed planning, we do not wish to pay the price of all the SAT checks involved. Instead, we look at a simplified problem that can be solved quickly, while ensuring the completeness (but not soundness, of course) of the approach. The idea is to look at the relaxed problem, i.e., to ignore delete lists, and to select (only) 2 literals out of each clause of the CNF that captures the true belief state semantics. The implications induced by these selected literals are kept as edges in the implication graph: e.g., selecting the literals $\neg p(t)$ and $p'(t+1)$ from a clause yields the implication $p(t) \Rightarrow p'(t+1)$, kept as an edge $(p(t), p'(t+1))$. The inference process is then done as known from 2-CNF reasoning (Aspvall, Plass, & Tarjan 1979).

To get an idea of the implication graph construction, suppose that our current belief state corresponds to the n-step action sequence P. Suppose the relaxed plan construction steps from layer i to layer $i+1$. The implication graph then contains edges $(l(t), l'(t'))$ between timed literals where t ranges from $-n$ to i and t' is either equal to t or to $t+1$. Times $t < 0$ correspond to the time points along the execution of P (i.e., $t = -n$ corresponds to the initial state, and so on), times $t \geq 0$ correspond to the respective relaxed plan graph layer. An edge $(l(t), l'(t'))$ is only inserted if both l and l' are unknown at the respective points in time. An *add effect edge* $(l(t), l'(t+1))$ is inserted if: for $t < 0$, the respective action in P has an effect e that possibly occurs, such that $l' \in add(e)$ and $l \in con(e)$; for $t \geq 0$, there is an action in A_t that has such an effect e that possibly occurs. If e has more than one unknown condition then only *one* such condition l is selected arbitrarily – this is the main simplification we make for computational efficiency. Note that the resulting implication $(l(t), l'(t+1))$ is stronger than the

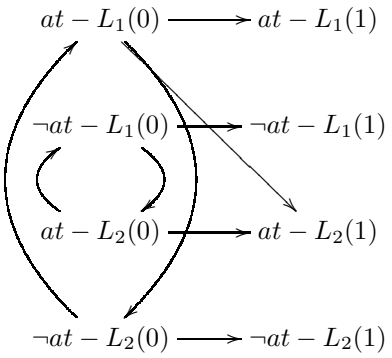

Figure 1: The implication graph for the example.

real implication as really *all* the effect conditions are needed to imply l'. In addition to the add effect edges there are *initial disjunction* edges $(l(0), l'(0))$ for each pair $\overline{l}, l'$ of literals that participate together in such a disjunction (these edges capture a stronger form of the constraints given by the initial disjunctions). Finally, there are *frame axiom edges* $(l(t), l(t+1))$ for propositions l that are unknown (possibly true) at both t and $t+1$.

Having built the implication graph up to layer $i+1$, we use standard, linear-time, validity checks to decide whether the propositions in uP_{i+1} can be inferred and thus moved into P_{i+1}. The validity check for a proposition p looks for backwards paths in the implication graph from $p(i+1)$ to both a proposition p' and its negation. If such a proposition p' is found, then p can be inferred. (To provide an intuition: all possible world states either satisfy p' or $\neg p'$; if p follows in both cases, then it is always true.) During relaxed plan extraction, the actions constituting the add effect edges on the backward paths to p' are, if p becomes a sub-goal, selected as actions into the relaxed plan. For a more detailed description, please consult our TR.

Let us re-consider the example. 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 $pP_1 = \{at\text{-}L_1, at\text{-}L_2\}$, then we insert 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 $(at\text{-}L_2(0), at\text{-}L_2(1))$.

Next, we 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. Figure 1 depicts this 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.

Note that the implication graph we use is a complete, but not necessarily sound, approximation of the set of known propositions. (Precisely, if, starting from a state s, some k-step plan makes a proposition p true then p will be contained in the proposition layer P_k of the relaxed planning graph built for s; but not vice versa.) 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. To understand this, you must remember that the implication graph contains a relaxed encoding of the effects of the current plan fragment leading to the current state. Thus, it may lead to conclusions that are not implied by the full CNF formula.

Repeated States

Forward search spaces in planning are, generally, graphs, i.e., the same search states can be reached via different search paths. In many examples, this happens so frequently that checking for repeated search states becomes crucial for performance. (In a forward search, a repeated search state corresponds to the same search state reached by several action sequences; this happens, e.g., when several actions are independent of each other and can be applied in any order.)

We avoid repeated search states by a hash table technique combined with a belief state domination test. In the classical planning setting, we say that a world state w *dominates* a world state w' if the set of true propositions in w contains the set of true propositions in w'. When classical FF performs search, it checks via a hash table lookup whether the new world state is dominated by some previous world state in the same hash entry. If so, the new state is pruned.

In our conformant extension of FF, we use the same hash table technique. The definition of state domination is extended to belief states as follows. Let s and s' be two belief states reached via the action sequences P and P', respectively. We say that s dominates s' if for every possible initial world state $I \in 2^{\mathcal{I}}$ the world state upon execution of P in I dominates, in the classical sense, the world state upon execution of P' in I. The domination relation between s and s' can be tested by checking for each proposition p whether

there exists a possible initial world state given which p does not hold in s, but does hold in s'. One such check requires a single satisfiability test using the formulas we already generated for s and s'. Let $\phi(P)$ and $\phi(P')$ be the formulas constructed for the action sequences P and P', such that $\phi(P)$ and $\phi(P')$ share the same propositional variables for time 0 propositions but have different propositional variables for all times greater than 0 (i.e., the conjunction of these formulas captures 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 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.[3]

Results

We implemented the techniques described in the previous sections in C, starting from the FF code (which is available at http://www.informatik.uni-freiburg.de/∼hoffmann/ff.html). We call the implemented system Conformant-FF. The system's overall architecture is identical to that of FF. An outline of the architecture is this:

1. Do one trial of "enforced hill-climbing": set the current search state s to the initial state; while s is not a goal state:
   (a) Starting from s, perform a breadth-first search for a world 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 $s := s'$.
2. If enforced hill-climbing failed, invoke complete best-first search, i.e., starting from the initial state expand all world states in order of increasing h value, avoiding repeated states.

Here, $h(s)$ denotes the heuristic value of a search state – the number of actions in a relaxed plan to the state. $H(s)$ denotes the so-called *helpful actions*, which are the only actions considered by the planner during hill-climbing. The helpful actions to a search state are those that could be selected to *start* the relaxed plan. In the classical setting these are those actions at the lowest level of the relaxed planning graph that add a subgoal (at this level). In the conformant setting, $H(s)$ are the actions at the lowest level of the relaxed planning graph that add a subgoal, or that were selected by the implication graph reasoning.

We experimentally evaluated Conformant-FF 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. All testing examples are available for download from http://www.informatik.uni-freiburg.de/∼hoffmann/cff-tests.tgz. The experiments were run on a PC with a Pentium 4 2.4 GHz processor with 0.5 Gigabyte memory and 512KB cache running Linux. Conformant-FF currently uses a very naive standard DPLL solver for the CNF reasoning. Given that Conformant-FF spends around 75% of its runtime in the unit propagation procedure, we expect dramatic runtime improvements from better implementations. In our experiments we compared the performance of Conformant-FF with MBP, GPT, and KACMBP. First, we describe the Conformant-FF vs. MBP experiments. These are the most extensive experiments because MBP appears to be the fastest among the conformant planners with a high-level input language.

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. Conformant-FF then generates the set of ground 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 actions, 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. For all test suites in the table (see leftmost column), the upper row provides Conformant-FF's runtimes whereas the lower row provides MBP's runtimes.

In the well-known Bomb-in-the-toilet domain we used four test suites, each of which contains five example instances. In the "Bomb-x-x" suite instances the number of toilets is the same as the number of bombs. In the "Bomb-x-c" suites the number of bombs varies, but the number of toilets is fixed to c. Across all these test suites, the examples 1 to 5 contain 5, 10, 20, 50, and 100 bombs, respectively. The top right corner of Table 1, e.g., says that Conformant-FF solves the task with 100 bombs and 100 toilets in no more than 2.39 seconds. MBP is competitive with Conformant-FF when there are very few toilets, but gets outperformed quickly as the number of toilets increases.

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 suite the task is to move into a corner of the grid. In the "Cube-center" test suite the task is to move into the center of the grid. In both test suites, the value of n is 3, 5, 7, 9, and 11 for examples 1 to 5, respectively. Conformant-FF is

[3] One can optimize the domination test by checking only those propositions p that are unknown in both s and s': if p is unknown in s but known in s' then s does not dominate s'; if p is known in s then p needs not be checked.

Test suite	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 suites. Times are in seconds, dashes indicate time-outs.

Test suite	exp1	exp2	exp3	exp4	exp5
Blocksworld-2	0.03	0.03	0.02	0.16	5.41
Blocksworld-2	0.77	16.83	-	-	-
Blocksworld-3	0.00	0.00	0.02	0.47	3.54
Blocksworld-3	1.10	10.86	-	-	-
Blocksworld-4	0.01	0.11	0.21	70.64	4.48
Blocksworld-4	0.71	18.85	-	-	-
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 test suites. Times are in seconds, dashes indicate time-outs.

clearly better suited than MBP to find a corner. None of the planners scales very well to the move into the center; MBP is somewhat better at that.

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. Note that this is different from a truly non-deterministic effect in that the outcome of the effect will be the same for all eggs, only we don't know what this constant outcome is. Still, there is no conformant plan to our Omlette tasks. In our test suite, 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 belief states).

The Ring domain is 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 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. A solution plan moves once through the ring and applies the "close" and "lock" actions after each move. Our test suite contains the tasks with n value 2, 3, 4, 5, and 6, respectively. The poor performance of Conformant-FF here is due to two weaknesses. First, a lack of informativity of the heuristic function in the presence of non-unary effect conditions – all but one of which are ignored by the implication graph. (In Ring, the conditions of "lock" are that one is in the right room *and* that the window is already closed.) Second, if a new belief state shares the same known propositions with an already seen belief state, then the states are checked for domination using SAT calls – without any further pre-checks to see if the CNF reasoning can be avoided. (In Ring, no proposition becomes known until the entire task is solved; turning the repeated states check off, Conformant-FF solves the five examples in 0.00, 0.29, 4.76, 50.76, and 445.50 seconds respectively.) We are currently experimenting with methods to improve on these two weaknesses (see the discussion of future work).

In the Safe domain, a safe has one out of n possible combinations, and one must try all combinations in order to open the safe. Our test suite contains the examples with n values 5, 10, 30, 50, and 70, respectively. Conformant-FF and MBP scale roughly similar here. We remark that, similar to what we observed in Ring, no proposition becomes known before the task is solved so that most of Conformant-FF's runtime is spent in SAT calls for checking belief 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.

In Table 2 we provide the results for our "mixed" benchmark domains, enriching classical planning benchmarks with uncertainty. Again the upper rows provide Conformant-FF's runtimes whereas the lower rows provide MBP's runtimes.

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. The uncertainty in our test suites is that the top blocks on each initial stack are arranged in an unknown order. In the test suites "Blocksworld-k" the order of the top k blocks on each stack (which might be the whole stack) is unknown. Putting a block x from a block y onto the table has conditional ef-

fects: (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. Across our three test suites, the examples are generated with the software by Slaney and Thiebaux (Slaney & Thiebaux 2001), and contain 5, 6, 7, 13, and 20 blocks respectively. Conformant-FF scales well and outperforms MBP dramatically. 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. We note that in some of these cases, MBP stopped before our 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. The amount of uncertainty increases with the size of the cities. In our test suites "Logistics-k" the city size (which is the same for each city) is k. Across all suites, the instances are randomly generated ones with the following parameters. Each city contains one truck. The first four examples contain 2-3 cities, 2-4 packages, and 1-2 airplanes; to demonstrate Conformant-FF's fine scaling behaviour, the largest examples contain 10 cities, 10 packages, and 10 airplanes (the plan found for the largest "Logistics-4" example, e.g., has 120 actions in it). MBP solves only the smaller instances.

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 some keys to specified goal positions. We introduced uncertainty about the locked positions on the grid that must be opened in order to solve the respective task. 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. In our test suites "Grid-k" the unknown locks have k possible shapes. Across the suites, the instances are the original five examples "prob01" ... "prob05" as used in the AIPS-98 competition. We remark that these tasks are very challenging. While FF finds a 174-steps plan for "prob05" without uncertainty, the plan Conformant-FF finds for "prob05" with $k = 2$ has 194 steps, for $k = 3$ it are 214 steps, for $k = 4$ it are 254 steps (with uncertainty, more keys must be transported). MBP can not even solve the "prob01" instances.

To cover a broader spectrum of conformant planners, we ran additional experiments with GPT and KACMBP. GPT is known to be slower than MBP on the tranditional domains, but we wanted to verify that this is true for the mixed domains, too. Thus, we conducted a number of experiments with GPT on a sample of mixed domains instances. In almost all of them, GPT was slower than MBP and much slower the Conformant-FF. For example, in three of the smallest Blocksworld-2 examples, the per-instance runtimes

Test suite	exp1	exp2	exp3	exp4	exp5
Bomb-x-x	0.00	0.00	0.00	0.24	2.39
Bomb-x-x	0.06	0.17	1.47	54.56	-
Bomb-x-1	0.00	0.00	0.07	4.69	113.70
Bomb-x-1	0.00	0.03	0.33	2.34	2.38
Bomb-x-5	0.00	0.01	0.10	4.68	113.24
Bomb-x-5	0.04	0.19	1.01	4.31	4.31
Bomb-x-10	0.00	0.00	0.04	3.32	97.50
Bomb-x-10	0.19	0.51	2.28	9.11	9.02
Cube-corner	0.00	0.06	0.48	1.80	5.66
Cube-corner	0.00	0.00	0.00	0.00	0.00
Cube-center	0.19	-	-	-	-
Cube-center	0.02	0.01	0.01	0.08	0.08
Ring	0.02	1.35	50.61	-	-
Ring	0.00	0.00	0.00	0.01	0.01
Omlette	0.02	0.03	0.30	1.14	3.74
Omlette	0.03	0.09	0.22	733.43	-
Blocksworld-2	0.01	0.03	0.01	0.14	4.75
Blocksworld-2	0.2	0.25	1.04	-	-
Blocksworld-3	0.01	0.01	0.01	0.45	2.98
Blocksworld-3	0.05	1.07	-	-	-
Blocksworld-4	0.00	0.03	0.02	0.15	4.80
Blocksworld-4	0.06	0.88	1.16	-	-
Logistics-2	0.00	0.01	0.01	0.02	3.33
Logistics-2	0.08	51.38	0.47	478.72	-
Logistics-3	0.00	0.02	0.03	0.06	41.80
Logistics-3	0.97	-	67.6	-	-
Logistics-4	0.01	0.12	0.01	0.07	71.20
Logistics-4	0.94	-	57.52	-	-
Grid-2	0.00	0.01	0.03	0.05	0.03
Grid-2	1.3	12.29	30.13	731.57	-
Grid-3	0.01	0.02	0.03	0.03	0.10
Grid-3	10.74	276.35	265.22	-	-
Grid-4	0.02	0.01	0.01	0.06	0.14
Grid-4	97.15	-	-	-	-

Table 3: Conformant-FF runtime (upper rows) vs. KACMBP runtime (lower rows) in our test suites. Times are in seconds, dashes indicate time-outs or out-of-memory.

(pure solution times excluding parsing and compilation) we got are 41.30, 93.82, and 936.88 seconds for GPT as compared with 6.17, 13.18, and 102.50 for MBP, and a few milliseconds for Conformant-FF. For the smallest three instances of Logistics-2, GPT takes 0.86, 28.74, and 382.35 seconds, MBP takes 1.63, 16.36, and 35.54 seconds, and Conformant-FF takes a few milliseconds. In the Grid suites, GPT kept producing error messages for reasons we could not figure out. These results clearly indicate that MBP is indeed superior to GPT in these domains, and that Conformant-FF is superior to both.

KACMBP is an earlier version of MBP, specifically designed for conformant planning, using a heuristic function that estimates the number of distinct world states in a belief state. KACMBP is no longer supported, has no documentation, and has no high-level input language. One has to explicitly specify *each* ground action, a tedious and error-prone task. Yet, KACMBP appears to be much better than MBP on conformant planning problems. We ran KACMBP with the same parameters as used in the experimental se-

tups included with the KACMBP download. Our results appear in Table 3. The upper rows provide Conformant-FF's runtimes whereas the lower rows provide KACMBP's runtimes. In the Grid domain, the instances used are randomly generated ones that are much smaller than those used in Table 2; in Blocksworld, for each entry we ran just one of the 25 random instances; in all other domains, the instances are the same as previously. KACMBP is much better than Conformant-FF in Ring, Cube, and in Bomb examples with few toilets. Conformant-FF performs much better in the Bomb-x-x examples, the Omlette domain and in all the mixed domains.

From our results in the "mixed" domains, we conclude that our approach has 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, KACMBP, and GPT. We consider this an important advantage as, in a real-world domain, one would surely expect only parts of the problem to be uncertain.

Discussion

We introduced a new, lazy, approach for representing belief states in conformant planning. Based on that, we extended the classical heuristic forward search planner FF (Hoffmann & Nebel 2001), into conformant planning. The necessary knowledge about belief states – the known propositions – is computed via a CNF reasoning technique. The relaxed planning method that FF uses to compute its heuristic function is modified accordingly, with an approximate linear-time reasoning about known propositions using a 2-CNF projection of the formula that captures the true belief state semantics. The resulting planner Conformant-FF is competitive with the state-of-the-art conformant planners MBP, KACMBP, and GPT. Further, Conformant-FF shows the potential to combine the strength of FF with conformant abilities. We emphasize that these results are obtained using a *completely naive* SAT solver as the underlying reasoning technique.

Conformant-FF shares various ideas and techniques with other conformant planning algorithms. CGP (Smith & Weld 1998), the first specialized conformant planner, uses the Graphplan algorithm, which Conformant-FF adopts to generate its heuristic function. However, CGP uses one planning graph for each possible initial world state, an approach that does not scale up well. GPT (Bonet & Geffner 2000) introduced the idea of planning in belief space using forward heuristic search which Conformant-FF uses. MBP and KACMBP (Cimatti & Roveri 2000; Bertoli & Cimatti 2002) developed this technique further, by using BDDs to represent belief states. The methods used by these planners to compute the heuristic function are quite different from Conformant-FF's. (Also, as said before these systems are capable of doing much more than conformant planning.) Petrick and Bacchus (Petrick & Bacchus 2002) introduced the use of logical formulas to represent the current state of *knowledge* of the agent. Conformant-FF adopted the idea of storing logical knowledge about the belief state. The major difference between the two approaches is that Conformant-FF stores only a partial knowledge, inferred from a standard task description, while Petrick and Bacchus *model* planning tasks at the knowledge level (which loses expressive power (Petrick & Bacchus 2002)).

SAT solvers play an important part in QBFPLAN (Rintanen 1999) and CPLAN (Ferraris & Giunchiglia 2000). CPLAN, in particular, introduced the use of SAT techniques for testing the validity of certain properties in the final state of a plan, which we use throughout the search process. CPLAN generates candidate conformant plans (using a SAT encoding, too) and then tests them to see if they are legal and if they lead to a goal state. Conformant-FFś use of unsatisfiability tests to check whether a proposition holds following an action sequence is virtually the same as the test CPLAN uses to recognize whether a candidate plan leads to a goal state. QBFPLAN uses more powerful (and more complex) satisfaction tests where quantified boolean formulas are used rather than propositional formulas.

We are currently working on various ideas that can help us address the two main weaknesses of Conformant-FF, as can be observed (e.g.) in the Ring domain. The first is Conformant-FF's use of only a single condition proposition per effect within the relaxed planning problem. This can lead to inadequate heuristic values when there are effects with more than one unknown condition. The second is the large overhead of repeated states checking in problems where there is a lack of different known propositions in the belief states. To overcome the first weakness, one can parameterize the heuristic function by using a k-projection of the belief state CNF; as the value of k increases, the heuristic is likely to become slower but more informative (with $k > 2$ the inference reasoning can/has to be done by a SAT solver). In our experiments the single SAT calls (for computing known propositions) were always very cheap, while the number of expanded belief states (and thus the number of calls to the heuristic function) was rather low. So values $k > 2$ are worth trying. To overcome the second weakness, one can introduce incomplete pre-tests to avoid full domination tests of state pairs s and s'. For example, say ϕ is the CNF whose satisfiability must (repeatedly) be tested for deciding whether s dominates s'. If even a 2-projection of ϕ is solvable, then ϕ is solvable, and s does not dominate s'.

Another interesting idea how to overcome Conformant-FF's current limitations is to try and combine the heuristic principles used in Conformant-FF and KACMBP. While the former system estimates goal distances, the latter system estimates belief state size. In both cases, states with lower heuristic value are preferred during search. The comparative data in Table 3 reveals that the two kinds of heuristics have very different strengths and weaknesses. KACMBP outperforms Conformant-FF in cases where planning comes close to reducing the initial uncertainty. Conformant-FF becomes superior when more classical planning aspects come into the play. A promising idea is to use a hybrid heuristic (h, b) where h estimates goal distance, b estimates belief state size, and $(h(s), b(s)) < (h(s'), b(s'))$ if $h(s) < h(s')$ or $h(s) = h(s')$ and $b(s) < b(s')$. To improve Conformant-FF, the main question to answer here is if, and how, belief state size can be estimated based on our data structures. Similarly, one can try to improve KACMBP by estimating goal

distances based on KACMBP's data structures.

Additional extensions we plan for Conformant-FF are: implement the support needed for handling non-deterministic effects (as described in our TR the extensions needed for this are moderate); implement more elaborate SAT solving strategies; introduce an ability to handle observations.

Acknowledgements. We would like to thank Piergiorgio Bertoli and Alessandro Cimatti for their help in using MBP and KACMBP, Blai Bonet for his help with GPT, and the anonymous reviewers for their useful input. Ronen Brafman is partially supported by the Paul Ivanier Center for Robotics and Production Management at Ben-Gurion University.

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. 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.

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

Ferraris, P., and Giunchiglia, E. 2000. Planning as satisfiability in nondeterministic domains. In *AAAI'00*, 748–753.

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

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.