Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 170 (2006) 507–541
Artificial Intelligence
www.elsevier.com/locate/artint

# Conformant planning via heuristic forward search: A new approach ☆

Jörg Hoffmann [a,*], Ronen I. Brafman [b]

[a] *Max Planck Institute for Computer Science, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany*
[b] *Department of Computer Science, Ben-Gurion University, PO Box 653, Beer-Sheva 84105, Israel*

Received 16 July 2004; received in revised form 2 December 2005; accepted 4 January 2006

Available online 9 February 2006

**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 implication tests on propositional formulas in conjunctive normal form (CNF) that capture the action sequence semantics. Based on this approach, we extend the classical heuristic forward-search planning system FF to the conformant setting. The key to this extension is an appropriate extension of the relaxation that underlies FF's heuristic function, and of FF's machinery for solving relaxed planning problems: the extended machinery includes a stronger form of the CNF implication tests that we use to reason about the effects of action sequences. Our experimental evaluation shows the resulting planning system to be superior to the state-of-the-art conformant planners MBP, KACMBP, and GPT in a variety of benchmark domains.


*Keywords:* Planning under uncertainty; Heuristic search planning; Relaxed plan heuristic

## 1. 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 [1] introduced the idea of planning in belief space using heuristic

☆ A preliminary version of this paper appears in [R. Brafman, J. Hoffmann, Conformant planning via heuristic forward search: A new approach, in: S. Koenig, S. Zilberstein, J. Koehler (Eds.), Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04), Whistler, Canada, Morgan Kaufmann, San Mateo, CA, 2004, pp. 355–364].

* Corresponding author.
*E-mail addresses:* hoffmann@mpi-sb.mpg.de (J. Hoffmann), brafman@cs.bgu.ac.il (R.I. Brafman).

0004-3702/$ – see front matter 
doi:10.1016/j.artint.2006.01.003

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 a number of planners, Bertoli, Cimatti, and Roveri [2,3] 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.[1]

In this paper we suggest a third, lazy, approach to represent belief states. The approach assumes, in its current form, that the goal and action preconditions are restricted to simple *conjunctions* of propositions, as in STRIPS.[2] We observe that, for conformant planning in this setting, 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. 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 *act* 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 $p$ is always true after executing *act*. We say that such propositions are *known* in $s$. Doing the test for a proposition is hard in general; the corresponding decision problem is easily shown to be co-NP complete. Our implementation does the test by checking whether the proposition is implied by a CNF formula that captures the semantics of the action sequence.

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 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 largely naive DPLL [4,5] 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 [6] is one particularly successful system that uses this technique. Beside the search space representation, the main contribution of our work is the adaptation of the relaxation, and of FF's heuristic function, to the conformant setting. We discuss different options how to extend the relaxation to the conformant setting. After choosing one of these options, we extend FF's relaxed planning process to handle such relaxed conformant problems. We enrich FF's machinery with a stronger form of the CNF reasoning that is done to determine the known propositions in the belief states. In a nutshell, the stronger reasoning corresponds to a 2-CNF projection of the real formula that would be needed to decide about known propositions during relaxed planning. Relative to that formula, the reasoning is thus complete, but not sound. We prove that the reasoning is complete *and* sound relative to our extended relaxation.

Our implementation, which we call Conformant-FF, is based on the FF code, and uses the same overall search arrangement as FF. We experimentally compare Conformant-FF with the state-of-the-art conformant planners MBP and KACMBP [3]—the latter is a conformant planner that has more advanced conformant planning heuristics than MBP. In a number of traditional conformant benchmarks, our experiments show Conformant-FF to be competitive: it often outperforms MBP, but is generally not as efficient as KACMBP. On the positive side, our approach demonstrates 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 as well as KACMBP. We also run limited experiments with GPT

[1] It should be noted that GPT and the BDD-based MBP planner are capable of doing much more than conformant planning.

[2] Extension to handle disjunctions is conceptually easy; we outline in the conclusion of the paper how to do so.

and with POND [7], a recent conformant planner using techniques related to Conformant-FF. The results indicate that, in many domains, Conformant-FF outperforms these planners too.

In our current implementation of Conformant-FF, we only deal with uncertainty about the initial state, i.e. we do *not* deal with non-deterministic effects. The extensions necessary to make Conformant-FF handle such effects are conceptually easy, with one important exception, namely repeated states checking. While there is a straightforward extension for repeated states checking in the presence of non-deterministic effects, it is more than doubtable that this extension would work well, i.e., have reasonable pruning power, in practice. We have not yet worked out an extended technique that appears practical. We leave this open for future work. To inform the reader about the main issues in this important aspect of Conformant-FF's further development, we include an extra section on non-deterministic effects. In the other parts of the paper, only uncertainty about the initial state is considered.

The paper is organized as follows. Section 2 briefly describes the planning framework we consider. Section 3 describes the search space representation and the computation of known propositions. Section 4 discusses conformant relaxations and explains our extension of FF's heuristic function (this is the technically most challenging part of our work, and the section forms more than half of the technical part of this paper). Section 5 explains how we avoid repeated search states. Section 6 gives some implementation details and our empirical results. Section 7 discusses non-deterministic effects, Section 8 discusses related work. Section 9 concludes the paper and gives an outlook on future work. All non-trivial proofs are moved into Appendix A, and are replaced in the text by proof sketches, to improve readability.

## 2. 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 *goals*. $I$ and $G$ are sets of propositions. 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 $pre(a) \subseteq w$, i.e., if the world state satisfies all of $a$'s preconditions. 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 conditional effects $e \in E(a)$ get executed whose condition satisfies $con(e) \subseteq w$ (unconditional effects have $con(e) = \emptyset$); we say that such effects *occur*. Executing a conditional effect $e$ in $w$ results in the world state $w - del(e) + add(e)$. We require that actions are not self-contradictory, i.e., if there is a proposition $p$ such that, in world state $w$, $p \in add(e)$ and $p \in del(e')$ for two effects $e$ and $e'$ that both occur, then the result of applying the respective action is undefined. An action sequence is a *plan* (a solution) if the world state that results from iterative execution of the actions, starting in the initial world state, leads to a *goal state*, i.e. to a world state that contains all the goals.

The conformant planning setting we consider extends the above with uncertainty about the initial state. 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 $2^{\mathcal{I}}$. An action sequence is called *executable* if all actions in it are applicable at their point of execution no matter what initial world state one starts from. An action sequence $act \in A^*$ is a *plan* for a task $(A, \mathcal{I}, G)$ if, for any possible initial world state $I \in 2^{\mathcal{I}}$, executing $act$ in $I$ results in a goal state. (Note that plans are executable by this definition.)

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. An alternative definition is to say that, if an action is not applicable in a world state, then applying the action does not change that world state. This alternative definition is of theoretical interest for our heuristic function (Section 4.1), and we call it *generous*. If an action sequence $act$ solves a task $(A, \mathcal{I}, G)$ according to the generous definition of the result function, then $act$ is called a *generous plan* for $(A, \mathcal{I}, G)$. Note that, given a task $(A, \mathcal{I}, G)$, if for all actions $a$ and effects $e \in E(a)$ one includes $pre(a)$ into $con(e)$, and then sets $pre(a) := \emptyset$, then the resulting task under the non-generous semantics

corresponds exactly to the original task under the generous semantics. So the generous semantics can be simulated with the non-generous semantics and using the latter does not make us lose generality.[3]

## 3. 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 *act* that leads from the initial state to $s$. In line with our definition of action semantics, during search we make sure that every action in the sequence will be applicable, no matter what the initial world state is; that is, we ensure that every considered action sequence is executable. The reasoning necessary for this, as well as for detecting goal belief states, is based on the computation of known propositions.

For each belief state encountered during search (including the initial belief state), 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 executable action sequence $act \in A^*$, and a proposition $p$, we say that $p$ is *known* in $s$ if, for all $I \in 2^{\mathcal{I}}$, executing *act* 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 *act* 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.

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

**Proof.** This result essentially follows from Theorem 2 of [9]. The following is a direct argument: 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 *act*. NP-hardness follows by a trivial reduction from SAT. Given a propositional CNF formula $\phi$, just take that formula to be the initial formula $\mathcal{I}$, and ask if some proposition $p$ not contained in $\phi$ is not known in the state that results from executing the empty action sequence. This is the case, by definition, iff $\phi$ is satisfiable: $p$ can only be known if $2^{\mathcal{I}}$ is empty. □

Note that the proposition holds independently of the allowed goal conditions, and that the hardness result holds even if there are no actions at all. The hardness of the decision problem follows already from the fact that the initial state is a CNF formula.

We compute the sets of known and negatively known propositions in a search (belief) state by using a CNF corresponding to the semantics of the respective (executable) 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 $act = \langle a_1, \ldots, a_n \rangle$. We obtain our CNF $\phi(act)$ as follows. We initialize $\phi(act)$ as $\mathcal{I}$ indexed with time 0 (i.e., for each clause $l_1 \vee \cdots \vee l_k$ in $\mathcal{I}$ we add $l_1(0) \vee \cdots \vee l_k(0)$ into $\phi(act)$). We then use $a_1$ to extend $\phi(act)$:

- *Effect Axioms*. For every effect $e$ of $a_1$, $con(e) = \{c_1, \ldots, c_k\}$, and every proposition $p \in add(e)$, we insert the *add* effect axiom clause $\neg c_1(0) \vee \cdots \vee \neg c_k(0) \vee p(1)$. For every proposition $p \in del(e)$, we insert the *delete* effect axiom clause $\neg c_1(0) \vee \cdots \vee \neg c_k(0) \vee \neg p(1)$.
- *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 $c_1, \ldots, c_n$ such that $c_i \in con(e_i)$ we insert the *positive* frame axiom clause $\neg p(0) \vee c_1(0) \vee \cdots \vee c_n(0) \vee p(1)$. (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 $c_1, \ldots, c_n$ with $c_i \in con(e_i)$ the *negative* frame axiom clause $p(0) \vee c_1(0) \vee \cdots \vee c_n(0) \vee \neg p(1)$ (if $p$ was false before and has not been added, it is still false after $a_1$).

[3] One can also simulate the non-generous semantics with the generous semantics, by introducing a new goal proposition that is true initially, and that gets deleted whenever an action is executed whose precondition is not satisfied [8].

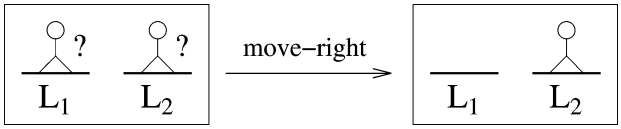


Fig. 1. A simple example task.

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. Note that, if one of the $e_i$ that deletes/adds $p$ has an empty effect condition, then no positive/negative frame axioms are inserted for $p$.[4] The resulting CNF $\phi(act)$ captures the semantics of *act* in the following sense.

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

**Proof.** Given fixed values for all variables at time 0, the axioms explicitly enforce a value for every proposition at every time step—either it is a (conditional) effect or its value remains the same due to an appropriate frame axiom. The enforced proposition values correspond exactly to the values that the respective propositions take on at the respective points in the plan. □

Note that $\phi(act)$ is basically a description of *act*'s semantics in the situation calculus [10]. The formula does not need to talk about action preconditions since *act* is assumed to be executable. (If *act* is not executable then Proposition 2 does not hold because a non-applicable action will result in an undefined state, but not affect the satisfying assignment to the formula.)

Proposition 2 immediately leads to the following simple result, which we use to compute the sets of known propositions.

**Proposition 3.** *Given a conformant planning task* $(A, \mathcal{I}, G)$*, a belief state* $s$ *represented by an executable n-step action sequence* $act \in A^*$*, and a proposition* $p$*. Then* $p$ *is known in* $s$ *iff* $\phi(act)$ *implies* $p(n)$*.*

**Proof.** The formula $\phi(act) \wedge \neg p(n)$ is unsatisfiable, with Proposition 2, iff there is no $I \in 2^{\mathcal{I}}$ such that $p$ does not hold upon executing *act* in $I$. □

We remark that Proposition 3 is not a new result; it can be obtained using Theorem 2 of [9]. The proof via Proposition 2 is kept in this paper just for reasons of self-containedness.

We use Proposition 3 to compute the set of known propositions as follows. Start with the empty set. Then, for each proposition $p$, hand $\phi(act) \wedge \neg p(n)$ 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 $\phi(act) \wedge p(n)$ over to the SAT solver.

**Example 1.** Say we have a robot that is initially at one out of two locations, modeled as $\mathcal{I} = \{at\text{-}L_1 \vee at\text{-}L_2, \neg at\text{-}L_1 \vee \neg at\text{-}L_2\}$. Our goal is to be at $L_2$, and we have a *move-right* action that has an empty precondition, and the conditional effect $(con = \{at\text{-}L_1\}, add = \{at\text{-}L_2\}, del = \{at\text{-}L_1\})$. A graphical sketch of this example is given in Fig. 1; the example will be re-used throughout the paper.

The known propositions in the search state $s$ corresponding to the sequence $act = \langle move\text{-}right \rangle$ are computed as follows. The formula $\phi(act)$ consists of the following clauses:

[4] Note also that the number of frame axioms is exponential in the number of distinct conditional effects of a single action that can add/delete the same proposition. One can avoid this by introducing, for each time point $t$, a new proposition $p(e)(t)$ for each conditional effect $e$ of the action applied at $t$, and ensuring that $p(e)(t)$ is true iff $e$ occurs at time $t$. Single frame axioms of the form $\neg p(t) \vee p(e_1)(t) \vee \cdots \vee p(e_n)(t) \vee p(t+1)$ then suffice. We do not do this because in practice actions rarely affect the same proposition with different effects.

- $at\text{-}L_1(0) \vee at\text{-}L_2(0)$ and $\neg at\text{-}L_1(0) \vee \neg at\text{-}L_2(0)$: initial state formula.
- $\neg at\text{-}L_1(0) \vee at\text{-}L_2(1)$: add effect axiom for *move-right*.
- $\neg at\text{-}L_1(0) \vee \neg at\text{-}L_1(1)$: delete effect axiom for *move-right*.
- $\neg at\text{-}L_1(0) \vee at\text{-}L_1(0) \vee at\text{-}L_1(1)$: positive frame axiom for $at\text{-}L_1$. The axiom says that, if $at\text{-}L_1$ is true at 0, and the delete effect of *move-right* does not occur, then $at\text{-}L_1$ is still true at 1. Note that this cannot hold (since the delete effect occurs if $at\text{-}L_1$ is true), and the clause 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 $at\text{-}L_1$.
- $at\text{-}L_2(0) \vee at\text{-}L_1(0) \vee \neg at\text{-}L_2(1)$: 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 $\phi(act) \wedge \neg at\text{-}L_1(1)$. The result is "sat": a satisfying assignment $\sigma$ 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: $\phi(act) \wedge \neg at\text{-}L_2(1)$ 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 state 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 Proposition 3 gets us around enumerating all possible initial world states for computing whether a given proposition is known upon execution of *act* or not. While we *do* need to perform worst-case exponential reasoning about the formula $\phi(act)$, 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 $\phi(act)$ 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 *act*. In effect, $\phi(act)$ only contains variables whose value is unknown at the respective points of *act*'s execution.
- Make SAT calls only on propositions $p$ such that $p$ may change its truth value due to a conditional effect $e$ that *possibly* occurs: all of $e$'s condition propositions are either known or unknown, and at least one of them is unknown. (For the initial belief state, SAT calls can be avoided for propositions occurring in unary clauses.)

Once the known propositions in $s$ are computed, we can easily check if *act* is a plan: this is the case iff all goal propositions are known in $s$; similarly for action preconditions, see below. Note that one could check entailment of the goal (of an action precondition) with a single SAT call by negating the goal (the precondition), and conjoining the resulting disjunction with $\phi(plan)$. Decomposing this test into several tests for the individual propositions has the advantage that the knowledge about the propositions can be re-used for other tests.

The actions that are applicable to $s$, and that are used to generate the successor states, are those actions whose preconditions are all known in $s$, and whose effects cannot be self-contradictory in $s$. To make sure of the latter, we (potentially) do some additional SAT tests. Two effects $e$ and $e'$ are potentially self-contradictory if $add(e) \cap del(e') \neq \emptyset$ or $add(e') \cap del(e) \neq \emptyset$. For all such pairs $e$ and $e'$ of effects of each action whose preconditions are known in $s$, we check whether $e$ and $e'$ occur together in $s$ for a possible initial world state. The check is made by a SAT call on the formula $\phi(act) \wedge \bigwedge_{c \in con(e) \cup con(e')} c(n)$. If the result is "sat" then $e$ and $e'$ can occur together and we skip the action. 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.

## 4. 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. In every world state $w$ during the search, the relaxed task starting from $w$ is solved, and the length of the relaxed plan is taken to be the heuristic value to $w$. We now extend this idea to the conformant setting. This extension forms the technically most challenging part of our work, so we discuss it in detail.

This section is structured as follows. In Section 4.1, we discuss the advantages and drawbacks of possible options for an extended relaxation, choosing one of them for use in our system. In Section 4.2, we adapt the algorithm used by FF to solve relaxed tasks, and we prove that the adapted algorithm is sound and complete relative to the (chosen) extended relaxation. In Section 4.3, we explain two optimizations of the algorithm that we use in our implementation, in Section 4.4 we discuss two variations of the algorithm that we have also implemented.

### *4.1. Conformant relaxations*

In the design of a heuristic function, two main decisions have to be made. The first of these is whether the heuristic should be *admissible*.[5] Admissible heuristics can be used to find optimal solutions using A*. In our work, as in the original FF system we sacrifice admissibility, and with that optimality, for runtime performance (see also below).

The second design decision to be taken in the creation of a heuristic function is a trade-off between the *informativity* and the computational efficiency of that function.[6] The more time we invest into reasoning inside the heuristic function, the more informative will that function be; at the extreme ends we have an exact heuristic function solving the entire problem to obtain its return value, and a trivial heuristic function returning always the same value.

In pure STRIPS planning, if we ignore the delete lists then the complexity of deciding plan existence goes down from PSPACE to P [11]. The complexity of deciding *bounded* plan existence is still NP-hard without delete lists, so it would be impractical to compute a provably shortest relaxed plan, and with that an admissible heuristic function. But we *can* use this relaxation as the basis for a heuristic function, by computing a relaxed plan in polynomial time, and taking its length as the heuristic value—this is done in FF. In STRIPS, not much more variation seems to be possible in this particular trade-off between informativity and computational efficiency: it is unclear how to ignore only some of the delete lists without falling back into the general, PSPACE-hard, planning case.

In the conformant setting we consider here, the design decisions possible in defining the relaxation, and thereby the heuristic function, are much more interesting and manifold. First of all, as a simple consequence of Proposition 1, deciding plan existence is still co-NP-hard without delete lists, and indeed without any actions. If one wants to obtain a heuristic function computable in worst-case polynomial time, then there is no way around relaxing (not only the actions but also) the initial state formula. On the other hand, in our approach to belief space search, as described in the previous section, several SAT calls may be made for every single generated search state. These SAT calls are worst-case exponential but seem to be feasible in practice, and a priori there is no reason why the same successful use of SAT solvers as part of the computation should not be possible in the heuristic function. It is important to note here that computing optimal relaxed plans, while also "only" NP-complete in the classical setting, seems practically infeasible; at least no technique for doing it reasonably efficiently has yet been developed (published). So we will stick, in the conformant setting, to not trying to compute optimal relaxed plans.

To avoid the computational complexity arising from destructive interactions between action effects, we also definitely want to stick to ignoring the delete lists in our conformant relaxation. The design decision to be taken is what other simplifications/relaxations we make to the conformant task description.

First, we observe that deciding conformant plan existence with empty delete lists can be done by a single CNF satisfiability test *if one uses the generous action execution semantics*. Recall from Section 2 that, under the generous execution semantics, non-applicable actions do not affect the world state, rather than resulting in an undefined state.

**Theorem 1** *($A^+$-complexity). Given a conformant task $(A, \mathcal{I}, G)$ where the delete lists of all effects are empty. Deciding whether there exists a generous plan for $(A, \mathcal{I}, G)$ is co-NP-complete.*

**Proof sketch.** Hardness follows directly from Proposition 1. Membership can be shown by constructing a CNF formula that encodes the semantics of executing all actions in parallel $m$ times, where $m$ is the number of distinct conditional effects in the task. The formula implies the goals (at time $m$) iff there exists a generous plan for $(A, \mathcal{I}, G)$. □

The CNF formula construction in the proof does not work if one assumes a non-generous execution semantics of actions, because in this case one cannot easily model a "parallel" execution of all actions. Intuitively, the generous

[5] A heuristic function is called admissible if it always underestimates the distance to the nearest solution state.

[6] Informativity is an informal concept meaning the quality of the guidance information that a heuristic provides.

execution semantics gives us the possibility to "select", for every initial world state, appropriate subsets from the parallel action steps. It is an open question what the complexity of deciding plan existence is in the absence of delete lists with a non-generous execution semantics. We remark that Theorem 8(iv) of [12] describes a slightly more general case (e.g., delete lists are allowed) for which the existence of a plan of bounded size is $D^P$-complete. This implies that our problem is in $D^P$ (the class of all languages that can be defined as the intersection between an NP and a co-NP language).

Theorem 1 suggests to relax conformant planning tasks (only) by ignoring the delete lists, and then use a single SAT call per search state to compute a relaxed plan.[7] We did not try this out in an implementation because it does not appear very promising, or at least poses a number of serious difficulties. First of all, the size of the formula to be considered is a square function in the number of (ground) action effects and will become very large when there are many actions. It seems likely that SAT reasoning about such a large formula for every search state will take a lot of time. One might be able to ameliorate the difficulty by not doing a single SAT check asking for goal fulfillment after $m$ parallel action steps, but instead looking at a sequence of CNF formulas with $1, 2, \ldots$ steps, and stopping when the goal first becomes implied (or when a termination criterion is met). Still even with this approach the CNF formulas reasoned about will be large. Another serious difficulty arises from the fact that we are looking for a heuristic estimate of the *number of actions* needed to achieve the goal. The experience from classical planning tells us that the number of parallel (relaxed) action steps is typically a very bad estimate for this number [13]. So, say we have found a CNF for $k$ parallel action steps so that the goals are implied. That is, we have proved by an unsatisfiability result that this CNF implies truth of the goals after the $k$th step. So how do we actually extract a relaxed plan? What we need to find is a (minimal, ideally) subset of the actions (of the actions at the time steps) that still implies goal fulfillment. This corresponds to a (minimal, ideally), unsatisfiable subset of our CNF. There are techniques that extract unsatisfiable subsets of a CNF from Davis-Putnam style refutation proofs [14], by keeping track of the clauses involved in the resolution sequence corresponding to that proof. It seems questionable if such techniques can be made to work reliably and efficiently enough for use in the context of the heuristic computations we consider here.[8]

In our first version of Conformant-FF, we designed and implemented a heuristic function from the other end of the possible spectrum, with a worst-case polynomial computation. The relaxation that we made on top of ignoring the delete lists was to consider only a 2-CNF-projection of the relevant CNF formulas. That is, we considered a sequence of CNF formulas as above with $1, 2, \ldots$ parallel action applications, selected only two literals out of each clause, and stopped—with success—when the 2-projected CNF implied the goals, or—with failure—when a simple termination criterion told us that no success would be possible in future iterations.[9] In the case of success, from the reasoning done to prove the implication in the 2-CNF setting a relaxed plan could easily be extracted.

The worst-case polynomial computation was made possible because the whole CNF-semantics of the task, *including* the initial state formula, was 2-projected. However, it turned out that 2-projection of the initial state formula yielded very uninformative heuristic values in a lot of cases. In many examples, the uncertainty about the initial state lies in that the values of a set of multiple-valued variables—the positions/internal states of a set of objects—are unknown. Modeling such uncertainty with boolean variables, for a multiple-valued variable with $k$ values one gets (a set of binary clauses and) a clause with $k$ literals in the initial state formula. For values of $k$ larger than 2, ignoring all but 2 of the literals in this clause quickly results in very bad approximations to the semantics of the multiple-valued variable.

In our current version of Conformant-FF—i.e. in the version reported about in this paper—we use a relaxation in between the above two alternatives. We do not do any relaxations to the initial state formula, thus paying the price of full SAT checks against that formula in order to have the benefit of taking full account of the information it contains. We do, however, relax the actions in that we ignore the delete lists of their effects, and we also re-use

[7] If the SAT call comes to the conclusion that there is no relaxed plan then it follows that there is no real plan from the search state either, and the state can be skipped.

[8] At the very least, significant implementation challenges would have to be overcome. Also, there would probably be ambiguities in the extraction of the unsatisfiable subsets of the CNFs, i.e., decisions that can be taken arbitrarily and that significantly affect the size of the found subset. Such ambiguities would directly translate into ambiguities—arbitrary significant variations—in the returned heuristic values, which are likely to affect search efficiency.

[9] The termination criterion was the same as we currently use, see below. The implementation of the process made extensive use of information about propositions already known to be true at certain time steps, similarly to our current implementation, see below.

the 2-projection that we previously used for the clauses related to action effects. At a time step $t$, for an action $a$ that is (known to be) applicable at $t$, for an effect $e \in E(a)$ with condition $con(e) = \{c_1, \ldots, c_k\}$, for $p \in add(e)$, the valid implication is $c_1(t) \wedge \cdots \wedge c_k(t) \rightarrow p(t+1)$. Our 2-projection selects only one of the conditions $c_i$ for inclusion in the implication clause, i.e. we get an implication of the form $c_i(t) \rightarrow p(t+1)$ ($c_i$ is selected arbitrarily as the first condition proposition in our internal numbering). Formally, this simplification comes down to a relaxation that ignores, for each action effect, the effect's delete list as well as all but one of the propositions in the effect's condition. Given a set $A$ of actions, we denote by $|_1^+$ any function from $A$ into the set of all actions, such that $|_1^+$ maps each action $a$ to the same action but with empty delete lists and with all but one condition proposition of each effect removed. We call $|_1^+$ a *relaxation function* for $A$. By $A|_1^+$ we denote the action set resulting from applying $|_1^+$ to each action in $A$. For an action sequence $act$, by $act|_1^+$ we denote the sequence of the images of the actions under $|_1^+$. For a conformant planning task $(A, \mathcal{I}, G)$, we call $(A|_1^+, \mathcal{I}, G)$ a *relaxation* of $(A, \mathcal{I}, G)$. If $act|_1^+$ is a plan for $(A|_1^+, \mathcal{I}, G)$, then $act$ is called a *relaxed plan* for $(A, \mathcal{I}, G)$.

The algorithms that we use to solve relaxed tasks are the topic of the next subsection. Put briefly, the advantage of ignoring all but one effect condition (on top of ignoring all delete effects) is that, with this relaxation, the dependencies that the actions induce between propositions (at time steps) come down to binary implications, where the set of all these implications forms a tree. The reasoning needed to find out if a proposition $p$ is known at $t$ can be done by a simple backward chaining over the tree edges that end in $p(t)$, followed by a SAT check to see if the initial state formula implies the disjunction of the reachable tree leafs.

Obviously, our relaxation induces an over-approximation of reachability. If $act$ is a plan for a task $(A, \mathcal{I}, G)$ then, for any relaxation function $|_1^+$ for $A$, $act|_1^+$ is a plan for $(A|_1^+, \mathcal{I}, G)$. Note that this implies that, if there is no plan for a relaxation of a task, then there is no plan for the real task either. So if our reasoning about relaxed tasks is complete relative to the relaxation, and figures out that there is no relaxed plan to a search state, then that state can be safely excluded from the search.

### 4.2. Solving relaxed tasks

Our algorithms for solving relaxed conformant tasks are a natural extension to the algorithms used in FF [6], and are best presented and understood this way. We now first outline FF's computations, then we present their extension to our conformant setting/relaxation. We prove that the overall algorithm is sound and complete relative to the relaxation.

FF's method of solving relaxed tasks is a specialized version of the Graphplan algorithm [15]. Starting from a world state $w$, build a *relaxed planning graph* as a sequence of alternating proposition layers $P(t)$ and action layers $A(t)$, where $P(0)$ is the same as $w$, $A(t)$ is the set of all actions whose preconditions are contained in $P(t)$, and $P(t+1)$ is $P(t)$ plus the add effects (with fulfilled conditions) of the actions in $A(t)$. From a proposition layer $P(m)$ in which the goals are contained one can find a relaxed plan by a simple backchaining loop. One selects achieving actions at layers $t < m$ for all goals in $P(m)$, one inserts those actions' preconditions and the respective effect conditions as new subgoals (which by construction are at layers below the respective actions), then one steps backwards and selects 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 fix point $P(t) = P(t+1)$ without reaching the goals; $h(w)$ is then set to $\infty$ (excluding the state from the search space).

In the conformant setting, we take the sets $P(t)$ and $A(t)$ to contain the propositions that are known to hold at $t$ and the actions that are known to be applicable at $t$, respectively. Additionally we introduce sets $uP(t)$ of propositions that are unknown at step $t$, i.e. propositions that are true when starting from some initial world states, but that are false when starting from others. We introduce machinery that decides if a proposition $p$ is known at $t$. The machinery maintains a set *Imp* of implications between propositions at adjacent time steps. A variation of the machinery determines, during relaxed plan extraction, what actions must be inserted into the relaxed plan in order to make $p$ known at $t$, if that is required.

The process is additionally complicated by the fact that, in difference to the classical setting where all relevant information is contained in the world state $w$ in question, in our search space representation only *partial* knowledge about the belief state is available. All we have in memory are the sets of known and unknown propositions in the belief state. These do not tell us anything about any *constraints* between the unknown propositions that might follow from the actions we executed on the path to the state. If we ignore such constraints arising from the past, then the relaxed planning algorithm becomes incomplete. Consider, for example, a goal proposition $g$ that can be made true

by an action $a_p$, with a conditional effect that achieves $g$ given we have $p$ (whose value is initially unknown). If we are in the belief state $s$ where only $a_p$ was applied, then there is a constraint $\neg p \vee g$ in our unknown propositions. Ignoring this constraint, a relaxed plan treats $s$ exactly like the initial state, and (potentially) needs to make use of $a_p$ although that is unnecessary in $s$. For one thing, this makes the resulting heuristic function lose distinctions between states, since different belief states are treated equally. Even more importantly, a predecessor action might no longer be applicable, which may make us lose the *ability* to solve the problem, even in its relaxed form (example: $a_p$ has as precondition, and deletes, some proposition that cannot be re-achieved).

Due to these observations, we need to let the relaxed planning algorithm reason about what constraints follow from the actions in *act*. In our approach, in a similar fashion to how we compute known propositions, this reasoning is done by keeping track of the conditional effects that may occur along the execution of *act*, depending on the initial world state. The reasoning is based on the same set *Imp* of (timed) implications that we maintain to determine known propositions at later layers of the relaxed planning graph. Thus the reasoning is easiest understood as additional relaxed planning graph layers referring to the past. The layers are indexed from $-n, \ldots, -1$, corresponding to the actions in *act*, $n$ being the length of *act*. See the conformant relaxed planning graph (*CRPG*) building algorithm in Fig. 2.

We first explain Fig. 2, then we consider an example, then we state that the CRPG algorithm is complete, then we show how one can extract a relaxed plan from a successfully built CRPG.

---

**procedure** build-CRPG($act, A, \mathcal{I}, G, |_1^+$),
  returns a Bool saying if there is a relaxed plan for the belief state
    given by $act = \langle a_{-n}, \ldots, a_{-1} \rangle$, and
  builds data structures from which a relaxed plan can be extracted
$Imp := \emptyset$, $P(-n) := \{p \mid p \text{ is known in } \mathcal{I}\}$, $uP(-n) := \{p \mid p \text{ is unknown in } \mathcal{I}\}$
**for** $t := -n \ldots -1$ **do**
  $A(t) := \{a_t|_1^+, NOOP\}$
  build-timestep($t$, $A(t)$)
**endfor**
$t := 0$
**while** $G \not\subseteq P(t)$ **do**
  $A(t) := \{a|_1^+ \mid a \in A,\ pre(a) \subseteq P(t)\} \cup \{NOOP\}$
  build-timestep($t$, $A(t)$)
  **if** $P(t+1) = P(t)$ and
    $uP(t+1) = uP(t)$ and
    $\forall p \in uP(t+1) : Impleafs(p(t+1)) = Impleafs(p(t))$ **then**
    **return** FALSE
  **endif**
  $t := t + 1$
**endwhile**
$m := t$, **return** TRUE

**procedure** build-timestep($t$, $A$),
  builds $P(t+1)$, $uP(t+1)$, and the implication edges from $t$ to $t+1$,
  as induced by the action set $A$
**for** all effects $e$ of an action $a \in A$ **do**
  **if** $con(e) \in P(t)$ **then** $P(t+1) := P(t+1) \cup add(e)$ **endif**
  **if** $con(e) \in uP(t)$ **then**
    $uP(t+1) := uP(t+1) \cup add(e)$
    $Imp := Imp \cup \{(con(e)(t), p(t+1)) \mid p \in add(e)\}$
  **endif**
**endfor**
**for** all $p \in uP(t+1) \setminus P(t+1)$ **do**
  **if** $\mathcal{I} \rightarrow \bigvee_{l \in Impleafs(p(t+1))} l$ **then** $P(t+1) := P(t+1) \cup \{p\}$ **endif**
**endfor**
$uP(t+1) \setminus = P(t+1)$

---

Fig. 2. Building a conformant relaxed planning graph (CRPG).

Let us just read Fig. 2 from top to bottom. The CRPG is initialized by setting the implication set *Imp* to ∅, and setting the initial proposition layer $P(-n)$ and $uP(-n)$ to the information provided by $\mathcal{I}$ (we have this information from our reasoning about known propositions). The **for** loop then builds the time steps $-n, \ldots, -1$ with action layers corresponding to the actions in *act*, relaxed with $|_1^+$. We additionally introduce a *NOOP* action, to simplify the presentation. The *NOOP* simply transports all propositions from layer $t$ to layer $t+1$. More formally, $pre(NOOP) = \emptyset$ and

$$E(NOOP) = \big\{(\{p\}, \{p\}, \emptyset) \mid p \text{ is a proposition in the task at hand}\big\}.$$

The **while** loop in Fig. 2 builds time steps $t$ from 0 on until a proposition layer $m$ is reached that contains the goals $G$. The actions in each layer $t$ are those whose preconditions are reached at $t$ (plus the *NOOP*), relaxed with $|_1^+$. The algorithm terminates with return value FALSE if a termination criterion is met. We focus on the termination criterion further below. Right now let us consider how the time steps are built. The first **for** loop proceeds over all effects of the actions in the given set. Effects whose condition is known to hold at $t$ yield (potentially) new known propositions at $t+1$. Effects whose condition can hold depending on the initial world state yield (potentially) new such propositions at $t+1$, together with the appropriate implication edges in *Imp*. Note that, to ease the presentation, we assume here that there are no unconditional effects, i.e. no effects $e$ for which $con(e) = \emptyset$. This is no restriction: for such effects, one can simply introduce a new proposition $p$ that is always true, and set $con(e) := \{p\}$. With this we can assume that all effect conditions (of the relaxed actions) contain *exactly* one proposition. To emphasize this, in the notation we use the set $con(e)$ as if it was a single proposition. The second **for** loop of the build-timestep procedure goes over all $p$ that may be true at $t+1$, but that were not yet proved to always be true at $t+1$. It is checked if the initial state formula implies the disjunction of the leafs of the *Imp* tree that are reachable from $p(t+1)$: $Impleafs(p(t'))$, for any $t'$, is defined as

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

In our implementation, such leaf sets are computed by a simple backchaining loop over the edges ending in $p(t')$. The implication $\mathcal{I} \rightarrow \bigvee_{l \in Impleafs(p(t+1))} l$ is checked by a call to the SAT solver, and if the implication holds, $p$ is inserted into the known propositions at $t+1$. As a last step, the procedure removes from the set of unknown propositions, i.e., from those propositions for which it has been shown that there is at least one initial world state that makes them become true, all $p$ that were proved to be always true (i.e., known) at $t+1$.

To illustrate the CRPG algorithm, let us reconsider the example from Section 3.

**Example 2.** A robot is initially at one out of two locations, modeled as $\mathcal{I} = \{at\text{-}L_1 \vee at\text{-}L_2, \neg at\text{-}L_1 \vee \neg at\text{-}L_2\}$. Our goal is to be at $L_2$, and we have a *move-right* action that has an empty precondition, and the conditional effect $(\{at\text{-}L_1\}, \{at\text{-}L_2\}, \{at\text{-}L_1\})$. The CRPG to the initial state—i.e. the CRPG to the empty sequence $act = \langle\rangle$—is computed as follows. $P(0)$ is set to ∅, $uP(0)$ is set to $\{at\text{-}L_1, at\text{-}L_2\}$. In the first iteration, $t = 0$, of the **while** loop, build-timestep is called with $A(0) = \{move\text{-}right, NOOP\}$. In the first **for** loop of build-timestep, the *move-right* action yields the implication edge $(at\text{-}L_1(0), at\text{-}L_2(1))$, while the *NOOP* yields the edges $(at\text{-}L_1(0), at\text{-}L_1(1))$ and $(at\text{-}L_2(0), at\text{-}L_2(1))$. In the second **for** loop, we find that $Impleafs(at\text{-}L_1(1)) = \{at\text{-}L_1(0)\}$ which is not implied by $\mathcal{I}$; however, $Impleafs(at\text{-}L_2(1)) = \{at\text{-}L_1(0), at\text{-}L_2(0)\}$ whose disjunction *is* obviously implied by $\mathcal{I}$. Consequently, $at\text{-}L_2$ is moved into $P(1)$ and we end up with $P(1) = \{at\text{-}L_2\}$, $uP(1) = \{at\text{-}L_1\}$. So in the next iteration, $t = 1$, of the **while** loop, the algorithm terminates with success and returns TRUE. Now, say we call the CRPG algorithm on the belief state given by the action sequence $act = \langle move\text{-}right\rangle$. In this case the whole above process is moved one step "to the left". We obtain $P(-1) = \emptyset$, and $uP(-1) = \{at\text{-}L_1, at\text{-}L_2\}$. $A(-1)$ is set to $\{move\text{-}right, NOOP\}$, which leads to $P(0) = \{at\text{-}L_2\}$ and $uP(0) = \{at\text{-}L_1\}$ just like above. As we will explain below, actions for inclusion in the relaxed plan to the state are only selected for goals at layers $t > 0$, so with the goal $at\text{-}L_2$ being contained in $P(0)$ our relaxed plan will be empty. (In difference to the initial state, where $at\text{-}L_2$ is first contained in $P(1)$ and will lead to the selection of *move-right* into the relaxed plan.)

The reasoning described above is adequate for our relaxation (more precisely, Lemma 1 in Appendix A holds) because *Imp* captures all dependencies introduced by the relaxed actions between propositions at time steps. Note that the *NOOP* action introduces implication edges of the form $(p(t), p(t+1))$ saying that a proposition $p$ will still be true at $t+1$ if it was true at $t$ (remember that there are no delete lists). The termination criterion of the CRPG

computation asks if, from time step $t$ to time step $t+1$, neither the known propositions nor the unknown propositions nor the reachable *Imp* tree leafs of the latter propositions have changed. It is relatively easy to see that, in this case, the same would hold true at all future time steps $t' > t$, implying that the goals will never be reached.

More formally, the following holds.

**Theorem 2** *(CRPG-completeness). Given a conformant task $(A, \mathcal{I}, G)$, an executable action sequence act, and a relaxation function $|_1^+$ for A. If* build-CRPG$(act, A, \mathcal{I}, G, |_1^+)$ *returns FALSE then there is no relaxed plan for $(A, \mathcal{I}, G)$ that starts with the sequence act.*

**Proof sketch.** The proof is based on two observations. First, the propositions in the sets $P(t)$ (respectively $uP(t)$) are exactly those propositions that are known (respectively unknown) after executing all the actions in the action layers up to $A(t-1)$. Second, if the CRPG algorithm returns FALSE in some iteration then the termination criterion would also hold in all future iterations. With the first observation, if there is an $m$-step relaxed plan for $(A, \mathcal{I}, G)$ that starts with *act*, then the goals are contained in $P(m)$. With the second observation, if the CRPG algorithm returns FALSE then no such layer $m$ exists. □

If the CRPG building algorithm succeeds in reaching the goals, then a relaxed plan is extracted and the length of that relaxed plan is used to generate the heuristic value of the belief state—see below. Relaxed plan extraction proceeds as specified in Fig. 3.

We explain Fig. 3, then we reconsider the example, then we state that the CRPG algorithm is sound, then we explain how the heuristic value to the state is generated.

The process makes use of proposition sets $G(1), \ldots, G(m)$, which are used to store the goals and sub-goals arising at layers $1 \leqslant t \leqslant m$ during relaxed plan extraction. The sub-goal procedure simply inserts all given propositions as sub-goals at their first layer of appearance in the CRPG. The sub-goals are considered layer-by-layer, starting from

```
procedure extract-CRPlan(CRPG(act, A, I, G, |_1^+), G),
          selects actions from A(0), ..., A(m − 1) that
          form a plan for (A|_1^+, I, G) when appended to act|_1^+
sub-goal(G)
for t := m, ..., 1 do
  for all g ∈ G(t) do
    if ∃a ∈ A(t − 1), e ∈ E(a), con(e) ∈ P(t − 1), g ∈ add(e) then
      select one such a /* and e */ at t − 1
      sub-goal(pre(a) ∪ con(e))
    else
      let L be a minimal subset of Impleafs(g(t)) s.t. I → ⋁_{l∈L} l
      for all i ⩾ 0, actions a ∈ A(i) \ {NOOP}, and effects e ∈ E(a) s.t.
          e is responsible for an edge in a path from l(−n), l ∈ L, to g(t) do
          select a at i
          sub-goal(pre(a))
      endfor
    endif
  endfor
endfor

procedure sub-goal(P),
          inserts the propositions in P as sub-goals
          at the layers of their first appearance in the CRPG
for all p ∈ P do
  let t_0 be the smallest t s.t. p ∈ P(t)
  if t_0 ⩾ 1 then G(t_0) := G(t_0) ∪ {p} endif
endfor
```

Fig. 3. Extracting a conformant relaxed plan.

the top layer $m$ going downwards to layer 1. For each sub-goal $g$ at layer $t$, supporting actions are selected into the relaxed plan. If there is an action $a$ at layer $t-1$ that guarantees to always achieve $g$, then $a$ is selected at $t-1$. Otherwise, a minimal subset $L$ of $\mathit{Impleafs}(g(t))$ is determined such that $\mathcal{I} \rightarrow \bigvee_{l \in L} l$. This is done by starting from $L := \mathit{Impleafs}(g(t))$ and, iteratively, removing propositions $p$ from $L$ if $\bigvee_{p \neq l \in L} l$ is still implied by $\mathcal{I}$, until no such $p$ is left in $L$. Then all actions are selected, at the respective times, that are responsible for the implication paths from $L$ at $-n$ to $g$ at $t$. The motivation behind the use of a minimal implied subset $L$ is that we want to avoid the inclusion of superfluous actions into the relaxed plan. Such superfluous actions lead to an unnecessary over-estimation of the real goal distance, and can seriously affect the informativity of the resulting heuristic function.

Selection of actions is to be understood as a set union operation, i.e. if an action $a$ is selected that was selected at the same time before, then nothing happens. Note that, in the first case of the **if** statement, when an action guarantees to always achieve $g$, then that action cannot be the *NOOP* because otherwise $g$ would be contained in $P(t-1)$ already.

We reconsider the illustrative example above.

**Example 3.** A robot is initially at one out of two locations, $\mathcal{I} = \{\mathit{at}\text{-}L_1 \vee \mathit{at}\text{-}L_2, \neg \mathit{at}\text{-}L_1 \vee \neg \mathit{at}\text{-}L_2\}$. The goal is to be at $L_2$, and the *move-right* action has an empty precondition, and the conditional effect $(\{\mathit{at}\text{-}L_1\}, \{\mathit{at}\text{-}L_2\}, \{\mathit{at}\text{-}L_1\})$. To the initial state, we obtain the CRPG $P(0) = \emptyset, uP(0) = \{\mathit{at}\text{-}L_1, \mathit{at}\text{-}L_2\}$, $P(1) = \{\mathit{at}\text{-}L_2\}, uP(1) = \{\mathit{at}\text{-}L_1\}$, and $\mathit{Imp} = \{(\mathit{at}\text{-}L_1(0), \mathit{at}\text{-}L_2(1)), (\mathit{at}\text{-}L_1(0), \mathit{at}\text{-}L_1(1)), (\mathit{at}\text{-}L_2(0), \mathit{at}\text{-}L_2(1))\}$. Relaxed plan extraction starts from layer $m = 1$, with $G(1) = \{\mathit{at}\text{-}L_2\}$. There is no action that guarantees achievement of $\mathit{at}\text{-}L_2$, and we get $L = \mathit{Impleafs}(\mathit{at}\text{-}L_2(1)) = \{\mathit{at}\text{-}L_1(0), \mathit{at}\text{-}L_2(0)\}$. The edges from $L$ at 0 to $\mathit{at}\text{-}L_2$ at 1 are $(\mathit{at}\text{-}L_1(0), \mathit{at}\text{-}L_2(1))$ and $(\mathit{at}\text{-}L_2(0), \mathit{at}\text{-}L_2(1))$. The former edge is induced by *move-right* at 0, the latter edge is induced by *NOOP* at 0. So (only) *move-right* at 0 is selected into the relaxed plan. In the CRPG built to $\mathit{act} = \langle \mathit{move\text{-}right} \rangle$, as explained above we have $\mathit{at}\text{-}L_2 \in P(0)$. So we get $m = 0$ and relaxed plan extraction does not perform any sub-goal achievement iterations, thus delivering an empty relaxed plan.

It is relatively easy to see that the actions selected by extract-CRPlan form a relaxed plan for the belief state given by $\mathit{act}$.

**Theorem 3** *(CRPG-soundness). Given a conformant task $(A, \mathcal{I}, G)$, an executable action sequence act, and a relaxation function $|_1^+$ for A, s.t.* build-CRPG$(\mathit{act}, A, \mathcal{I}, G, |_1^+)$ *returns TRUE. Let $A(0)^s, \ldots, A(m-1)^s$ be the actions selected from $A(0), \ldots, A(m-1)$ by* extract-CRPlan$(\mathit{CRPG}(\mathit{act}, A, \mathcal{I}, G, |_1^+), G)$. *Then $\mathit{act}|_1^+$ concatenated with $A(0)^s, \ldots, A(m-1)^s$ in an arbitrary linearization is a plan for $(A|_1^+, \mathcal{I}, G)$.*

**Proof sketch.** It can be shown by induction over $t$ that, for any proposition $g$ that is made a sub-goal at time $t$ $(0 < t \leqslant m)$ during the execution of extract-CRPlan, $g$ is known after executing the actions in $\mathit{act}|_1^+$ and $A(0)^s, \ldots, A(t-1)^s$. This shows the claim. The main argument in the induction is that, if actions were selected for all effects that were responsible for an edge in a path from $l(-n)$, $l \in L$, to $g(t)$, where $\mathcal{I} \rightarrow \bigvee_{l \in L} l$, then in every $I \in 2^{\mathcal{I}}$ one $l \in L$ is true, and the actions on the path from $l(-n)$ to $g(t)$ make $g$ true at $t$. □

Theorems 2 and 3 together imply that build-CRPG$(\mathit{act}, A, \mathcal{I}, G, |_1^+)$ returns TRUE if and only if there is a relaxed plan for $(A, \mathcal{I}, G)$ that starts with $\mathit{act}$.

If the CRPG returns FALSE, we set our heuristic value $h$ to $\infty$: the state is proved to be a *dead end*, i.e. a state from which the goals can't be reached (this holds because, as pointed out above, $|_1^+$ induces an over-approximation of reachability). The state will be excluded from the search space. If the CRPG succeeds in reaching the goals, let $h$ be the number of actions selected by extract-CRPlan$(\mathit{CRPG}(\mathit{act}, A, \mathcal{I}, G, |_1^+), G)$, i.e., the number of actions in the relaxed plan to the belief state. As in the classical setting of FF, our intention is to use $h$ as the heuristic value for the state. There is, however, one subtlety here in the conformant setting. We relax the semantics of not only the actions in the CRPG layers at times $0, \ldots, m$, but also of the actions at times $-n, \ldots, -1$, i.e., of *the actions on the path to the state*. Because of this, the CRPG is an over-approximation not only of what we will be able to achieve in the future, but also of what we already achieved in the past. In consequence, it may be that we obtain an empty relaxed plan—the goals are all reached at $P(0)$ already—although the goals are *not* known in the state. So our heuristic value may be 0 in non-goal states. Since normally a heuristic value of 0 is taken to indicate goal states, we add 1 to $h$ in case the goals are not known after *act* (which we know about from our reasoning about the search states). Note that the

phenomenon—over-approximation of what was achieved in the past—may seriously affect the informativity of the heuristic function in regions of the search space where many actions were applied already. We explain below, amongst other things, a variation of the heuristic function with which we have tried to overcome this problem.

Altogether, the above processes define a heuristic $h$ as a function from belief states into $\mathbf{N}_0 \cup \{\infty\}$, i.e. into the natural numbers with 0 and $\{\infty\}$. States $s$ with $h(s) = 0$ are goal states, states with $h(s) = \infty$ are proved unsolvable and can be skipped.

### 4.3. Optimizations

One can spare some time in the creation of the CRPG, and obtain a somewhat more informative heuristic function, by re-using the knowledge stored along the search path to the state, i.e., along the action sequence *act*, when building the CRPG layers $-n, \ldots, -1$. Instead of computing the proposition layers $P(-n), uP(-n), \ldots, P(-1), uP(-1)$ by the same machinery as used for the higher CRPG layers, one can simply set these proposition layers to the values stored in the search states along *act*, as were computed during the search. The *Imp* edges can be inserted accordingly. This obviously saves some time, and also it ameliorates the problem outlined above, over-estimation of the past achievements. Note that, however, the reasoning done on the *Imp* tree still over-estimates the semantics of the actions in *act*. One of the variations described below avoids this problem altogether, at the cost of more expensive SAT reasoning.

There is no need to decide off-line, for each effect, exactly which of the condition propositions one ignores. Indeed, in some of the computations made when building the CRPG, it is not necessary to ignore all but one of the effect conditions. One can avoid applying effects that are known to not occur, and one can avoid ignoring the known aspects of an effect condition. In our implementation, say we build a time step $t$ and, during the first **for** loop of build-timestep, consider an action effect $e$. We look at the original form of $e$, with the whole effect condition. If $con(e) \subseteq P(t)$, the effect condition is known to hold and all $p \in add(e)$ are inserted into $P(t+1)$. If $con(e) \not\subseteq P(t) \cup uP(t)$, the effect is known to not occur, and we skip it. If the above two cases do not hold, i.e. we have $con(e) \not\subseteq P(t)$ and $con(e) \subseteq P(t) \cup uP(t)$, then the effect *possibly* occurs. We then (arbitrarily) select one $c \in con(e)$ such that $c \in uP(t)$, and insert all $p \in add(e)$ into $uP(t+1)$, with edges $(c(t), p(t+1))$.

### 4.4. Variations

Throughout the rest of the article, we refer to the $h$ values delivered by the above machinery, including the optimizations, as the *initial-formula* heuristic function, denoted with $h^{\mathcal{I}}$ to indicate that it is obtained by SAT checks against $\mathcal{I}$. We also call $h^{\mathcal{I}}$ the *standard* heuristic function because we use it as the default in our implemented system.

In the first variation of the heuristic, called the *incomplete* heuristic function and denoted as $h^{inc}$, we do not do any SAT calls to check the implications $\mathcal{I} \rightarrow \bigvee_{l \in L} l$. We only do a simple sufficient test. The motivating observation is that, often, in our examples, when $\mathcal{I}$ implied $\bigvee_{l \in Impleafs(p(t))} l$, the reason was simply a clause in $\mathcal{I}$ that was a subset of $Impleafs(p(t))$. (Note that this is also the case for $Impleafs(at\text{-}L_2(1)) = \{at\text{-}L_1(0), at\text{-}L_2(0)\}$ in our illustrative example.) So our sufficient condition only sees if there is a clause $C$ in $\mathcal{I}$ such that $C \subseteq L$. If that is not the case, then we assume that the implication does not hold. This can, obviously, help to save runtime. Likewise obviously, the resulting CRPG is no longer complete relative to the relaxation, i.e., the CRPG can return FALSE even if there is a relaxed plan. So, if the CRPG cannot reach the goals, to preserve the overall completeness of the search algorithm, $h^{inc}$ returns some large integer value instead of $\infty$.

Our second variation of $h^{\mathcal{I}}$ is called the *state-formula* heuristic function, and denoted with $h^{\phi(act)}$. Recall that $h^{\mathcal{I}}$ relaxes the semantics of all actions including those in *act*, and checks *Imp* leaf disjunctions for implication by $\mathcal{I}$. In contrast, the relaxed planning algorithm behind $h^{\phi(act)}$ only relaxes the semantics of the actions in the "future" layers of the CRPG, $t \geqslant 0$, and checks *Imp* leaf disjunctions for implication by *the formula* $\phi(act)$ *that encodes the semantics of the belief state given by act*. Indeed, the CRPG now only contains layers $t$ with $t \geqslant 0$. Precisely, the variation is obtained from Fig. 2 as follows. Skip the first **for** loop. Set $P(0)/uP(0)$ to the known/unknown propositions after *act*, and start with the **while** loop. Replace, in build-timestep, the condition $\mathcal{I} \rightarrow \bigvee_{l \in Impleafs(p(t+1))} l$ with $\phi(act) \rightarrow \bigvee_{l \in Impleafs(p(t+1))} l(n)$. Otherwise, the algorithm remains unchanged. In the relaxed plan extraction algorithm, Fig. 3, the only change made is that now $L$ is defined to be a minimal subset of $Impleafs(g(t))$ such that $\phi(act)$, rather than $\mathcal{I}$, implies $\bigvee_{l \in L} l$.

The $h^{\phi(act)}$ approach has the advantage that reasoning about the past is precise, with no over-estimation of what was achieved by *act*. In particular, if the returned relaxed plan is empty, then this means that $s$ is a goal state. The disadvantage of $h^{\phi(act)}$ is that checking implication against the formulas $\phi(act)$ is typically much more costly than checking implication against $\mathcal{I}$. The latter formula is the same for all search states. The former formula grows with the length of *act*, which are up to a few hundred steps in some of our examples. Note that the checks for $\phi(act) \rightarrow \bigvee_{l \in Impleafs(p(t+1))} l(n)$ need to be done every time when the CRPG has to test if an unknown proposition becomes known at a layer, and every time the relaxed plan selects such a proposition as a sub-goal. (In the latter case, even several implication checks are needed in order to identify a minimal implied subset.)

We remark that, in our experiments, often $h^{\phi(act)}$ and $h^{inc}$ yielded performance very similar to that of $h^{\mathcal{I}}$. Details are in Section 6.

## 5. 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. (For example, repeated search states occur 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 equivalence test (or a belief state domination test, see below). When we perform search, we check via a hash table lookup whether the new belief state is equivalent to (dominated by) some previous belief state in the same hash entry. If so, the new belief state is pruned. The hash value of a state is a function of the union of the propositions that are known or unknown in the state, which ensures that equivalent belief states end up in the same hash entry (state dominations may be missed due to different hash values).

In the classical setting, testing equality of world states is straightforward and easy: the states are proposition sets. In the conformant setting, one must test if two belief states, i.e., two *sets* of world states, are equal. We did not find a way to do this based on our sparse representation of belief states. Instead, we test a stronger, sufficient but not necessary, criterion which we call belief state *equivalence*. Let $s$ and $s'$ be two belief states reached via the action sequences *act* and *act*$'$, respectively. We say that $s$ is equivalent to $s'$ iff for every possible initial world state $I \in 2^{\mathcal{I}}$ the world state upon execution of *act* in $I$ is equal to the world state upon execution of *act*$'$ in $I$. Note that $s$ and $s'$ may be equal but not equivalent because the "ordering" of their world states—the correspondence to the initial world states—may differ even though, as sets of world states, $s$ and $s'$ are equal. However, if $s$ and $s'$ are equivalent then obviously they are also equal.

The equivalence relation can be tested based on satisfiability checks very similar to those that we do for computing the known propositions. The key observation is the following. Two belief states $s$ and $s'$ are equivalent iff for all propositions $p$ there is no $I \in 2^{\mathcal{I}}$ starting from which $p$ has different truth values in (the respective world states in) $s$ and $s'$. From left to right, if $s$ and $s'$ are equivalent then obviously all $p$ show the same behavior from all $I$. From right to left, if there was an $I$ that yielded different world states $w$ and $w'$ in $s$ and $s'$, then at least one $p$ would have different values in $w$ and $w'$, in contradiction.

With the above, we can use the following satisfiability tests to decide about belief state equivalence. Let $\phi_{I=}(act)$ and $\phi_{I=}(act')$ be the formulas constructed for the action sequences *act* and *act*$'$, such that $\phi_{I=}(act)$ and $\phi_{I=}(act')$ 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 *act* and *act*$'$ in the *same* initial world state). For a proposition $p$, let $p(act)$ and $p(act')$ denote $p$'s value (the respective propositional variable) following *act* and *act*$'$, respectively. Then $s$ is equivalent to $s'$ iff the formulas $\phi_{I=}(act) \wedge \phi_{I=}(act') \wedge p(act) \wedge \neg p(act')$ and $\phi_{I=}(act) \wedge \phi_{I=}(act') \wedge \neg p(act) \wedge p(act')$ are unsatisfiable for all $p$.[10]

One can, of course, optimize the equivalence test by making use of the knowledge that was already computed for $s$ and $s'$. State equivalence can only hold if $s$ and $s'$ agree on the sets of known/negatively known/unknown

[10] Note that this set of SAT tests is basically a decomposed form of the non-CNF entailment test $\phi_{I=}(act) \wedge \phi_{I=}(act') \wedge \neg \bigwedge_p [(\neg p(act) \vee p(act')) \wedge (p(act) \vee \neg p(act'))]$, i.e., the test that asks if all $p$ behave equivalently at the ends of *act* and *act*$'$. The decomposition consists of putting $\neg \bigwedge_p [(\neg p(act) \vee p(act')) \wedge (p(act) \vee \neg p(act'))]$ into negation normal form, and proving unsatisfiability for each individual member of the resulting DNF.

propositions—which, in our code, is equivalent to $s$ and $s'$ falling into the same entry of the hash table. The SAT calls need only be made for those $p$ that are unknown (in both $s$ and $s'$).

In many domains, including all but one of our experimental domains, a stronger form of repeated state pruning is valid. In these domains it is the case that it can only be better to have more propositions true. More formally, we say that a world state $w$ *dominates* a world state $w'$ if $w \supseteq w'$. In many domains, if $w$ dominates $w'$, then every action sequence that solves the goal starting from $w'$ also solves the goal when starting from $w$. Then, if in a classical search one has already seen $w$, and $w'$ comes up as a new state, $w'$ can be skipped without losing completeness. In the conformant setting, we say that a belief state $s$ dominates a belief state $s'$ if, for every initial world state $I \in 2^{\mathcal{I}}$, the corresponding world state in $s$ (the state after executing $act$) dominates the corresponding world state in $s'$ (the state after executing $act'$). If it is always better to have more propositions true, and $s$ dominates $s'$, then every conformant plan from $s'$ is also a conformant plan from $s$. So if one has already seen $s$ then $s'$ can be skipped. Belief state domination can be tested by the exact same technique as explained for the equivalence test above, except that one needs only a single SAT call per proposition, namely that for $\phi_{I=}(act) \wedge \phi_{I=}(act') \wedge \neg p(act) \wedge p(act')$. It is easy to see that $s$ dominates $s'$ iff this formula is unsatisfiable for all $p$.[11] Obviously, one can skip some SAT calls by observing that $s$ can only dominate $s'$ if its known (known or unknown) propositions are a superset of those known (unknown) in $s'$. SAT calls need only be made for those $p$ unknown in both $s$ and $s'$.

In comparison to the equivalence test, the domination test has more pruning potential on top of being cheaper in terms of the number of SAT calls made. But in our conformant planning setting, it is not generally the case that it can only be better to have more propositions true. The reason lies in the conditional effects (in pure STRIPS, obviously having more propositions true can only be better). If $w \supseteq w'$, and $p \in w \setminus w'$, then there may be a conditional effect that has $p$ among its conditions, and that deletes important propositions (that would not be deleted when executing the same action in $w'$). However, there is a simple sufficient condition implying that the latter cannot happen. Say a conditional effect $e$ is of the form $(\{c\}, add, (del \cap SREL) \subseteq \{c\})$. Here $SREL$ denotes the set of all propositions that occur either in the goal or in some action precondition or in some effect condition. That is, $e$ has only a single condition $c$ and a delete list whose only delete that may be needed is $c$. Then if $e$ occurs from $w$ but not from $w'$, the only important proposition that may be deleted is an "additional" one, i.e. one in $w \setminus w'$. With this, it is easy to see that, if all effects are of the form $(\{c\}, add, (del \cap SREL) \subseteq \{c\})$, then with $w \supseteq w'$ every plan from $w'$ is also a plan from $w$. It is then valid to prune a belief state $s'$ if one has already seen a belief state $s$ that dominates $s'$.

In our implementation of Conformant-FF, before search starts we (automatically) check if all conditional effects have the form explained above. If so, the repeated states checking uses the stronger belief state domination test. Otherwise, the full equivalence test is used. In all our experimental domains, with a single exception (the Omelette domain, namely), all conditional effects have the necessary form, and the domination test is used.

In a few cases in our experiments we found that repeated states checking incurred a huge overhead, rendering instances prohibitively hard that were otherwise—when turning the repeated states check off—easy to solve. This phenomenon occurs in search spaces where no or only very few propositions become known/negatively known before a plan is found. In such a situation, almost every visited state ends up in the same hash entry, since the hashing function has no means to distinguish between them. Every time a new state is generated, it is compared to (almost) every state visited before. So, in total, the number of comparisons made is a square function in the number of explored search states. Every single comparison can involve several SAT calls, and so, clearly, checking for repeated states becomes all-consuming.

To overcome the above deficiency, we have also implemented a weaker variant of repeated states checking, that we call *checking for stagnating states*. There, a new search state $s'$ is only compared to the states $s$ visited *on the path to* $s'$. The total number of state comparisons is then bounded by the number of explored states multiplied with the depth of the search tree (rather than the size of that tree, as before). The drawback is, clearly, a loss of pruning potential.

The default setting in our implementation is a full repeated states check. Most of the time, at least as reflected by our experiments, this behaves better than, or equally good as, the limited stagnation check. An alternative way to overcome the potential weakness of full repeated states checking may be to use fast pre-checks to see if the SAT calls can be avoided. We discuss this in the outlook in Section 9.

[11] Similarly to above, this set of SAT calls is a decomposed version of the test $\phi_{I=}(act) \wedge \phi_{I=}(act') \wedge \neg \bigwedge_p [p(act) \vee \neg p(act')]$.

## 6. Results

We implemented the techniques described in the previous sections in C. We evaluated the resulting system, Conformant-FF, on a collection of benchmarks including traditional conformant benchmarks, and classical benchmarks enriched with uncertainty. We compared Conformant-FF's performance in detail to that of MBP and KACMBP. We also ran some comparative tests with GPT and with POND.

In the next subsection, we give some implementation details of Conformant-FF, thereafter a subsection gives details on our experimental setup. Then two subsections give our results for Conformant-FF, MBP, and KACMBP. The first of these subsections considers the traditional conformant benchmarks, the second one considers the enriched classical benchmarks. A final subsection describes the results we obtained with GPT and POND.

### 6.1. Implementation

We implemented the Conformant-FF system in C, starting from the FF code.[12] Conformant-FF'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 for an implication path during relaxed plan extraction.

Conformant-FF uses a largely naive standard DPLL solver for the CNF reasoning. We also implemented a version using Chaff [16] as the SAT solver. But typically, when running Conformant-FF thousands of CNF formulas have to be considered, each of which is very easy to solve—most of the time, in our experiments, a single run of unit propagation sufficed to find a satisfying assignment or to prove unsatisfiability. In the version using Chaff, communicating all the CNFs to Chaff, and the time taken by Chaff to build its (rather complex) internal data structures, quickly became prohibitive for scaling while the overall time taken to actually solve the CNFs was negligible. So we implemented our own DPLL-based SAT solver within Conformant-FF, working directly on Conformant-FF's internal data structures. The implementation is a naive standard backtracking realized by recursive function calls, using no clever data structures whatsoever except, for each literal, a linked list to the clauses it participates in. The latter lists are used to quickly access the relevant clauses when a literal gets set by unit propagation. We did not implement watched literals because in our experiments typically around 95% of the generated clauses were binary anyway.[13] We did not even implement a variable selection heuristic (i.e., the variables are selected in an arbitrary order) because, as said above, most of the time a single run of unit propagation sufficed. For the same reason, we did not bother at all with clause learning techniques. If not run on examples with much more complex formulas as the initial states, it seems unlikely that any

[12] Available at http://www.mpi-sb.mpg.de/~hoffmann/ff.html.

[13] Note that this is just a phenomenon of the benchmarks used in the experiments; in general, of course the clauses may be arbitrarily large. Our design choice here, or rather our laziness in the implementation, is dependent on practical experience, i.e., on a certain (seemingly typical) range of examples. In particular, the DPLL procedure can *not* be replaced with a 2-CNF reasoner. A promising option may be to employ 2-CNF simplification algorithms, such as e.g. [17].

more fancy SAT solving techniques than what's implemented will yield significant performance improvements for Conformant-FF.

### 6.2. Experimental setup

The experiments were run on a PC running at 1.2 GHz with 1 GB main memory and 1024 KB cache running Linux, with a runtime cutoff of 1200 seconds. The traditional conformant benchmark domains we used were *Bomb-in-the-toilet* (short *Bomb*), *Cube*, *Omelette*, *Ring*, and *Safe* (see also, e.g., [2] and [18]). The enriched classical benchmark domains were *Blocksworld*, *Logistics*, and *Grid*. The individual domains are described below, when we discuss the respective results. For each example instance we provide, for each planner in the experiment, the runtime taken and the length of the found plan; in the case of Conformant-FF, we also provide the number of evaluated states, i.e. the number of states for which a heuristic value was computed. We summarize the results in the text, and, where appropriate and possible, give intuitive explanations of the observed behavior.

We performed a detailed comparison of Conformant-FF with the most recent available versions of MBP and KACMBP. These two systems have been shown to be highly competitive, far more efficient than all other existing systems in many cases. E.g., they are known to outperform GPT in the traditional conformant benchmarks. (KACMBP is a more advanced version of the conformant planner in MBP—see Section 8 for more details.) We run MBP with its default settings, and we run KACMBP with the heuristic forward-search option (which seems to give the overall best performance). These settings are also used in the experimental setups included with the respective downloads.

We also performed comparisons of Conformant-FF with GPT, and with POND. The comparison to GPT was done on fewer examples but suffices to give a relatively good impression of the performance differences. The comparison to POND was done on the same set of examples as those for MBP and KACMBP, but POND often behaved very badly and it is not completely clear what the reason for that is. More details on GPT and POND are below in Section 6.5.

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 the input language of KACMBP, one explicitly specifies each ground action, also with the possibility of using multi-valued variables, which we used as much as possible.

In the experiments, we found that the performance difference between the heuristics $h^{\mathcal{I}}$ and $h^{inc}$ was, most of the time, negligible. Thus, except in one results table where we consider exceptionally large instances, these tables contain the results for $h^{\mathcal{I}}$ and $h^{\phi(act)}$ only. In a single other case (test suite), there was a remarkable difference between the behavior of $h^{\mathcal{I}}$ and $h^{inc}$, which is described in the text. In a similar fashion, we report detailed results only for the default setting of repeated states checking, i.e. for the full test. Checking (only) for stagnating states yields improvements in just a few cases, which are discussed in the text.

### 6.3. Conformant benchmarks

Table 1 provides our experimental results in the Bomb domain. In our variant of this domain, there are $b \geqslant 1$ bombs, each of which is initially potentially armed (i.e., it is unknown if the bomb is armed or not). There are $t \geqslant 1$ toilets. One can "dunk" a bomb into a toilet. The precondition of this is that the toilet is not clogged. As effects, the action disarms the bomb if it is armed, and the toilet gets clogged. One can "flush" a toilet, with no precondition, making it un-clogged. In Table 1, the instances "Bomb-b*b*-t*t*" are the examples with $b$ bombs and $t$ toilets, respectively.

Regarding runtime, we observe the following. We first consider only the default version of Conformant-FF, i.e. the $h^{\mathcal{I}}$ heuristic. MBP and KACMBP are superior to Conformant-FF when there are few toilets. In the extreme case, when there is just a single toilet as in the topmost part of Table 1, Conformant-FF gets outperformed quickly as the number of bombs increases. Then, as the number of toilets rises to 5 and 10 in the next two parts of the table, we observe that Conformant-FF becomes faster while MBP and KACMBP become slower. For MBP, the loss of efficiency is dramatic,

Table 1
Results in the Bomb domain

| Instance | Conformant-FF | | MBP | KACMBP |
|---|---|---|---|---|
| | $h^{\mathcal{I}}$ | $h^{\phi(act)}$ | | |
| | $t/\|S\|/l$ | $t/\|S\|/l$ | $t/l$ | $t/l$ |
| Bomb-b5-t1 | 0.01/19/9 | 0.00/19/9 | 0.01/9 | 0.01/10 |
| Bomb-b10-t1 | 0.03/64/19 | 0.03/64/19 | 0.05/19 | 0.03/20 |
| Bomb-b20-t1 | 0.25/229/39 | 0.53/229/39 | 0.28/39 | 0.07/40 |
| Bomb-b50-t1 | 5.07/1324/99 | 10.98/1324/99 | 2.39/99 | 0.86/100 |
| Bomb-b100-t1 | 129.70/5149/199 | – | 20.09/199 | 2.65/200 |
| Bomb-b10-t5 | 0.02/30/15 | 0.00/30/15 | 0.74/15 | 0.14/20 |
| Bomb-b20-t5 | 0.17/155/35 | 0.32/155/35 | 2.63/35 | 0.38/40 |
| Bomb-b50-t5 | 4.67/1130/95 | 16.98/1130/95 | 31.43/95 | 1.37/100 |
| Bomb-b100-t5 | 120.99/4755/195 | – | 275.74/195 | 4.38/200 |
| Bomb-b5-t10 | 0.00/5/5 | 0.00/5/5 | 1.03/5 | 0.14/10 |
| Bomb-b20-t10 | 0.05/85/30 | 0.04/85/30 | 9.64/30 | 0.77/40 |
| Bomb-b50-t10 | 3.14/910/90 | 12.98/910/90 | 115.81/90 | 2.28/100 |
| Bomb-b100-t10 | 109.65/4285/190 | – | 952.09/190 | 8.69/200 |
| Bomb-b5-b5 | 0.00/5/5 | 0.01/5/5 | 0.20/5 | 0.13/10 |
| Bomb-b10-t10 | 0.00/10/10 | 0.01/10/10 | 2.32/10 | 0.36/20 |
| Bomb-b20-t20 | 0.03/20/20 | 0.03/20/20 | 46.12/20 | 1.59/40 |
| Bomb-b50-t50 | 0.56/50/50 | 0.47/50/50 | – | 46.16/100 |
| Bomb-b100-t100 | 3.13/100/100 | 5.77/100/100 | – | – |

Times $t$ in seconds, search space size $|S|$ (number of evaluated states), plan length $l$. Dashes indicate time-outs.

and with 10 toilets it already performs a lot worse than Conformant-FF. KACMBP remains faster than Conformant-FF, but still its loss of efficiency is significant. In the extreme case displayed in the bottom part of Table 1, there as many toilets as bombs. Under these circumstances, MBP and KACMBP both fail to scale up, while Conformant-FF solves even the task with 100 bombs easily.

Regarding the $h^{\phi(act)}$ version of Conformant-FF, it explores the same number of search states as $h^{\mathcal{I}}$, but it takes considerably more time for doing so. This is due to the time overhead incurred by the more costly SAT tests in the heuristic computation.

Regarding plan length, we observe that Conformant-FF and MBP find the shortest possible plans, making use of additional toilets—if there are $t > 1$ toilets then one can save time by dunking bombs into all of them, and thus sparing some flushing actions when no more bomb will be dunked into a toilet. In difference to this, KACMBP's plans flush the toilet after every dunking action.

The observed runtime behavior is, in fact, relatively easy to explain. As the number of toilets increases, MBP and KACMBP get in trouble due to, presumably, the exponentially increasing number of states in the belief space. For Conformant-FF, however, the more toilets there are the more accurate does its heuristic become. One can observe this in the data when comparing the plan lengths $l$ to the search space sizes $|S|$. In the top part of the table, $|S|$ is a lot larger than $l$. In the bottom part, however, both numbers are equal. The latter means that, to find the plan, the only states heuristically evaluated by Conformant-FF are those along the search path corresponding to the plan! That is, in Conformant-FF's Hill-climbing procedure, every state has a successor (the first one looked at) with a better heuristic value. Namely, if there are (at least) two non-clogged toilets left then dunking a (potentially armed) bomb into one of them yields a state with a one step shorter relaxed plan—that shorter relaxed plan can use the other non-clogged toilet to dispose of all the remaining bombs. If there is just a single non-clogged toilet, then the nearest successor with a shorter relaxed plan is two steps away. This yields the larger search spaces in the top three parts of Table 1.

Table 2 shows our results in our other traditional conformant benchmark domains, i.e. in Cube, Omelette, Ring, and Safe. From a quick glance, one sees that Conformant-FF is generally competitive with MBP, often outperforming it. KACMBP, however, is clearly superior to both other planners in all the domains. We now consider each of the domains in more detail.

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. We created two different test

Table 2
Results in the conformant benchmarks except Bomb

| Instance | Conformant-FF | | MBP | KACMBP |
|---|---|---|---|---|
| | $h^{\mathcal{I}}$ | $h^{\phi(act)}$ | | |
| | $t/\|S\|/l$ | $t/\|S\|/l$ | $t/l$ | $t/l$ |
| Cube-corner-3 | 0.01/6/6 | 0.01/7/7 | 0.02/8 | 0.00/6 |
| Cube-corner-5 | 0.03/12/12 | 0.07/13/13 | 2.66/16 | 0.00/12 |
| Cube-corner-7 | 0.12/18/18 | 0.46/19/19 | – | 0.00/18 |
| Cube-corner-9 | 0.35/24/24 | 1.31/25/25 | – | 0.00/24 |
| Cube-corner-11 | 0.79/30/30 | 2.76/31/31 | – | 0.00/30 |
| Cube-center-3 | 0.06/93/15 | 0.03/61/9 | 0.03/9 | 0.00/11 |
| Cube-center-5 | 16.98/2211/45 | 0.82/262/30 | 2.80/18 | 0.04/20 |
| Cube-center-7 | – | 5.81/825/55 | – | 0.07/31 |
| Cube-center-9 | – | 80.61/2052/97 | – | 0.13/40 |
| Cube-center-11 | – | 1049.74/4913/147 | – | 0.16/46 |
| Omelette-3 | 0.03/79/NS | 0.01/79/NS | 1.38/NS | 0.03/NS |
| Omelette-5 | 0.09/161/NS | 0.04/161/NS | 13.46/NS | 0.08/NS |
| Omelette-10 | 0.73/468/NS | 0.25/468/NS | 507.12/NS | 0.23/NS |
| Omelette-15 | 1.59/917/NS | 0.87/917/NS | – | 0.32/NS |
| Omelette-20 | 3.29/1551/NS | 2.11/1551/NS | – | 0.53/NS |
| Ring-2 | 0.02/18/7 | 0.02/22/12 | 0.00/7 | 0.00/5 |
| Ring-3 | 0.17/36/15 | 0.21/41/18 | 0.00/12 | 0.00/8 |
| Ring-4 | 2.69/66/26 | 25.13/127/25 | 0.00/17 | 0.00/11 |
| Ring-5 | – | – | 0.03/22 | 0.00/14 |
| Ring-6 | – | – | 0.06/27 | 0.01/17 |
| Safe-5 | 0.00/5/5 | 0.00/5/5 | 0.01/5 | 0.00/5 |
| Safe-10 | 0.05/10/10 | 0.07/10/10 | 0.15/10 | 0.01/10 |
| Safe-30 | 3.54/30/30 | 3.79/30/30 | – | 0.06/30 |
| Safe-50 | 81.58/50/50 | 90.68/50/50 | – | 0.16/50 |
| Safe-70 | 407.91/70/70 | 409.93/70/70 | – | 0.35/70 |

Times $t$ in seconds, search space size $|S|$ (number of evaluated states), plan length $l$. Dashes indicate time-outs, NS indicates "no solution", i.e. proved unsolvable.

suites, Cube-corner and Cube-center, where the goal is to move into a corner and the center of the grid, respectively. To get to a corner, it suffices to move $n$ steps in each dimension, in direction of the corner. To get to the center, one must first make sure to be in some corner, then move back into the center from there. The Cube-corner-$n$/Cube-center-$n$ instances in Table 2 are those with extension $n$.

In Cube-corner, we observe that both Conformant-FF versions scale well, if not competitively with the extreme efficiency of KACMBP. MBP fails to scale up. Regarding plan lengths, $h^{\mathcal{I}}$-Conformant-FF and KACMBP find the optimal plans. Using $h^{\phi(act)}$, the plans contain one unnecessary action (this is due to a minor implementation detail). MBP's plans are somewhat longer. It is interesting to observe that, for both Conformant-FF versions, the search space is, again, equal to the length of the plan and thus the smallest search space possible. The runtime spent mostly goes into the computation of state transitions, i.e. of known propositions.

In Cube-center, the behavior of the tested planners is very different. Most strikingly, $h^{\mathcal{I}}$-Conformant-FF completely fails to scale up. Using $h^{\phi(act)}$, performance is better but still a lot worse than in Cube-corner. For both Conformant-FF versions, the performance decrease is mainly due to a much larger search space than before. The behavior of MBP and KACMBP does not change that much; the most significant difference is that, now, it is MBP that finds the optimal plans. KACMBP plan's are a little longer than optimal, Conformant-FF's plans are a lot longer than optimal, most of the time.

Obviously, $h^{\phi(act)}$ is a much better heuristic than $h^{\mathcal{I}}$ in Cube-center. Also obviously, the reason for that must be $h^{\phi(act)}$'s more accurate reasoning about past achievements—there is no other difference between the heuristics. What's less obvious is exactly what phenomenon makes it important to reason about past achievements in Cube-center. Here are some intuitions we got from looking at some relaxed plans in small examples. For the initial state of a Cube-center instance, the relaxed plan (which is equal for both heuristics) is relatively short. When ignoring the delete lists, one can

reach the center of an $n$-cube (for odd $n$) with $n-1$ steps in each dimension: the old positions are not deleted, and so, to reach the center from any position, it suffices to do $\lfloor n/2 \rfloor$ steps up and $\lfloor n/2 \rfloor$ steps down. That is, in the relaxation the planner does not realize that it has to move to a corner first. This yields local minima that are hard to explore. They are harder to explore for $h^{\mathcal{I}}$ because, as the number of move actions on the path to a believe state increases, $h^{\mathcal{I}}$'s over-approximating reasoning about these action's consequences is likely to conclude that the goal was already achieved by them. Indeed, with $h^{\mathcal{I}}$ Conformant-FF gets down to a state with empty relaxed plan extremely quickly, and then spends ages trying to find a state that actually satisfies the goal.[14]

In the Omelette domain, $n$ eggs must be broken into a bowl without spoiling the bowl. There are two bowls that can both contain up to $n$ eggs. One can "grab" an egg (one can do this infinitely often, i.e. there is no finite store of eggs), one can "break" the egg into one of the bowls, or one can "clean" a bowl, i.e. dispose of its contents. One can also "pour" the contents of one bowl into another, with the obvious effect depending on the fill status of both bowls. Breaking an egg into a 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 Omelette tasks. We used the "dummy" encoding in the input tasks for all tested planners. The instances Omelette-$n$ in Table 2 are, of course, those with $n$ eggs in the goal.

Once again KACMBP scales—i.e., proves these tasks unsolvable—best, followed closely by Conformant-FF and not so closely by MBP. Note that proving unsolvability involves, for all the systems here, exhausting the space of reachable belief states. One interesting observation is that $h^{\mathcal{I}}$ and $h^{\phi(act)}$ share the same search space—the dead ends detected by both heuristics, i.e. the states with heuristic value $\infty$, are the same. Unexpectedly, $h^{\phi(act)}$ is a little faster. The heuristic computation of $h^{\mathcal{I}}$ involves traversing longer paths in the *Imp* tree (when finding leafs of this tree), since that tree goes back all the way to the initial state. This takes more time than the state-formula CNF reasoning done by $h^{\phi(act)}$. Probably this time overhead is just due to our relatively naive implementation of traversing paths in the *Imp* tree. We remark that, using $h^{inc}$, the runtime is roughly similar to that of $h^{\mathcal{I}}$, but a few more states (precisely, $n+2$ more states) are explored since $h^{inc}$ cannot detect dead ends.

In the Ring domain, there are $n$ rooms through which one can "move" in a cyclic fashion. The initial location is unknown. 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. The instances Ring-$n$ in Table 2 are those with $n$ rooms. MBP and KACMBP scale very well, Conformant-FF does not, independently of the heuristic used. KACMBP finds the optimal plans, MBP's plans are a little longer, Conformant-FF's plans are considerably longer.

The poor runtime performance of Conformant-FF here is, partly, due to the oddity in repeated states checking that we mentioned at the end of Section 5. No proposition becomes known until the entire task is solved, every visited state is stored in the same hash table entry, and the number of state comparisons is a square function in the search space size. Turning the repeated states check off, and using only the stagnating states check instead, Conformant-FF with $h^{\mathcal{I}}$ solves the five Ring examples with 0.00/18/7, 0.04/50/20, 0.29/88/39, 1.79/139/64, 9.55/201/95 seconds/search states/actions, respectively. Section 9 outlines some ideas on alternative ways to avoid the overhead of full repeated states checking.

In the Safe domain, a safe has one out of $n$ possible combinations, and one must "try" all combinations in order to assure the safe door is open. The instances in Table 2 are those with $n$ combinations. Once again, KACMBP is extremely efficient. Conformant-FF is a lot slower, MBP can solve only the two smallest examples within the time limit. All planners find the optimal $n$-action plans trying all combinations in some order. There is no significant difference between the performance of $h^{\mathcal{I}}$ and $h^{\phi(act)}$, which both return perfect (equal to the real goal distance) heuristic values. Like in Ring, most of the runtime goes into the repeated states checking, because the first time a proposition ("open-

[14] A possible way out of this dilemma would be to extend Conformant-FF with mechanisms structuring the search based on "necessary knowledge", as is done in KACMBP [19]: the "necessary knowledge" to reach the center of the cube is the precise position; posing this knowledge as a goal means to move into a corner, so a decomposition of this sort would likely solve Conformant-FF's difficulties in Cube-center.

safe") becomes known is when the task is solved. With only stagnating states checking, $h^{\mathcal{I}}$-Conformant-FF solves the tasks with 0.00/5/5, 0.00/10/10, 0.16/30/30, 1.59/50/50, 6.27/70/70 seconds/search states/actions, respectively.

### 6.4. Enriched classical benchmarks

Table 3 provides our experimental results in the enriched Blocksworld and Logistics domains. A quick glance at the results reveals that Conformant-FF clearly outscales both MBP and KACMBP in all our test suites.

In all our classical domains enriched with uncertainty, we scale the instances in terms of an *uncertainty parameter*, denoted "U" in the names of the instances, and in terms of various size parameters as appropriate in the domain. For each domain, we have three test suites, with U values 2, 3, and 4, respectively (more details are below).

The Blocksworld domain we use is the variant with three operators to put a block $x$ from another block $y$ onto the table ("move-to-table"), to put a block $x$ from a block $y$ onto a different block $z$ ("move"), and to put a block $x$ from the table onto a block $y$ ("move-from-table"). The uncertainty in our test suites is that the top U blocks on each initial stack (which may be the whole stack) are arranged in an unknown order. 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. Across our three test suites, the examples are generated with the software by Slaney and Thiebaux [20]. The instances named Blocksworld-U$U$-b$b$ in Table 3 contain $b$ blocks. Conformant-FF shows fine scalability; the behavior of $h^{\mathcal{I}}$

Table 3
Results in the enriched Blocksworld and Logistics domains

| Instance | Conformant-FF | | MBP | KACMBP |
|---|---|---|---|---|
| | $h^{\mathcal{I}}$ | $h^{\phi(act)}$ | | |
| | $t/\|S\|/l$ | $t/\|S\|/l$ | $t/l$ | $t/l$ |
| Blocksworld-U2-b5 | 0.00/9/7 | 0.00/9/7 | 0.75/7 | 0.35/5 |
| Blocksworld-U2-b6 | 0.02/19/10 | 0.01/19/10 | 16.01/10 | 0.53/17 |
| Blocksworld-U2-b7 | 0.03/31/13 | 0.04/31/13 | 1148.25/14 | 1.24/11 |
| Blocksworld-U2-b13 | 0.44/93/20 | 0.42/93/20 | – | – |
| Blocksworld-U2-b20 | 3.75/310/34 | 3.77/310/34 | – | – |
| Blocksworld-U3-b5 | 0.00/7/7 | 0.01/7/7 | 0.53/8 | 0.10/8 |
| Blocksworld-U3-b6 | 0.01/16/12 | 0.01/16/12 | 10.62/12 | 1.08/29 |
| Blocksworld-U3-b7 | 0.03/30/16 | 0.04/30/16 | – | – |
| Blocksworld-U3-b13 | 0.75/136/32 | 0.70/136/32 | – | – |
| Blocksworld-U3-b20 | 5.49/423/42 | 5.37/423/42 | – | – |
| Blocksworld-U4-b5 | 0.01/7/7 | 0.00/7/7 | 0.74/7 | 0.10/7 |
| Blocksworld-U4-b6 | 0.08/22/19 | 0.08/22/19 | 19.15/19 | 1.18/11 |
| Blocksworld-U4-b7 | 0.13/28/23 | 0.13/28/23 | 1028.14/21 | 1.35/13 |
| Blocksworld-U4-b13 | 2.92/83/43 | 4.73/83/43 | – | – |
| Blocksworld-U4-b20 | 1.96/171/48 | 1.87/171/48 | – | – |
| Logistics-U2-c2-p2-a1 | 0.00/12/11 | 0.00/12/11 | 1.56/13 | 0.14/17 |
| Logistics-U2-c2-p4-a1 | 0.01/23/23 | 0.02/23/23 | 31.71/25 | 3.15/49 |
| Logistics-U2-c3-p2-a1 | 0.01/26/17 | 0.01/26/17 | 38.16/17 | 0.80/27 |
| Logistics-U2-c3-p3-a2 | 0.03/48/26 | 0.04/48/26 | – | 615.95/196 |
| Logistics-U2-c10-p10-a10 | 4.81/298/81 | 4.52/298/81 | – | – |
| Logistics-U3-c2-p2-a1 | 0.03/27/19 | 0.03/27/19 | 13.87/22 | 0.66/23 |
| Logistics-U3-c2-p4-a1 | 0.05/39/32 | 0.04/39/32 | 908.11/54 | – |
| Logistics-U3-c3-p2-a1 | 0.03/32/20 | 0.02/32/20 | 780.19/28 | 6.41/29 |
| Logistics-U3-c3-p3-a2 | 0.05/43/26 | 0.06/43/26 | – | – |
| Logistics-U3-c10-p10-a10 | 12.90/437/110 | 14.83/437/110 | – | – |
| Logistics-U4-c2-p2-a1 | 0.01/21/17 | 0.01/21/17 | 76.34/28 | 1.25/24 |
| Logistics-U4-c2-p4-a1 | 0.14/57/42 | 0.16/57/42 | – | – |
| Logistics-U4-c3-p2-a1 | 0.05/31/18 | 0.04/31/18 | – | 60.13/36 |
| Logistics-U4-c3-p3-a2 | 0.16/77/41 | 0.18/77/41 | – | – |
| Logistics-U4-c10-p10-a10 | 26.95/555/114 | 31.70/555/114 | – | – |

Times $t$ in seconds, search space size $|S|$ (number of evaluated states), plan length $l$. Dashes indicate time-outs.

Table 4
Results in the enriched Grid domain, random instances

| Instance | Conformant-FF | | MBP | KACMBP |
|---|---|---|---|---|
| | $h^{\mathcal{I}}$ | $h^{\phi(act)}$ | | |
| | $t/\|S\|/l$ | $t/\|S\|/l$ | $t/l$ | $t/l$ |
| Grid-U2-x3-y3-s2-k11-l22 | 0.03/27/14 | 0.06/27/14 | MEM | 1.17/26 |
| Grid-U2-x3-y3-s2-k11-l22 | 0.05/33/16 | 0.08/33/16 | MEM | 10.75/54 |
| Grid-U2-x3-y4-s2-k12-l23 | 0.11/53/24 | 0.25/53/24 | MEM | 783.82/48 |
| Grid-U2-x3-y4-s2-k12-l23 | 0.26/114/27 | 0.59/114/27 | MEM | – |
| Grid-U2-x3-y3-s2-k11-l22 | 0.22/104/28 | 0.56/104/28 | MEM | – |
| Grid-U3-x3-y3-s3-k111-l111 | 0.00/13/8 | 0.01/13/8 | MEM | 10.36/14 |
| Grid-U3-x3-y4-s3-k112-l112 | 0.07/44/21 | 0.22/44/21 | MEM | 254.12/49 |
| Grid-U3-x3-y4-s3-k112-l112 | 0.12/48/24 | 0.33/48/24 | MEM | – |
| Grid-U3-x3-y3-s3-k111-l111 | 0.13/62/28 | 0.28/62/28 | MEM | – |
| Grid-U3-x3-y3-s3-k111-l111 | 0.60/223/40 | 1.44/223/40 | MEM | – |
| Grid-U4-x3-y4-s4-k1111-l1111 | 0.02/8/8 | 0.03/8/8 | MEM | 88.53/8 |
| Grid-U4-x3-y4-s4-k1111-l1111 | 0.02/12/12 | 0.06/12/12 | MEM | – |
| Grid-U4-x3-y4-s4-k1111-l1111 | 0.04/18/12 | 0.08/18/12 | MEM | – |
| Grid-U4-x3-y4-s4-k1111-l1111 | 0.24/60/25 | 0.70/60/25 | MEM | – |
| Grid-U4-x3-y4-s4-k1111-l1111 | 0.29/64/30 | 0.83/64/30 | MEM | – |

Times $t$ in seconds, search space size $|S|$ (number of evaluated states), plan length $l$. Dashes indicate time-outs, MEM indicates out of memory.

and $h^{\phi(act)}$ is nearly identical. MBP and KACMBP do not scale up. No planner is clearly superior in terms of plan length.

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, i.e. city size is our uncertainty parameter U. The instances named Logistics-U$U$-c$c$-p$p$-a$a$ in Table 3 contain $c$ cities (of size $U$), $p$ packages to be transported, and $a$ airplanes. In each city, there is a single truck. All the examples are randomly generated by uniformly choosing the initial positions of trucks and airplanes, and the initial positions and goal positions of packages. The first four instances in each of the test suites were chosen relatively small in order to be able to obtain some plans with MBP. The largest examples, with 10 cities, packages, and airplanes, were chosen to show Conformant-FF's fine scalability. Again there is not much difference between using $h^{\mathcal{I}}$ or $h^{\phi(act)}$; the latter is a little slower in most examples (particularly, in the largest one). The plans found by MBP or KACMBP are, in most cases, considerably longer than those found by Conformant-FF.

Table 4 shows our comparative results in the enriched Grid domain. The Grid domain was used in the 1st international planning competition (alongside AIPS-98). 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. The shape of these locks was specified to be an unknown one out of U 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. The instances parameters of this domain are more complicated than those of the domains we have seen previously, and so the names of the instances, as show up in Table 4, are a little complicated. Grid-U$U$-x$x$-y$y$-s$s$ means that the grid has $x$-extension $x$ and $y$-extension $y$, and there are $s$ different shapes. The parameters indicated by "-k" and "-l" provide $s$ numbers: the number of keys and locks of each individual shape, namely. So in Grid-U2-x3-y3-s2-k11-l22 there are one key, and two locks, of each of the two shapes. We generated our instances randomly, using software that uniformly distributed the initial positions of robot, keys, and locks, as well as the goal positions of the keys. Of course, such a generation process can result in unsolvable examples—e.g. if, in order to get to a key with a certain shape, the robot has to move across a lock with the same shape. Since the examples we generated for the experiment were relatively small, Conformant-FF could determine unsatisfiability within a few seconds for all the tasks we generated. For the test suites, we used the first few solvable tasks that came up. As one can see in Table 4, the test examples within the suite for each U value do not differ very much in terms of their size parameters. The difficulty of instances in Grid varies a lot already with the distribution of

Table 5
Results in the enriched Grid domain, AIPS'98 instances

| Instance | $h^{inc}$ $t/\|S\|/l$ | $h^{\mathcal{I}}$ $t/\|S\|/l$ | $h^{\phi(act)}$ $t/\|S\|/l$ |
|---|---|---|---|
| Grid-U2-AIPS98-1 | 0.09/25/16 | 0.16/25/16 | 0.23/25/16 |
| Grid-U2-AIPS98-2 | 1.71/198/61 | 1.71/198/61 | 4.50/198/61 |
| Grid-U2-AIPS98-3 | 3.39/386/83 | 3.74/386/83 | 13.63/386/83 |
| Grid-U2-AIPS98-4 | 3.86/368/60 | 3.83/368/60 | 7.79/368/60 |
| Grid-U2-AIPS98-5 | 150.37/1516/224 | 163.47/1516/224 | – |
| Grid-U3-AIPS98-1 | 0.28/52/24 | 0.46/52/24 | 1.19/52/24 |
| Grid-U3-AIPS98-2 | 5.14/378/96 | 5.96/378/96 | 36.78/472/91 |
| Grid-U3-AIPS98-3 | 7.43/548/83 | 8.68/548/83 | 40.72/548/83 |
| Grid-U3-AIPS98-4 | 10.44/692/79 | 11.50/692/79 | 37.31/692/79 |
| Grid-U3-AIPS98-5 | 782.55/3744/279 | 832.01/3744/279 | – |
| Grid-U4-AIPS98-1 | 0.73/71/34 | 0.71/71/34 | 2.43/71/34 |
| Grid-U4-AIPS98-2 | 25.44/943/127 | 29.45/943/127 | 485.62/1870/138 |
| Grid-U4-AIPS98-3 | 14.86/763/97 | 18.12/763/97 | 123.75/763/97 |
| Grid-U4-AIPS98-4 | 18.70/1102/91 | 21.92/1102/91 | 87.79/1102/91 |
| Grid-U4-AIPS98-5 | 759.44/2477/289 | 828.39/2477/289 | – |

Times $t$ in seconds, search space size $|S|$ (number of evaluated states), plan length $l$. Dashes indicate time-outs.

the locks and keys. We ordered the instances in terms of their difficulty as indicated by the length of the plan found by Conformant-FF. As above in Blocksworld and Logistics, both Conformant-FF versions behave very similar, sharing the same search space; in difference to before, $h^{\phi(act)}$ is quite clearly slower than $h^{\mathcal{I}}$. In all examples, MBP runs out of memory before starting the actual search; the largest instance we could solve with MBP had only 4 positions on the Grid. KACMBP can solve some of the smaller instances, finding comparatively long plans except in one case.

In our random generation process for Grid, the need for filtering unsolvable examples limits the example size severely. To show that Conformant-FF really scales up in the domain, we therefore also ran an experiment with the five Grid instances used in the AIPS-98 competition. These examples are rather large. In each of them there are 4 different shapes. They feature $5 \times 5$, $6 \times 6$, $7 \times 7$, $8 \times 8$, and $9 \times 9$ grids, respectively; there are 9, 10, 11, 12, and 13 keys; there are 8, 8, 10, 9, and 19 locks. As before, we introduced uncertainty into the instances by giving every lock a range of U possible shapes, U increasing from 2 to 4. We did not try to run MBP and KACMBP on these large examples. Instead we show results for all three variations of Conformant-FF's heuristic function, since they all differ considerably in these challenging tasks. The data are in Table 5.

The main difference between the heuristic functions lies in the speed of computation. In all except two cases—Grid-U3-AIPS98-2 and Grid-U4-AIPS98-2—the search spaces (and returned plans) are the same. Comparing $h^{inc}$ to $h^{\mathcal{I}}$, skipping the SAT reasoning in the heuristic computation yields a small but consistent time advantage. Comparing $h^{\mathcal{I}}$ to $h^{\phi(act)}$, doing the SAT reasoning with the initial state formula, rather than with the formula describing the considered belief state, yields huge savings. Note that the examples become considerably harder to solve as uncertainty increases (with the single exception of Grid-U3-AIPS98-5 and Grid-U4-AIPS98-5). The plan length increases in all cases. With more uncertainty, more keys must be transported to open the necessary locks.

All in all, from our results in the enriched classical 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 and KACMBP. We consider this an important advantage because it seems likely that, in real-world domains, classical and conformant (uncertain) aspects typically occur together.

### 6.5. Comparison to GPT and POND

We also ran some experiments with GPT and with POND. We briefly summarize the results, first for GPT then for POND.

GPT is known to be slower than MBP and KACMBP in the traditional conformant domains. 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 KACMBP, and much slower

than Conformant-FF. In the three smallest Blocksworld-U2 examples, the per-instance runtimes (pure solution times excluding parsing and compilation) we got are 45.77, 112.91, and 1025.33 seconds for GPT as compared with 0.75, 16.01, and 1148.25 for MBP, 0.35, 0.53, and 1.24 for KACMBP, and a few milliseconds for Conformant-FF. For the smallest three instances of Logistics-U2, GPT takes 1.89, 37.19, and 415.44 seconds, MBP takes 1.56, 31.71, and 38.16 seconds, KACMBP takes 0.14, 3.15, and 0.80 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, to a varying degree, all of MBP, KACMBP, and Conformant-FF are superior to GPT in these domains.

POND's input language for conformant planning is very similar to that of Conformant-FF; all one needs to do is to replace some keywords in the problem descriptions. We ran POND on all the test examples used in the detailed experiments above, but could get it to find plans in only a fairly small subset of these. If that is due to details in our encoding of the examples, or to inefficiencies in POND's implementation, or to real weaknesses of POND's approach (more on that approach below in Section 8), is hard to tell.

In the Cube examples, POND terminated with a segmentation fault. In most of the other test suites, in the larger examples POND ran out of memory during the pre-process that builds its "labelled uncertainty graph" (*LUG*), the structure that underlies its heuristic computations (see also Section 8). In the small examples that were solved, that pre-process terminated very quickly. The detailed results were as follows (we do not repeat the data for the other planners, given in the tables above).

- In Bomb, the LUG ran out of memory on all examples with more than 10 bombs. The other examples, Bomb-t1-b5, Bomb-t10-b10, Bomb-1-5, Bomb-1-10, Bomb-5-10, Bomb-10-5, were solved in 1.08, 59.01, 0.20, 1.63, 19.10, and 4.23 seconds, respectively. This is considerably slower than either of Conformant-FF, MBP, and KACMBP; compare Table 1. The number of plan steps was always twice the number of bombs, i.e. as with KACMBP no use of additional toilets is made.
- In Omelette, the LUG ran out of memory even in the smallest example.
- The smaller four instances of Ring were solved in 0.13/6, 0.64/11, 14.18/15, and 694.91/15 seconds/steps, respectively. On the largest instance, POND ran out of time. That is, here POND scaled better than Conformant-FF, finding good-quality plans.
- The five Safe instances were solved in 0.13, 0.18, 7.12, 90.77, and 549.54 seconds, with optimal plans. Here, POND's performance is roughly similar to that of Conformant-FF.
- In Blocksworld, the LUG ran out of memory on the two larger examples in each suite, i.e., on those with more than 7 blocks. In the smaller examples, POND is competitive with MBP and KACMBP(solving one more example), but lags far behind Conformant-FF. Precisely, the three smaller Blocksworld-U2 examples are solved in 2.32/6, 16.91/10, and 65.18/12 seconds/steps. In Blocksworld-U3 these numbers are 1.95/7, 10.00/9, and 48.63/15. In Blocksworld-U4 they are 1.67/8, 32.74/19, and 206.13/24.
- In Logistics, the LUG ran out of memory in all but three of the smallest examples. The three solved examples are Logistics-U2-c2-p2-a1, Logistics-U2-c2-p4-a1, and Logistics-U3-c2-p2-a1. They are solved in 3.73/7, 16.00/11, and 41.55/10 seconds/steps. Compared to the other planners here, POND is relatively slow but finds very good plans (the best one found, in all three cases).
- In Grid, the LUG ran out of memory even in the smallest example.

## 7. Non-deterministic effects

If an effect is marked as non-deterministic, and is applied to a world state $w$, then the outcome are two possible successor world states $w'$ and $w''$—one where the effect was applied, one where it was not applied. In the presence of such effects, a conformant plan must achieve the goals for all possible initial world states *and* for all possible outcomes of the non-deterministic effects. Formulated in terms of belief states, the criterion that a conformant plan has to meet is still that it ends in a belief state $s$ where all world states $w$ in $s$ contain the goals. We have not yet implemented support for non-deterministic effects in Conformant-FF. While the needed extensions are very easy for the computation of known propositions, and for the computation of relaxed plans, it is unclear how to adequately extend our repeated states checking. Doing the latter is a topic for future work. Here, we outline the main issues arising in that important context.

The computation of known propositions can be extended to non-deterministic effects as follows. When generating the formula $\phi(act)$ encoding the semantics of an action sequence *act*, simply skip non-deterministic effects when inserting the effect axioms, i.e., insert those for deterministic effects only. The rest of the formula's definition, as well as the SAT calls made to check for known/negatively known propositions, can remain exactly the same. To see this, reconsider Proposition 2. With non-deterministic effects, there are now several satisfying assignments to $\phi_I(act)$—namely, corresponding exactly to all possible outcomes of the non-deterministic effects. (When non-deterministic effects do not enforce their adds and deletes by effect axioms, but the frame axioms for the respective proposition values respect these effects, then consequently the affected variables can take on both truth values in the next time step.) Still, the formula *implies* a proposition value at a time step only if the value holds at this step in *all* possible outcomes of the action sequence, which is exactly what we need to know about.

To extend the relaxed planning process to non-deterministic effects, it suffices to ignore these. A non-deterministic effect cannot be used to establish a proposition, and in the relaxed setting a non-deterministic effect, like the deterministic ones, cannot interact with the propagation of a positive truth value. Note that we make no claims whatsoever about how this technique, and the technique described above, will behave in practice; this has to be tested empirically.

In the repeated states checking, one can, in theory, proceed with an extension just as simple as the ones outlined above. Consider the belief state equivalence test. Say we simply leave the entire machinery unchanged, using the extended definition of the formulas $\phi(act)$. Say $\phi_{I=}(act) \wedge \phi_{I=}(act') \wedge \neg p(act) \wedge p(act')$ is unsatisfiable. With the extended definition of the formulas $\phi(act)$, this means that for all possible initial world states, for all possible outcomes of the non-deterministic effects, $p$ can't be false after *act* but true after $act'$. So by doing exactly the same SAT calls as before, one gets a sufficient test for belief state equality. The test succeeds iff, starting from every possible initial world state, the resulting world states in $s$ and $s'$ are the same *independently of the outcomes of the non-deterministic effects*. Note that this is a very strong requirement. It is the nature of non-deterministic effects to result in several different world states. Unless these different world states are merged back into a single one by later actions in the sequence, the test will fail. Consider the following example. The initial state formula is empty, and there is a single action $a$ that has no other effects than non-deterministically adding a proposition $p$. Obviously, the belief state after $act = \langle a \rangle$ is equal to the belief state after $act' = \langle a, a \rangle$. The reached world states are $\{p\}$ and $\emptyset$ in both cases. However, the formula $\phi_{I=}(act) \wedge \phi_{I=}(act') \wedge \neg p(act) \wedge p(act')$ asks if it can be the case that $p$ is false after *act* but true after $act'$. And of course that can happen—namely, when $a$'s effect does not occur on *act*, but does occur on $act'$. Note that the SAT test would fail (the formula would be satisfiable) even if we were to compare *act* to itself.

Similar difficulties as above can be observed with the (straightforward extension of the) belief state domination test. The problem with the SAT-based equivalence/domination tests is that, in the presence of non-deterministic effects, the formulas $\phi$ encode a *set* of world states (rather than just a single world state) for every possible initial world state $I$. This makes the SAT tests lose precision. As said, developing repeated states checking methods that are practically useful, in the presence of non-deterministic effects, is a topic for future work.

## 8. Related work

A better understanding of Conformant-FF can be obtained by examining its relation to a number of strands of work on conformant planning. Below we describe the three main strands of research on conformant planning, and then place Conformant-FF in their context.

CGP [21] is the first specialized conformant planner. CGP takes the essential ideas behind the Graphplan [15] algorithm and applies them in the context of conformant planning. For its forward reachability phase, CGP builds one planning graph for each possible initial world state, while recording mutexes within each graph and between graphs. It searches backward for a solution on these planning graphs. Action choices made in one graph are propagated to the others during this process. However, it has difficulty scaling up when the number of initial states is large. Thus, we can characterize CGP's approach as based on utilizing representation and solution methods from classical planning for reasoning about individual initial states, while propagating information between these solutions in a sophisticated way. A close relative of this is Frag-Plan [22], which generates plans (using planning-graphs) for concrete initial states, attempting to first tackle initial states for which finding a plan is harder. The initial plan is then extended to cover all possible initial states by including fragments from plans for different initial states. This requires some back-and-forth reasoning between different states: while adapting an existing plan to work for a new initial state $s$ we may make it inappropriate for some previously handled initial state $s'$. While fixing this problem, we may introduce a

problem somewhere else, etc. C-Plan [8] is related to CGP and Frag-Plan in the sense that it also relies on classical planning techniques to a large extent. Namely, it uses sat-based (classical) planning techniques to generate candidate plans: a candidate plan is a satisfying assignment to the initial state formula plus the usual SAT encoding, meaning that the candidate is a plan for at least one possible initial world (the one chosen by the assignment). It is then tested (by another SAT test) whether the candidate is, in fact, a conformant plan. Like CGP and Frag-Plan, this approach is likely to have trouble scaling in the number of initial states—one would expect the number of candidate plans to be closely related to the number of initial states, although the relationship can be complicated, in general. Unlike CGP and Frag-Plan, C-Plans generate-and-test approach has no backtracking element; the candidate plans are viewed separately and no interactions between them are explored.

The most successful strand of work on conformant planning is based on the idea of search in the space of belief states. Planners from this breed differ in two central parameters: how they represent belief-states and how they search in this space. In particular, among these planners, the most successful ones employ some form of heuristic search. The first planner to utilize belief-space search is GPT [1]. GPT represents belief-states explicitly, and searches this space using A*-based forward heuristic search. The heuristic utilizes relaxed dynamic programming techniques to compute reachability and distance estimates. GPT finds optimal plans, but has problems scaling up because, first, its heuristic is often not informative enough and, second, aside from the size of the belief *space*, the size of the belief *states* can be prohibitive even in relatively simple conformant planning problems. To remedy this problem, CMBP [2] utilizes symbolic, BDD-based data structures for representing belief states, and uses BDD manipulation techniques to perform state update. This change leads to a much more compact and manageable approach for planning in belief space. However, CMBP performs uninformed breadth-first search, and so its ability is still limited, like GPT's, through the size of the belief space. This problem was addressed by HSCP [23]. HSCP uses symbolic machinery much like CMBP to perform backward heuristic search. That is, it performs regression at the level of belief state representation. The regression of a belief-state $b$ over an action $a$ is the largest belief-state for which by applying $a$ we get to a belief state that is contained in $b$. The heuristics used there is based solely on the cardinality of the belief state. While better than CMBP, this heuristic is somewhat simplistic. A more advanced planner related to HSCP is KACMBP [19]. KACMBP uses forward search. Thus, it is informed by the initial state and considers reachable states only. KACMBP also uses a more sophisticated heuristic approach. Although it also considers the cardinality of belief states, it does not do so blindly. KACMBP attempts to actively achieve knowledge that it deems useful by preferring actions that lead to states with more desirable knowledge.[15]

CAltAlt and POND [7,24] are two more recent planners that utilize heuristic search over belief states. CAltAlt searches backwards, whereas POND searches forward. Like CMBP and its descendants, POND uses BDDs to represent the current belief state, while CAltAlt uses a clause-based representation. However, they are also closely related to CGP in that they make extensive use of planning graphs in generating their heuristics estimates. Indeed, the main novel ideas in these planners are their more informed heuristic estimates and the data-structures used to obtain them. In particular, they introduce a labeled uncertainty graph, a planning graph-like structure that stores in one planning graph information that CGP obtains from multiple planning graphs. This is done by labeling certain facts with the initial state conditions that support them (represented using BDDs), and propagating these labels over the actions. However, unlike CGP, the planning graph is not used to actually perform the search.

Finally, there is a third strand of work on conformant planning which we shall refer to as the logic-based approach. What is common to the set of planners in this (more loosely related) category is the reduction of conformant planning to inference tasks in existing logical formalisms. We have already discussed the C-Plan planner which uses SAT techniques. C-Plan generates candidate plans using planning-as-satisfiability based methods, and then uses unsatisfiability checks to see whether they are actually conformant plans. QBFPLAN [25] uses the more powerful (and more complex) formalism of quantified boolean formulas. That is, it reduces the question of the existence of a (bounded-length) conformant plan to the satisfiability of a quantified boolean formula. A different reduction is used by DLV$^k$ [26]. It reduces (bounded) conformant planning to answer-set programming and uses the rich domain description language $\mathcal{K}$. Somewhat related to this approach is the Petrick and Bacchus' PKSPlan [18] which shares with DLV$^k$ the idea of us-

[15] Note: CMBP was integrated into the more general MBP planner, which supports, in addition, different search methods (forward, backward, heuristic) as well as contingent planning and more. Thus, in the experimental section and throughout much of the paper we speak of MBP, not CMBP. In terms of historical progression of ideas, the above is a more useful description. However, restricted to conformant planning, KACMBP is a more advanced planner than the current MBP.

ing a rich domain description language, albeit a different one. PKSPlan uses logical formulas to represent the current state of *knowledge* of the agent. Its actions explicitly manipulate these knowledge formulas.

In terms of performance, belief-space search planners are the best studied and, probably, the best performers. A good experimental analysis of the more successful approaches appear in [19]. Of these planners, aside from Conformant-FF, KACMBP appears to be the most efficient, outperforming CGP and its related planners. We note that w.r.t. Conformant-FF, their results basically agree with our results: KACMBP is generally better on the traditional conformant domains, whereas Conformant-FF is better on the mixed domains. It is somewhat difficult to compare them to the logic-based planners because of differences in the input languages. For instance, the input language used by DLV$^k$ is much richer than the PDDL-like language Conformant-FF uses, whereas the language used by PKSPlan is somewhat incomparable. The comparisons of CAltAlt and KACMBP in [24] shows that CAltAlt is competitive with KACMBP, in fact, fairing better than it in some rover and logistics problems. However, that comparison does not include many standard test domains (e.g., cube, ring, omelette, safe).

Conformant-FF is best viewed as a belief-space search planner, as it represents the current state of the search process using a structure that is equivalent to a belief state. Its main innovation is the implicit manner in which it describes these belief-states. This description is much more compact than the explicit description used by any of the other belief-space search planners. The price Conformant-FF pays for this space reduction is in the time required to answer queries about each belief state. However, as evident from our results, this tradeoff often pays off. Another interesting innovation of Conformant-FF is the use of a single planning-graph for computing a relaxed plan. Unlike the multiple planning graphs used by CGP, this single planning graph is able to capture information about multiple initial states, by explicitly differentiating between known and unknown propositions. This allows Conformant-FF to produce a relatively informed heuristic estimate (in fact, a relaxed plan) quickly. CAltAlt and POND also utilize more complex reasoning on top of a single planning graph, but that appears more complex and more costly to construct. Conformant-FF does not take into account issues such as the cardinality of a belief-state or desirable knowledge in this computation. Possibly, by taking these issues into consideration, following KACMBP, a more robust heuristic estimate can be obtained (see also next section). Finally, Conformant-FF is related to the logic-based strand in its use of logical formulas to represent the current belief-state, and the use of (un)sat queries to check properties of this state, which is virtually identical with the use of such checks by C-Plan to determine whether a candidate plan solves the task. Conformant-FF simply makes much more extensive use of these checks, using them to check properties of partial plans, as well.

## 9. Conclusion

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 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 a complete but unsound form of the CNF reasoning about known propositions. The resulting planner Conformant-FF is competitive with the state-of-the-art conformant planners MBP, KACMBP, GPT, and POND, sometimes outperforming all of them. Conformant-FF shows the potential to combine the strength of FF with conformant abilities.

An 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 tries to minimize estimated goal distances, the latter system tries to minimize belief state size (more precisely, tries to accomplish useful information about the state). The comparative data obtained in our experiments reveals that the two kinds of approaches 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.

It may be possible to overcome the huge overhead induced by Conformant-FF's repeated states checking, as observed in the Ring and Safe domains, when there is a lack of different known/unknown propositions in the search space. One can introduce incomplete pre-tests to avoid full CNF tests for state pairs $s$ and $s'$ that can quickly shown

to be non-equal/non-dominating. For example, say $\phi$ is the CNF whose satisfiability must (repeatedly) be tested for deciding whether $s$ is equal to $s'$. If even a 2-projection of $\phi$ is solvable, then $\phi$ is solvable, and $s$ is not equal to $s'$.

In the extension to non-deterministic effects, as said it is most important (and difficult) to come up with an adequate extension of repeated states checking. One idea that may turn out feasible is to enumerate the outcomes of the non-deterministic effects on the action sequences *act* and *act*$'$. Say it is the case that, for every initial world state, for every outcome of the effects on *act*, there is an outcome of the effects on *act*$'$ so that the resulting world states are equal. Then the belief state represented by *act*$'$ is a superset of the one represented by *act*, and *act*$'$ can be pruned if *act* was visited before. Note that, formulated in terms of a satisfiability test, the outlined test involves a quantifier alternation in the prefix of a CNF formula. Such a test might be best implemented using a QBF solver.

An important observation is that our approach is not inherently restricted, through relying on its "known propositions" computation, to conjunctive goals and action preconditions. If the goal is a DNF formula $G$, then an action sequence *act* ends in a goal state iff $\phi(act)$ implies $G$ at its end point. That entailment check can be done by conjoining $\phi(act)$ with the negation of $G$, yielding a new CNF, and seeing whether that new CNF is satisfiable. Similarly, DNF action preconditions can be dealt with. DNF effect conditions can be equivalently split up into a separate effect for each disjunct. Together with the standard pre-processes known from classical planning, compiling quantifiers and negations away [27], with this, one can handle the full scale of ADL syntax (in fact, quantifiers, negations, and disjunctive effect conditions are handled this way already by our current code). Of course, such a compilation process (in particular bringing formulas into DNF) is worst-case exponential and may be prohibitively costly. In classical planning, with some simple optimizations (detecting static predicates, e.g.) the compilation is usually manageable in practice. Whether the same holds true in conformant planning has to be empirically verified.

In a work effort we have already (largely) completed, we extended Conformant-FF to contingent planning, i.e. to handle observations, and to produce plans with branches depending on the observed properties [28]. In the long term, we plan to extend Conformant-FF into a probabilistic planner.

## 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, Daniel Bryce for his help with POND, and Dave Smith for his help in understanding and describing related work. We also thank the anonymous reviewers for their comments, which helped to improve the paper. Ronen Brafman is partially supported by the NASA Intelligent Systems Program, and the Paul Ivanier Center for Robotics and Production Management at Ben-Gurion University. Jörg Hoffmann was, at the time of doing the described work, supported by the DFG (Deutsche Forschungsgemeinschaft), project HEU-PLAN II.

## Appendix A. Proofs

This appendix provides the full proofs to (as well as some discussions related to) the theorems contained in Section 4.

We first show that deciding conformant plan existence with empty delete lists is co-NP-complete, given one assumes the generous action execution semantics, i.e., drops the requirement that the actions in the plan have to be applicable at their point of execution no matter what initial world one starts from. (Remember that in the generous semantics, a non-applicable action does not change the world state, while in the non-generous semantics a non-applicable action leads into an undefined world state.)

**Theorem 1** *($A^+$-complexity). Given a conformant task $(A, \mathcal{I}, G)$ where the delete lists of all effects are empty. Deciding whether there exists a generous plan for $(A, \mathcal{I}, G)$ is co-NP-complete.*

**Proof.** Hardness follows directly from Proposition 1. We show membership by constructing a propositional CNF formula $\phi$ (of polynomial size) so that $\phi$ is unsatisfiable iff there exists a generous plan for $(A, \mathcal{I}, G)$. The idea behind the construction is to encode the semantics of executing all actions in parallel $m$ times, where $m$ is the number of distinct conditional effects in the task. We first explain the construction of the formula $\phi$, then we argue why no more than $m$ parallel action steps are needed to achieve the goal, if that is possible.

The formula $\phi$ is similar to the formulas $\phi(act)$ introduced in Section 3 to encode the semantics of action sequences. We use a time index to differentiate between values of propositions at different points along the hypothetical execution of the parallel action steps. First, as before $\phi$ contains $\mathcal{I}$ indexed with time 0 (i.e., for each clause $l_1 \vee \cdots \vee l_k \in \mathcal{I}$ we add $l_1(0) \vee \cdots \vee l_k(0)$ into $\phi$). Then, for all $0 \leqslant t < m$, for all actions $a$ with precondition $pre(a) = \{p_1, \ldots, p_k\}$, for all effects $e \in E(a)$ with condition $con(e) = \{c_1, \ldots, c_l\}$, and for each $p \in add(e)$, we insert the clause $\neg p_1(t) \vee \cdots \vee \neg p_k(t) \vee \neg c_1(t) \vee \cdots \vee \neg c_l(t) \vee p(t+1)$, i.e. the implication $p_1(t) \wedge \cdots \wedge p_k(t) \wedge c_1(t) \wedge \cdots \wedge c_l(t) \rightarrow p(t+1)$. Also, for all $0 \leqslant t < m$, and for all propositions $p$, we insert the clause $\neg p(t) \vee p(t+1)$, i.e. the implication $p(t) \rightarrow p(t+1)$. In the absence of delete effects, these clauses obviously capture the semantics of applying all actions $m$ times in parallel (with a generous execution model), except that one can make a proposition $p(t+1)$ true "without a reason" because we have not included any clauses that force $p$ to stay false if it was false at $t$ and was not added by any action effect. But it is also possible to let $p$ stay false in this latter case. We now, finally, add the clause $\neg g_1(m) \vee \cdots \vee \neg g_n(m)$ into $\phi$ where $G = \{g_1, \ldots, g_n\}$. The overall formula is then unsatisfiable iff, from every possible initial world state, applying all actions in parallel $m$ times forces the goals to be true.

If we fix any possible initial world state $I$, then no plan needs more than $m$ steps to achieve $G$. This is simply because, without delete lists, if an action application does not result in the application of any conditional effects that have not already been applied earlier in the action sequence, then the action can be skipped because it does not add any new propositions to the world state it is executed in. Now, say the action sequence $act$ is a generous plan for $(A, \mathcal{I}, G)$. Then, for any possible initial world state $I \in 2^{\mathcal{I}}$, $act$ contains a subsequence $act'$ so that executing $act'$ in $I$ achieves $G$. With the above, we can assume that the length of $act'$ is at most $m$. This means that $act'$ is contained in the $m$ parallel action steps encoded in $\phi$, so if one sets the $t = 0$ variables in $\phi$ to the values given by $I$ then one obtains a contradiction to the goals-false clause at $m$. It follows that, if there is a generous plan for $(A, \mathcal{I}, G)$, then $\phi$ is unsatisfiable. The other way around, if $\phi$ is unsatisfiable then obviously a generous plan is to apply all actions in sequence, $m$ times iteratively. This concludes the proof. □

The construction in the proof does not work if one assumes a non-generous execution semantics of actions, because in this case one cannot easily model a "parallel" execution of all actions. The formula $\phi$ in the proof assumes that, at plan execution time, non-applicable actions in the parallel execution do not harm the plan. More formally, unsatisfiability of $\phi$ does not imply that there is a non-generous plan.

We now prove that the CRPG algorithm from Fig. 2 works as required. We use the following notation. Given an action sequence $act = \langle a_{-n}, \ldots, a_{-1} \rangle$, and a CRPG built for $act$ up to proposition layer $m$, for any $t$ with $-n \leqslant t < m$ we denote by *act-CRPG*$(t)$ the action sequence that consists of all action layers $A(-n), \ldots, A(t)$ in an arbitrary linearization; for $t = -n - 1$, the action sequence is assumed empty.

We first prove a lemma containing the main technical argument. Before reading the proof, recall Section 4.2 and Fig. 2. In particular, remember that the action layers contain the actions in their relaxed form (by a relaxation function $|_1^+$), i.e. without delete lists and with all but one of the effect conditions removed. The prove is not overly deep, but a little complicated due to the many details that need to be taken into account.

**Lemma 1.** *Given a conformant task $(A, \mathcal{I}, G)$, an executable action sequence act, and a relaxation function $|_1^+$ for A. For all proposition layers t built by* build-CRPG$(act, A, \mathcal{I}, G, |_1^+)$*, and for all propositions p, p is known after act-CRPG$(t-1)$ iff $p \in P(t)$.*

**Proof.** Because $act$ is executable, $act|_1^+$ also has that property, i.e., when executing $act|_1^+$ in any initial world state $I \in 2^{\mathcal{I}}$, the preconditions of the actions are true at the respective points of execution. We prove the following properties, for all propositions $p$ and for all $-n \leqslant t < m$, by induction over $t$.

(A) For any initial world state $I \in 2^{\mathcal{I}}$, $p$ is true after executing *act-CRPG*$(t-1)$ in $I$ iff either
  (1) there exist $a \in A(t-1)$ and $e \in E(a)$ s.t. $p \in add(e)$ and $con(e) \in P(t-1)$, or
  (2) there exists $l \in Impleafs(p(t))$ s.t. $l \in I$.

(B) $p$ is known after *act-CRPG*$(t-1)$ iff $p \in P(t)$.

(C) $p$ is unknown after *act-CRPG*$(t-1)$ iff $p \in uP(t)$.

Say the length of *act* is $n$. The base case, $t = -n$, is trivially true for all three properties. Assume the properties hold true for some proposition layer $t - 1 \geqslant -n$. Assume build-CRPG$(act, A, \mathcal{I}, G, |_1^+)$ builds the next proposition layer, time step $t$. We show that then the properties hold for that layer, too. We consider the three properties from top to bottom.

*Property* (A)*, from left to right*. Say $p$ is true after *act-CRPG*$(t-1)$. Then $p$ is added by an occurring effect $e$ of an action $a$ at layer $t-1$. That is, we have $a \in A(t-1)$ and $e \in E(a)$ s.t. $p \in add(e)$, and $con(e)$ is true after *act-CRPG*$(t-2)$. If there is one such pair $a$ and $e$ s.t. $con(e) \in P(t-1)$, then property (A)(1) is satisfied and we are done. Otherwise, let $a$ and $e$ be one pair as above. We can conclude two things about $a$ and $e$:

- First, consider the induction hypothesis on property (A) for $con(e)$ and *act-CRPG*$(t-2)$. Because $con(e) \notin P(t-1)$, property (A)(1) does not hold. Thus property (A)(2) must hold, i.e. there exists $l \in Impleafs(con(e)(t-1))$ s.t. $l \in I$.
- Second, consider the induction hypothesis on property (B) for $con(e)$ and *act-CRPG*$(t-2)$. Because $con(e) \notin P(t-1)$, $con(e)$ is not known after *act-CRPG*$(t-2)$. But $con(e)$ is true after *act-CRPG*$(t-2)$, so we have that $con(e)$ is unknown after *act-CRPG*$(t-2)$. We can now apply the induction hypothesis on property (C) for $con(e)$ and *act-CRPG*$(t-2)$, and we get that $con(e) \in uP(t-1)$. Therefore, by construction of the build-CRPG algorithm, there is an edge $(con(e)(t-1), p(t)) \in Imp$.

With $l \in Impleafs(con(e)(t-1))$ and $(con(e)(t-1), p(t)) \in Imp$ we have $l \in Impleafs(p(t))$, so property (A)(2) holds for $p$ and *act-CRPG*$(t-1)$, and we are done.

*Property* (A)*, from right to left*. We first prove that property (A)(1) implies truth of $p$ after *act-CRPG*$(t-1)$, then we do the same for property (A)(2).

- Case (1). Say there exist $a \in A(t-1)$ and $e \in E(a)$ s.t. $p \in add(e)$ and $con(e) \in P(t-1)$. First, observe that the preconditions $pre(a)$ of $a$ are true after *act-CRPG*$(t-2)$. If $t - 1 \geqslant 0$, this follows from induction hypothesis on property (B) for (all) $pre(a)$ and *act-CRPG*$(t-2)$. If $t - 1 < 0$, i.e. if $a$ is one of the actions in the sequence *act* leading to our search state, then the build-CRPG algorithm imposed no restriction on $a$'s preconditions. But *act* is executable by prerequisite, and with the argument given at the top of this proof we have that $pre(a)$ is true at the point of $a$'s execution, i.e., after *act-CRPG*$(t-2)$. So $a$ is applicable. By induction hypothesis on property (B) for $con(e)$ and *act-CRPG*$(t-2)$, we have that $con(e)$ is true after *act-CRPG*$(t-2)$. So $p$ is true after *act-CRPG*$(t-1)$, and we are done.
- Case (2). Say there exists $l \in Impleafs(p(t))$ s.t. $l \in I$. Going one step backwards over the edges in *Imp*, we then get that there exists a $p'$ with $(p'(t-1), p(t)) \in Imp$ and $l \in Impleafs(p'(t-1))$. By induction hypothesis on property (A) for $p'$ and *act-CRPG*$(t-2)$, $p'$ is true after *act-CRPG*$(t-2)$. Now, let $a$ and $e$ be the action and effect responsible for the edge $(p'(t-1), p(t)) \in Imp$, i.e. $a \in A(t-1)$ and $e \in E(a)$ with $p \in add(e)$ and $con(e) = p'$. With the same arguments as above in case (1), we have that $pre(a)$ is true after *act-CRPG*$(t-2)$. Since we have already shown that $con(e) = p$ is also true after *act-CRPG*$(t-2)$, it follows that $p$ is true after *act-CRPG*$(t-1)$, and we are done.

*Property* (B). We prove this by equivalently transforming the left hand side of the property to the right hand side of the property. We first give the sequence of transformation steps, then we explain why these steps are valid.

(a) $p$ is known after *act-CRPG*$(t-1)$ $\Leftrightarrow$
(b) For all $I \in 2^{\mathcal{I}}$, $p$ is true after executing *act-CRPG*$(t-1)$ in $I$ $\Leftrightarrow$
(c) For all $I \in 2^{\mathcal{I}}$, either property (A)(1) holds for $p$ or property (A)(2) holds for $p$ $\Leftrightarrow$
(d) Either property (A)(1) holds for $p$, or for all $I \in 2^{\mathcal{I}}$ property (A)(2) holds for $p$ $\Leftrightarrow$
(e) Either property (A)(1) holds for $p$, or $\mathcal{I} \rightarrow \bigvee_{l \in Impleafs(p(t))} l$ $\Leftrightarrow$
(f) $p \in P(t)$

The step from (a) to (b) just inserts the definition of known propositions. The step from (b) to (c) is valid because we have already proved property (A) for $p$ and *act-CRPG*$(t-1)$. Note that, at this point, we need the induction hypothesis as used to prove property (A). The step from (c) to (d) is valid because property (A)(1) does not depend

on $I$. The step from (d) to (e) is valid because there is a tree leaf $l$ in *every* initial world state iff the initial formula implies the disjunction of the leafs. The step from (e) to (f) is valid by construction of the build-CRPG algorithm: (e) is the precise condition under which the algorithm inserts $p$ into $P(t)$.

*Property* (C)*, from left to right*. Because $p$ is unknown after *act-CRPG*$(t-1)$, there exists $I \in 2^{\mathcal{I}}$ s.t. $p$ is true after *act-CRPG*$(t-1)$. We already proved property (A) for $p$ and *act-CRPG*$(t-1)$, so we get that either property (A)(1) or property (A)(2) holds for $p$ and $I$. We show below that

$(*)$ assuming property (A)(1) leads to a contradiction.

So property (A)(2) must hold. That is, there exists $l \in \mathit{Impleafs}(p(t))$ s.t. $l \in I$. This implies, in particular, that there exists an edge $(p'(t-1), p(t)) \in \mathit{Imp}$. By construction of the build-CRPG algorithm, this implies that either $p \in uP(t)$, or $p \in P(t)$. (The latter can be the case if $p$ was inferred due to $\mathcal{I} \rightarrow \bigvee_{l \in \mathit{Impleafs}(p(t))} l$.) But by the already proved property (B) for $p$ and *act-CRPG*$(t-1)$, we have that $p \notin P(t)$, since it would otherwise be known after *act-CRPG*$(t-1)$. It follows that $p \in uP(t-1)$, and we are done.

It remains to show $(*)$. We have that $p$ is unknown after *act-CRPG*$(t-1)$, and we assume that property (A)(1) holds for $p$ and *act-CRPG*$(t-1)$. That is, there exist $a \in A(t-1)$ and $e \in E(a)$ s.t. $p \in \mathit{add}(e)$ and $\mathit{con}(e) \in P(t-1)$. With exactly the same arguments as given above in the right-to-left proof of Property (A), case (1), we can show that $a$ will be applicable, at its point of execution in *act-CRPG*$(t-1)$, from *every* initial world state. Further, with induction hypothesis on property (B) for $\mathit{con}(e)$ and *act-CRPG*$(t-2)$, we have that $\mathit{con}(e)$ is known after *act-CRPG*$(t-2)$. It follows that $p$ is known after *act-CRPG*$(t-1)$, in contradiction. This concludes the argument.

*Property* (C)*, from right to left*. If $p \in uP(t)$ then, by construction of the build-CRPG algorithm, it follows that there exists a proposition $p' \in uP(t-1)$ with an edge $(p'(t-1), p(t)) \in \mathit{Imp}$. Let $a$ and $e$ be the action and effect responsible for the edge $(p'(t-1), p(t))$, i.e., $a \in A(t-1)$, $e \in E(a)$, $p \in \mathit{add}(e)$, $\mathit{con}(e) = p'$. Observe that $a$ will be applicable, at its point of execution in *act-CRPG*$(t-1)$, when starting from any initial world state: again, this follows by exactly the same arguments as given above in the right-to-left proof of Property (A), case (1). Now, observe that, by induction hypothesis on property (C) for $p'$ and *act-CRPG*$(t-2)$, $p'$ is unknown after *act-CRPG*$(t-2)$. This means in particular that there exists $I \in 2^{\mathcal{I}}$ s.t. $p'$ is true after executing *act-CRPG*$(t-2)$ in $I$. With applicability of $a$, it follows that $p$ is true after executing *act-CRPG*$(t-1)$ in $I$. Thus $p$ is either known or unknown after *act-CRPG*$(t-1)$. Assume $p$ is known after *act-CRPG*$(t-1)$. Then, by the already proved property (B) for $p$ and *act-CRPG*$(t-1)$, we get $p \in P(t)$. But, by construction of the build-CRPG algorithm, $P(t) \cap uP(t) = \emptyset$, in contradiction to our prerequisite $p \in uP(t)$. It follows that $p$ is unknown after *act-CRPG*$(t-1)$. □

Next, we consider the termination criterion in Fig. 2, i.e. the criterion that makes the CRPG algorithm return FALSE. The criterion asks if, in a time step $t$ just built, the sets of known and unknown propositions, as well as the reachable leafs of the *Imp* tree for the latter propositions, have stayed the same from layer $t$ to layer $t+1$. It is relatively easy to see that, in this case, the same would hold true at all future time steps $t' > t$: if there was progress in the future then there would also be some progress from $t$ to $t+1$ already.

**Lemma 2.** *Given a conformant task* $(A, \mathcal{I}, G)$*, an action sequence act, and a relaxation function* $|_1^+$ *for A. If, for proposition layers t and* $t+1$ *built by* build-CRPG$(\mathit{act}, A, \mathcal{I}, G, |_1^+)$*, it holds that* $P(t+1) = P(t)$*,* $uP(t+1) = uP(t)$*, and* $\forall p \in uP(t+1)$: $\mathit{Impleafs}(p(t+1)) = \mathit{Impleafs}(p(t))$*, then the same conditions would hold for layers* $t+1$ *and* $t+2$*.*

**Proof.** Assume build-CRPG$(\mathit{act}, A, \mathcal{I}, G, |_1^+)$ would not return FALSE in iteration $t$, and proceed to iteration $t+1$. With $P(t+1) = P(t)$, $A(t+1) = A(t)$ would hold. With that, with $P(t+1) = P(t)$, and with $uP(t+1) = uP(t)$, the first **for**-loop in the call to build-timestep$(t+1, A(t+1))$ would do exactly the same things as in the previous iteration. Consider the second **for**-loop in the call to build-timestep$(t+1, A(t+1))$. It proceeds over the same propositions $p$ as in iteration $t$. We now show that, for all these $p$, $\mathit{Impleafs}(p(t+2)) = \mathit{Impleafs}(p(t+1))$, which then concludes the argument.

We have $\mathit{Impleafs}(p(t+2)) \supseteq \mathit{Impleafs}(p(t+1))$ due to the NOOP action. To see the opposite direction, say we have a proposition $l$ with $l \in \mathit{Impleafs}(p(t+2))$. Then the path of *Imp* edges from $l$ to $p(t+2)$ goes through an edge

$(p'(t+1), p(t+2)) \in Imp$. Obviously, we have $l \in Impleafs(p'(t+1))$. Now, we can show that the same edge and relation to $l$ exist one step earlier in the CRPG.

(1) Observe that $p' \in uP(t+1)$ – if $p'$ was in $P(t+1)$, then $p$ would be in $P(t+2)$.
(2) Because $A(t+1) = A(t)$, the action $a \in A(t+1)$ responsible for the edge $(p'(t+1), p(t+2))$ also occurs in $A(t)$, yielding the edge $(p'(t), p(t+1)) \in Imp$.
(3) With $p' \in uP(t+1)$, we have $Impleafs(p'(t+1)) = Impleafs(p'(t))$. This yields $l \in Impleafs(p'(t))$.

With $l \in Impleafs(p'(t))$ and $(p'(t), p(t+1)) \in Imp$, we get $l \in Impleafs(p(t+1))$, and are done. □

As a side remark, note that we don't need to postulate executability of *act* for Lemma 2 to hold. Lemmas 1 and 2 together allow the conclusion that the CRPG algorithm is complete, i.e., returns FALSE only if there is no relaxed plan.

**Theorem 2** *(CRPG-completeness). Given a conformant task $(A, \mathcal{I}, G)$, an executable action sequence act, and a relaxation function $|_1^+$ for A. If* build-CRPG$(act, A, \mathcal{I}, G, |_1^+)$ *returns FALSE then there is no relaxed plan for $(A, \mathcal{I}, G)$ that starts with the sequence act.*

**Proof.** Say $act = \langle a_{-n}, \ldots, a_{-1} \rangle$. Say $\langle a_{-n}, \ldots, a_{-1}, a_0, \ldots, a_{m-1} \rangle$ is a relaxed plan for $(A, \mathcal{I}, G)$. We prove below that

$(**)$ if we let the algorithm run until iteration $m$, then $G \subseteq P(m)$ holds.

Once this is shown, we can argue as follows. If the algorithm returns FALSE then it does so before reaching iteration $m$, i.e. in iteration $t < m$, with $G \not\subseteq P(t)$. By Lemma 2 all future iterations $t' > t$ have, in particular, $P(t') = P(t)$. So we get $P(m) = P(t)$, in contradiction with $G \subseteq P(m)$.

It remains to prove $(**)$. Denote, for $-n \leqslant t \leqslant m$, by $P|_{rp}(t)$ the set of propositions that are known after $\langle a_{-n}, \ldots, a_{-1}, a_0, \ldots, a_{t-1} \rangle|_1^+$. We show by a simple induction over $t$ that, for all $t$, $P(t) \supseteq P|_{rp}(t)$. With $t = m$, this suffices because, since $\langle a_{-n}, \ldots, a_{-1}, a_0, \ldots, a_{m-1} \rangle$ is a relaxed plan, the goals are contained in $P|_{rp}(m)$.

Base argument, $t = -n$. Then $\langle a_{-n}, \ldots, a_{-1}, a_0, \ldots, a_{t-1} \rangle$ is empty and the claim is obvious.

Inductive argument, from $t - 1 \Rightarrow t$. First, observe that $a_{t-1} \in A(t-1)$. If $t - 1 < 0$, then this is obvious by construction of the build-CRPG algorithm. If $t - 1 \geqslant 0$ then $a_{t-1} \in A(t-1)$ because $pre(a_{t-1}) \subseteq P|_{rp}(t-1)$ (the actions in the relaxed plan must be applicable), and $P|_{rp}(t-1) \subseteq P(t-1)$ (induction hypothesis). Thus we have that $\langle a_{-n}, \ldots, a_{-1}, a_0, \ldots, a_{t-1} \rangle|_1^+$ is a sub-sequence of $act\text{-}CRPG(t-1)$. But with that, since the relaxation ignores the delete lists, $P|_{rp}(t)$ is a subset of the propositions that are known after $act\text{-}CRPG(t-1)$. The latter set of propositions is, by Lemma 1, equal to $P(t)$. This concludes the argument. □

It is relatively easy to see that the actions selected by the extract-CRPlan procedure do indeed form a relaxed plan for the respective belief state.

**Theorem 3** *(CRPG-soundness). Given a conformant task $(A, \mathcal{I}, G)$, an executable action sequence act, and a relaxation function $|_1^+$ for A, s.t.* build-CRPG$(act, A, \mathcal{I}, G, |_1^+)$ *returns TRUE in iteration m. Let $A(0)^s, \ldots, A(m-1)^s$ be the actions selected from $A(0), \ldots, A(m-1)$ by* extract-CRPlan$(CRPG(act, A, \mathcal{I}, G, |_1^+), G)$. *Then $act|_1^+$ concatenated with $A(0)^s, \ldots, A(m-1)^s$ in an arbitrary linearization is a plan for $(A|_1^+, \mathcal{I}, G)$.*

**Proof.** First, observe that no errors occur in the execution of extract-CRPlan, due to the way the CRPG is computed. During the backwards loop over $t$, for all $g \in G(t)$, by construction of extract-CRPlan we have $g \in P(t)$. So by construction of build-CRPG either we have $\exists a \in A(t-1), e \in E(a), con(e) \in P(t-1), g \in add(e)$, or we have $\mathcal{I} \rightarrow \bigvee_{l \in Impleafs(g(t))} l$. Also, during the loop, for all actions $a \in A(t)$, by construction of build-CRPG we have that all of $a$'s preconditions are contained in $P(t)$ and thus in some $P(t')$, $t' \leqslant t$.

Denote, for $0 \leqslant t < m$, by $act\text{-}CRPG(t)^s$ the concatenation of $act|_1^+$ with $A(0)^s, \ldots, A(t)^s$ in an arbitrary linearization. We prove that, for any fact $g$ that is made a sub-goal at time $t$ $(0 < t \leqslant m)$ during the execution of extract-CRPlan,

$g$ is known after *act-CRPG*$(t-1)^s$. This shows the claim: the goals are known after *act-CRPG*$(m-1)^s$. The proof proceeds by induction over $t$. Before we start, note that with $P(t-1) \subseteq P(t)$ for all $t$ (due to the NOOP action), we have that all $p$ that are contained in a set $P$ on which sub-goal is called, but for which $t_0 \leqslant 0$, are contained in $P(0)$. With Lemma 1 we then have that all these $p$ are known after $act|_1^+$.

Base argument, $t = 1$. We distinguish two cases.

- First case, say there exist $a \in A(0), e \in E(a), con(e) \in P(0), g \in add(e)$. Then $a \in A(0)^s$ for one such $a$ and $e$, and because, as argued above, $a$'s preconditions are known after $act|_1^+$, and $con(e)$ is known after $act|_1^+$, we are done.
- In the other case, actions $a$ were selected from $A(0)$ for all effects $e$ that were responsible for an edge in a path from $l(-n), l \in L$, to $g(1)$, where $\mathcal{I} \to \bigvee_{l \in L} l$. Let $I$ be an initial world state, $I \in 2^{\mathcal{I}}$. Then $l \in I$ for some $l \in L$. There is a path in *Imp* from $l(-n)$ to $g(1)$. For all effects responsible for an edge on that path, the respective action $a$ is in *act-CRPG*$(0)^s$. Because *act* is executable by prerequisite, and because $pre(a) \subseteq P(0)$ for $a \in A(0)$, all actions are applicable at their point of execution in *act-CRPG*$(0)^s$. By construction of *Imp*, and because $l \in I$, it follows that all effects on the path from $l(-n)$ to $g(1)$ occur. So $g$ is true after executing *act-CRPG*$(0)^s$ in $I$. Since this holds for all $I \in 2^{\mathcal{I}}$, we are done.

Inductive argument, $t-1 \Rightarrow t$. The proof is the same as above for the base argument, except that now we can and must use the induction hypothesis to see that the preconditions of the selected actions $a$, as well as (if that applies) the effect condition $con(e)$—which are all made sub-goals at layers $t' < t$—will be true at the point of execution of $a$ in *act-CRPG*$(t-1)^s$. In more detail, say $g$ was made a sub-goal at $t$.

- First case, say there exist $a \in A(t-1), e \in E(a), con(e) \in P(t-1), g \in add(e)$. Then $a \in A(t-1)^s$ for one such $a$ and $e$. By construction of extract-CRPlan, $pre(a)$ and $con(a)$ were made sub-goals at layers $t' \leqslant t-1$. By induction hypothesis, these propositions are thus known after the respective action sequences *act-CRPG*$(t'-1)^s$. So they are also known after the longer sequence *act-CRPG*$(t-2)^s$. It follows that $a$ is applicable when executed in *act-CRPG*$(t-1)^s$, and that $e$ occurs. So, as required, $g$ is known after *act-CRPG*$(t-1)^s$.
- In the other case, actions $a$ were selected from $A(0), \ldots, A(t-1)$ for all effects $e$ that were responsible for an edge in a path from $l(-n), l \in L$, to $g(t)$, where $\mathcal{I} \to \bigvee_{l \in L} l$. Let $I \in 2^{\mathcal{I}}$, and $l \in I \cap L$. There is a path in *Imp* from $l(-n)$ to $g(t)$. For all effects responsible for an edge on that path, the respective action $a$ is in *act-CRPG*$(t-1)^s$. Say $a$ is selected at time $t'$ ($t' \leqslant t-1$ holds). By construction of extract-CRPlan, $a$'s preconditions were made sub-goals at layers $t'' \leqslant t'$, and, as above in the first case, by induction hypothesis we know that $a$ is applicable when executing *act-CRPG*$(t-1)^s$. With $l \in I$ the effects on the path from $l(-n)$ to $g(t)$ will occur, giving us that $g$ is be true after *act-CRPG*$(0)^s$. This holds for all $I \in 2^{\mathcal{I}}$, and we are done. □

## References


[1] B. Bonet, H. Geffner, Planning with incomplete information as heuristic search in belief space, in: S. Chien, R. Kambhampati, C. Knoblock (Eds.), Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00), AAAI Press, Menlo Park, CA, 2000, pp. 52–61.

[2] A. Cimatti, M. Roveri, Conformant planning via symbolic model checking, JAIR 13 (2000) 305–338.

[3] P. Bertoli, A. Cimatti, Improving heuristics for planning as search in belief space, in: [30], pp. 143–152.

[4] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (3) (1960) 201–215.

[5] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, Comm. ACM 5 (7) (1962) 394–397.

[6] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, J. Artificial Intelligence Res. 14 (2001) 253–302.

[7] D. Bryce, S. Kambhampati, D.E. Smith, Planning in belief space with a labelled uncertainty graph, in: Proc. AAAI-04 Workshop on Learning and Planning in Markov Decision Processes, 2004.

[8] P. Ferraris, E. Giunchiglia, Planning as satisfiability in nondeterministic domains, in: Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI-00), Austin, TX, MIT Press, Cambridge, MA, 2000, pp. 748–753.

[9] E. Giunchiglia, Planning as satisfiability with expressive action languages: Concurrency, constraints, and nondeterminism, in: A. Cohn, F. Giunchiglia, B. Selman (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the 7th International Conference (KR-00), Breckenridge, CO, Morgan Kaufmann, San Mateo, CA, 2000.

[10] J. McCarthy, P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer, D. Michie (Eds.), Machine Intelligence, vol. 4, Edinburgh University Press, Edinburgh, UK, 1969, pp. 463–502.

[11] T. Bylander, The computational complexity of propositional STRIPS planning, Artificial Intelligence 69 (1–2) (1994) 165–204.
[12] H. Turner, Polynomial-length planning spans the polynomial hierarchy, in: S. Flesca, S. Greco, N. Leone, G. Ianni (Eds.), Logics in Artificial Intelligence—8th European Conference JELIA-02, Cosenza, Italy, Springer-Verlag, Berlin, 2002, pp. 111–124.
[13] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence 129 (1–2) (2001) 5–33.
[14] L. Zhang, S. Malik, Extracting small unsatisfiable cores from unsatisfiable boolean formulas, in: Proc. SAT-03, 2003.
[15] A.L. Blum, M.L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1–2) (1997) 279–298.
[16] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: Proceedings of the 38th Design Automation Conference (DAC-01), 2001, http://www.ee.princeton.edu/~chaff/DAC2001v56.pdf.
[17] R. Brafman, A simplifier for propositional formulas with many binary clauses, in: [31], pp. 515–520.
[18] R.P.A. Petrick, F. Bacchus, A knowledge-based approach to planning with incomplete information and sensing, in: [30], pp. 212–221.
[19] A Cimatti, M. Roveri, P. Bertoli, Conformant planning via symbolic model checking and heuristic search, Artificial Intelligence 159 (1–2) (2004) 127–206.
[20] J. Slaney, S. Thiebaux, Blocks world revisited, Artificial Intelligence 125 (2001) 119–153.
[21] D.E. Smith, D. Weld, Conformant Graphplan, in: Proceedings of the 15th National Conference of the American Association for Artificial Intelligence (AAAI-98), Madison, WI, MIT Press, Cambridge, MA, 1998, pp. 889–896.
[22] J. Kurien, P.P. Nayak, D.E. Smith, Fragment-based conformant planning, in: [30], pp. 153–162.
[23] A. Cimatti, M. Roveri, P. Bertoli, Heuristic search + symbolic model checking = efficient conformant planning, in: [31], pp. 467–472.
[24] D. Bryce, S. Kambhampari, Heuristic guidance measures for conformant planning, in: [29], pp. 365–374.
[25] J. Rintanen, Constructing conditional plans by a theorem-prover, J. Artificial Intelligence Res. 10 (1999) 323–352.
[26] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A logic programming approach to knowledge-state planning, II: The DLVK system, Artificial Intelligence 144 (1–2) (2003) 157–211.
[27] B.C. Gazen, C. Knoblock, Combining the expressiveness of UCPOP with the efficiency of Graphplan, in: S. Steel, R. Alami (Eds.), Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97), in: Lecture Notes in Artificial Intelligence, vol. 1348, Toulouse, France, Springer-Verlag, Berlin, 1997, pp. 221–233.
[28] J. Hoffmann, R. Brafman, Contingent planning via heuristic forward search with implicit belief states, in: S. Biundo, K. Myers, K. Rajan (Eds.), Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05), Monterey, CA, Morgan Kaufmann, San Mateo, CA, 2005, in press.
[29] S. Koenig, S. Zilberstein, J. Koehler (Eds.), Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04), Whistler, Canada, Morgan Kaufmann, San Mateo, CA, 2004.
[30] M. Ghallab, J. Hertzberg, P. Traverso (Eds.), Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02), Toulouse, France, Morgan Kaufmann, San Mateo, CA, 2002.
[31] B. Nebel (Ed.), Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01), Seattle, Washington, Morgan Kaufmann, San Mateo, CA, 2001.