Journal of Artificial Intelligence Research 30 (2007) 565-620 Submitted 3/07; published 12/07

Probabilistic Planning via Heuristic Forward Search and Weighted Model Counting

Carmel Domshlak DCARMEL@IE.TECHNION.AC.IL
Technion - Israel Institute of Technology,
Haifa, Israel
Jörg Hoffmann JOERG.HOFFMANN@DERI.AT
University of Innsbruck, DERI,
Innsbruck, Austria

Abstract

We present a new algorithm for probabilistic planning with no observability. Our algorithm, called Probabilistic-FF, extends the heuristic forward-search machinery of Conformant-FF to problems with probabilistic uncertainty about both the initial state and action effects. Specifically, Probabilistic-FF combines Conformant-FF's techniques with a powerful machinery for weighted model counting in (weighted) CNFs, serving to elegantly define both the search space and the heuristic function. Our evaluation of Probabilistic-FF shows its fine scalability in a range of probabilistic domains, constituting a several orders of magnitude improvement over previous results in this area. We use a problematic case to point out the main open issue to be addressed by further research.

1. Introduction

In this paper we address the problem of *probabilistic planning with no observability* (Kushmerick, Hanks, & Weld, 1995), also known in the AI planning community as conditional (Majercik & Littman, 2003) or conformant (Hyafil & Bacchus, 2004) probabilistic planning. In such problems we are given an initial belief state in the form of a probability distribution over the world states W, a set of actions (possibly) having probabilistic effects, and a set of alternative goal states $W_G \subseteq W$. A solution to such a problem is a single sequence of actions that transforms the system into one of the goal states with probability exceeding a given threshold θ. The basic assumption of the problem is that the system cannot be observed at the time of plan execution. Such a setting is useful in controlling systems with uncertain initial state and non-deterministic actions, if sensing is expensive or unreliable. Non-probabilistic conformant planning may fail due to non-existence of a plan that achieves the goals with 100% certainty. Even if there is such a plan, that plan does not necessarily contain information about what actions are most useful to achieve only the requested threshold θ.

The state-of-the-art performance of probabilistic planners has been advancing much more slowly than that of deterministic planners, scaling from 5-10 step plans for problems with $\approx$20 world states to 15-20 step plans for problems with $\approx$100 world states (Kushmerick et al., 1995; Majercik & Littman, 1998; Hyafil & Bacchus, 2004). Since probabilistic planning is inherently harder than its deterministic counterpart (Littman, Goldsmith, & Mundhenk, 1998), such a difference in evolution rates is by itself not surprising. However, recent developments in the area (Onder, Whelan, & Li, 2006; Bryce, Kambhampati, & Smith, 2006; Huang, 2006), and in particular our work here, show that dramatic improvements in probabilistic planning *can* be obtained.

In this paper we introduce Probabilistic-FF, a new probabilistic planner based on *heuristic forward search* in the space of *implicitly represented* probabilistic belief states. The planner is a natural extension of the recent (non-probabilistic) conformant planner Conformant-FF (Hoffmann & Brafman, 2006). The main trick is to replace Conformant-FF's SAT-based techniques with a recent powerful technique for probabilistic reasoning by weighted model counting (WMC) in propositional CNFs (Sang, Beame, & Kautz, 2005). In more detail, Conformant-FF does a forward search in a belief space in which each belief state corresponds to a set of world states considered to be possible. The main trick of Conformant-FF is the use of CNF formulas for an implicit representation of belief states. Implicit, in this context, means that formulas $\phi(\overline{a})$ encode the semantics of executing action sequence $\overline{a}$ in the initial belief state, with propositional variables corresponding to facts with time-stamps. Any actual knowledge about the belief states has to be (and can be) inferred from these formulas. Most particularly, a fact p is known to be true in a belief state if and only if $\phi(\overline{a}) \rightarrow p(m)$, where m is the time endpoint of the formula. The only knowledge computed by Conformant-FF about belief states are these *known facts*, as well as (symmetrically) the facts that are known to be false. This suffices to do STRIPS-style planning, that is, to determine applicable actions and goal belief states. In the heuristic function, FF's (Hoffmann & Nebel, 2001) relaxed planning graph technique is enriched with approximate SAT reasoning.

The basic ideas underlying Probabilistic-FF are:

(i) Define time-stamped Bayesian networks (BNs) describing probabilistic belief states.

(ii) Extend Conformant-FF's belief state CNFs to model these BNs.

(iii) In addition to the SAT reasoning used by Conformant-FF, use weighted model-counting to determine whether the probability of the (unknown) goals in a belief state is high enough.

(iv) Introduce approximate probabilistic reasoning into Conformant-FF's heuristic function.

Note the synergetic effect: Probabilistic-FF re-uses all of Conformant-FF's technology to recognize facts that are true or false with probability 1. This fully serves to determine applicable actions, as well as detect whether part of the goal is already known. In fact, it is as if Conformant-FF's CNF-based techniques were specifically made to suit the probabilistic setting: while without probabilities one could imagine successfully replacing the CNFs with BDDs, *with* probabilities this seems much more problematic.

The algorithms we present cover probabilistic initial belief states given as Bayesian networks, deterministic and probabilistic actions, conditional effects, and standard action preconditions. Our experiments show that our approach is quite effective in a range of domains. In contrast to the SAT and CSP based approaches mentioned above (Majercik & Littman, 1998; Hyafil & Bacchus, 2004), Probabilistic-FF can find *100-step plans for problem instances with billions of world states*. However, such a comparison is not entirely fair due to the different nature of the results provided; the SAT and CSP based approaches provide guarantees on the length of the solution. The approach most closely related to Probabilistic-FF is implemented in POND (Bryce et al., 2006): this system, like Probabilistic-FF, does conformant probabilistic planning for a threshold θ, using a non-admissible, planning-graph based heuristic to guide the search. Hence a comparison between Probabilistic-FF and POND is fair, and in our experiments we perform a comparative evaluation of Probabilistic-FF and POND. While the two approaches are related, there are significant differences in the search space representation, as well as in the definition and computation of the heuristic function.[1] We run the two approaches on a range of domains partly taken from the probabilistic planning literature, partly obtained by enriching conformant benchmarks with probabilities, and partly obtained by enriching classical benchmarks with probabilistic uncertainty. In almost all cases, Conformant-FF outperforms POND by at least an order of magnitude. We make some interesting observations regarding the behavior of the two planners; in particular we identify a domain – derived from the classical Logistics domain – where both approaches fail to scale. The apparent reason is that neither approach is good enough at detecting how many times, at an early point in the plan, a probabilistic action must be applied in order to sufficiently support a high goal threshold at the end of the plan. Devising methods that are better in this regard is the most pressing open issue in this line of work.

The paper is structured as follows. The next section provides the technical background, formally defining the problem we address and illustrating it with our running example. Section 3 details how probabilistic belief states are represented as time-stamped Bayesian networks, how these Bayesian networks are encoded as weighted CNF formulas, and how the necessary reasoning is performed on this representation. Section 4 explains and illustrates our extension of Conformant-FF's heuristic function to the probabilistic settings. Section 5 provides the empirical results, and Section 6 concludes. All proofs are moved into Appendix A.

2. Background

The probabilistic planning framework we consider adds probabilistic uncertainty to a subset of the classical ADL language, namely (sequential) STRIPS with conditional effects. Such STRIPS planning tasks are described over a set of propositions $\mathcal{P}$ as triples (A, I, G), corresponding to the *action set*, *initial world state*, and *goals*. I and G are sets of propositions, where I describes a concrete initial state w_I, while G describes the set of goal states $w \supseteq G$. Actions a are pairs $(pre(a), E(a))$ of the *precondition* and the *(conditional) effects*. A conditional effect e is a triple $(con(e), add(e), del(e))$ of (possibly empty) proposition sets, corresponding to the effect's *condition*, *add*, and *delete* lists, respectively. The precondition $pre(a)$ is also a proposition set, and an action a is *applicable* in a world state w if $w \supseteq pre(a)$. If a is not applicable in w, then the result of applying a to w is undefined. If a is applicable in w, then all conditional effects $e \in E(a)$ with $w \supseteq con(e)$ occur. Occurrence of a conditional effect e in w results in the world state $w \cup add(e) \setminus del(e)$.

If an action a is applied to w, and there is a proposition q such that $q \in add(e) \cap del(e')$ for (possibly the same) occurring $e, e' \in E(a)$, then the result of applying a in w is undefined. Thus, we require the actions to be not self-contradictory, that is, for each $a \in A$, and every $e, e' \in E(a)$, if there exists a world state $w \supseteq con(e) \cup con(e')$, then $add(e) \cap del(e') = \emptyset$. Finally, an action sequence $\overline{a}$ is a *plan* if the world state that results from iterative execution of $\overline{a}$'s actions, starting in w_I, leads to a *goal state* $w \supseteq G$.

2.1 Probabilistic Planning

Our probabilistic planning setting extends the above with (i) probabilistic uncertainty about the initial state, and (ii) actions that can have probabilistic effects. In general, probabilistic planning

1. POND does not use implicit belief states, and the probabilistic part of its heuristic function uses sampling techniques, rather than the probabilistic reasoning techniques we employ.

tasks are quadruples (A, b_I, G, θ), corresponding to the *action set*, *initial belief state*, *goals*, and *acceptable goal satisfaction probability*. As before, G is a set of propositions. The initial state is no longer assumed to be known precisely. Instead, we are given a probability distribution over the world states, b_I, where $b_I(w)$ describes the likelihood of w being the initial world state.

Similarly to classical planning, actions $a \in A$ are pairs $(pre(a), E(a))$, but the effect set $E(a)$ for such a has richer structure and semantics. Each $e \in E(a)$ is a pair $(con(e), \Lambda(e))$ of a propositional condition and a set of *probabilistic outcomes*. Each probabilistic outcome $\varepsilon \in \Lambda(e)$ is a triplet $(Pr(\varepsilon), add(\varepsilon), del(\varepsilon))$, where *add* and *delete* lists are as before, and $Pr(\varepsilon)$ is the probability that outcome ε occurs as a result of effect e. Naturally, we require that probabilistic effects define probability distributions over their outcomes, that is, $\sum_{\varepsilon \in \Lambda(e)} Pr(\varepsilon) = 1$. The special case of deterministic effects e is modeled this way via $\Lambda(e) = \{\varepsilon\}$ and $Pr(\varepsilon) = 1$. Unconditional actions are modeled as having a single effect e with $con(e) = \emptyset$. As before, if a is not applicable in w, then the result of applying a to w is undefined. Otherwise, if a is applicable in w, then there exists exactly one effect $e \in E(a)$ such that $con(e) \subseteq w$, and for each $\varepsilon \in \Lambda(e)$, applying a to w results in $w \cup add(\varepsilon) \setminus del(\varepsilon)$ with probability $Pr(\varepsilon)$. The likelihood $[b, a]\,(w')$ of a world state w' in the belief state $[b, a]$, resulting from applying a probabilistic action a in b, is given by

$$[b,a]\,(w') = \sum_{w \supseteq pre(a)} b(w) \sum_{\varepsilon \in \Lambda(e)} Pr(\varepsilon) \cdot \delta\left(w' = w \cup s \setminus s', s \subseteq add(\varepsilon), s' \subseteq del(\varepsilon)\right), \quad (1)$$

where e is the effect of a such that $con(e) \subseteq w$, and $\delta(\cdot)$ is the Kronecker step function that takes the value 1 if the argument predicate evaluates to TRUE, and 0 otherwise.

Our formalism covers all the problem-description features supported by the previously proposed formalisms for conformant probabilistic planning (Kushmerick et al., 1995; Majercik & Littman, 1998; Hyafil & Bacchus, 2004; Onder et al., 2006; Bryce et al., 2006; Huang, 2006), and it corresponds to what is called Unary Nondeterminism (1ND) normal form (Rintanen, 2003). We note that there are more succinct forms for specifying probabilistic planning problems (Rintanen, 2003), yet 1ND normal form appears to be most intuitive from the perspective of knowledge engineering.

Example 1 Say we have a robot and a block that physically can be at one of two locations. This information is captured by the propositions r_1, r_2 for the robot, and b_1, b_2 for the block, respectively. The robot can either move from one location to another, or do it while carrying the block. If the robot moves without the block, then its move is guaranteed to succeed. This provides us with a pair of symmetrically defined deterministic actions $\{move\text{-}right, move\text{-}left\}$. The action $move\text{-}right$ has an empty precondition, and a single conditional effect $e = (\{r_1\}, \{\varepsilon\})$ with $Pr(\varepsilon) = 1$, $add(\varepsilon) = \{r_2\}$, and $del(\varepsilon) = \{r_1\}$. If the robot tries to move while carrying the block, then this move succeeds with probability 0.7, while with probability 0.2 the robot ends up moving without the block, and with probability 0.1 this move of the robot fails completely. This provides us with a pair of (again, symmetrically defined) probabilistic actions $\{move\text{-}b\text{-}right, move\text{-}b\text{-}left\}$. The action $move\text{-}b\text{-}right$ has an empty precondition, and two conditional effects specified as in Table 1.

Having specified the semantics and structure of all the components of (A, b_I, G, θ) but θ, we are now ready to specify the actual task of probabilistic planning in our setting. Recall that our actions transform probabilistic belief states to belief states. For any action sequence $\overline{a} \in A^*$, and any belief

$E(a)$	$con(e)$	$\Lambda(e)$	$Pr(\varepsilon)$	$add(\varepsilon)$	$del(\varepsilon)$
e	$r_1 \wedge b_1$	ε_1	0.7	$\{r_2, b_2\}$	$\{r_1, b_1\}$
		ε_2	0.2	$\{r_2\}$	$\{r_1\}$
		ε_3	0.1	$\emptyset$	$\emptyset$
e'	$\neg r_1 \vee \neg b_1$	ε'_1	1.0	$\emptyset$	$\emptyset$

Table 1: Possible effects and outcomes of the action $move\text{-}b\text{-}right$ in Example 1.

state b, the new belief state $[b, \overline{a}]$ resulting from applying $\overline{a}$ at b is given by

$$[b, \overline{a}] = \begin{cases} b, & \overline{a} = \langle\rangle \\ [b, a], & \overline{a} = \langle a \rangle, a \in A \\ [[b, a], \overline{a}'], & \overline{a} = \langle a \rangle \cdot \overline{a}', a \in A, \overline{a}' \neq \emptyset \end{cases} . \tag{2}$$

In such setting, achieving G with certainty is typically unrealistic. Hence, θ specifies the required *lower bound* on the probability of achieving G. A sequence of actions $\overline{a}$ is called a *plan* if we have $b_{\overline{a}}(G) \geq \theta$ for the belief state $b_{\overline{a}} = [b_I, \overline{a}]$.

2.2 Specifying the Initial Belief State

Considering the initial belief state, practical considerations force us to limit our attention only to compactly representable probability distributions b_I. While there are numerous alternatives for compact representation of structured probability distributions, Bayes networks (BNs) (Pearl, 1988) to date is by far the most popular such representation model.[2] Therefore, in Probabilistic-FF we assume that the initial belief state b_I is described by a BN $\mathcal{N}_{b_I}$ over our set of propositions $\mathcal{P}$.

As excellent introductions to BNs abound (e.g., see Jensen, 1996), it suffices here to briefly define our notation. A BN $\mathcal{N} = (\mathcal{G}, \mathcal{T})$ represents a probability distribution as a directed acyclic graph $\mathcal{G}$, where its set of nodes $\mathcal{X}$ stands for random variables (assumed discrete in this paper), and $\mathcal{T}$, a set of tables of conditional probabilities (CPTs)—one table T_X for each node $X \in \mathcal{X}$. For each possible value $x \in Dom(X)$ (where $Dom(X)$ denotes the domain of X), the table T_X lists the probability of the event $X = x$ given each possible value assignment to all of its immediate ancestors (parents) $Pa(X)$ in $\mathcal{G}$. Thus, the table size is exponential in the in-degree of X. Usually, it is assumed either that this in-degree is small (Pearl, 1988), or that the probabilistic dependence of X on $Pa(X)$ induces a significant local structure allowing a compact representation of T_X (Shimony, 1993, 1995; Boutilier, Friedman, Goldszmidt, & Koller, 1996). (Otherwise, representation of the distribution as a BN would not be a good idea in the first place.) The joint probability of a complete assignment ϑ to the variables $\mathcal{X}$ is given by the product of $|\mathcal{X}|$ terms taken from the respective CPTs (Pearl, 1988):

$$Pr(\vartheta) = \prod_{X \in \mathcal{X}} Pr\left(\vartheta[X] \mid \vartheta[Pa(X)]\right) = \prod_{X \in \mathcal{X}} T_X\left(\vartheta[X] \mid \vartheta[Pa(X)]\right),$$

where $\vartheta[\cdot]$ stands for the partial assignment provided by ϑ to the corresponding subset of $\mathcal{X}$.

2. While BNs are our choice here, our framework can support other models as well, e.g. stochastic decision trees.

In Probabilistic-FF we allow $\mathcal{N}_{b_I}$ to be described over the multi-valued variables underlying the planning problem. This significantly simplifies the process of specifying $\mathcal{N}_{b_I}$ since the STRIPS propositions $\mathcal{P}$ do not correspond to the true random variables underlying problem specification.[3] Specifically, let $\bigcup_{i=1}^{k} \mathcal{P}_i$ be a partition of $\mathcal{P}$ such that each proposition set $\mathcal{P}_i$ uniquely corresponds to the domain of a multi-valued variable underlying our problem. That is, for every world state w and every $\mathcal{P}_i$, if $|\mathcal{P}_i| > 1$, then there is *exactly one* proposition $q \in \mathcal{P}_i$ that holds in w. The variables of the BN $\mathcal{N}_{b_I}$ describing our initial belief state b_I are $\mathcal{X} = \{X_1, \dots, X_k\}$, where $Dom(X_i) = \mathcal{P}_i$ if $|\mathcal{P}_i| > 1$, and $Dom(X_i) = \{q, \neg q\}$ if $\mathcal{P}_i = \{q\}$.

Example 2 For an illustration of such $\mathcal{N}_{b_I}$, consider our running example, and say the robot is known to be initially at one of the two possible locations with probability $Pr(r_1) = 0.9$ and $Pr(r_2) = 0.1$. Suppose there is a correlation in our belief about the initial locations of the robot and the block. We believe that, if the robot is at r_1, then $Pr(b_1) = 0.7$ (and $Pr(b_2) = 0.3$), while if the robot is at r_2, then $Pr(b_1) = 0.2$ (and $Pr(b_2) = 0.8$). The initial belief state BN $\mathcal{N}_{b_I}$ is then defined over two variables R ("robot") and B ("block") with $Dom(R) = \{r_1, r_2\}$ and $Dom(B) = \{b_1, b_2\}$, respectively, and it is depicted in Figure 1.

r_1	r_2
0.9	0.1

$$R \longrightarrow B$$

	b_1	b_2
r_1	0.7	0.3
r_2	0.2	0.8

Figure 1: Bayes network $\mathcal{N}_{b_I}$ for Example 1.

It is not hard to see that our STRIPS-style actions $a \in A$ can be equivalently specified in terms of the multi-valued variables $\mathcal{X}$. Specifically, if $|\mathcal{P}_i| > 1$, then no action a can add a proposition $q \in \mathcal{P}_i$ without deleting some other proposition $q' \in \mathcal{P}_i$, and thus, we can consider a as setting $X_i = q$. If $|\mathcal{P}_i| = 1$, then adding and deleting $q \in \mathcal{P}_i$ has the standard semantics of setting $X_i = q$ and $X_i = \neg q$, respectively. For simplicity of presentation, we assume that our actions are not self-contradictory at the level of $\mathcal{X}$ as well—if two conditional effects $e, e' \in E(a)$ can possibly occur in some world state w, then the subsets of $\mathcal{X}$ affected by these two effects have to be disjoint. Finally, our goal G directly corresponds to a partial assignment to $\mathcal{X}$ (unless our G is self-contradictory, requiring $q \wedge q'$ for some $q, q' \in \mathcal{P}_i$.)

3. Belief States

In this section, we explain our representation of, and reasoning about, belief states. We first explain how probabilistic belief states are represented as time-stamped BNs, then we explain how those BNs are encoded and reasoned about in the form of weighted CNF formulas. This representation of belief states by weighted CNFs is then illustrated on the belief state from our running example in Figure 2. We finally provide the details about how this works in Probabilistic-FF.

3. Specifying $\mathcal{N}_{b_I}$ directly over $\mathcal{P}$ would require identifying the multi-valued variables anyway, followed by connecting all the propositions corresponding to a multi-valued variable by a complete DAG, and then normalizing the CPTs of these propositions in a certain manner.

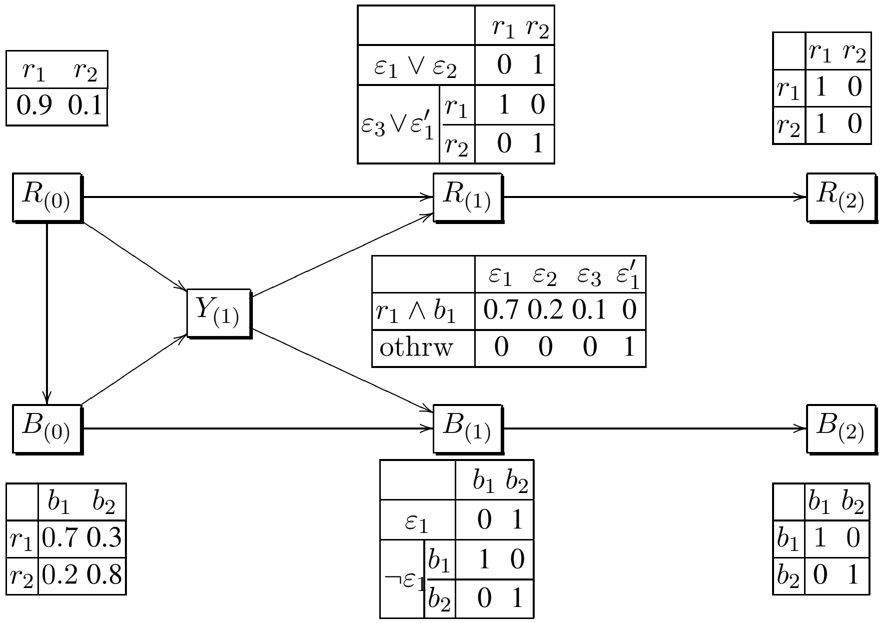


Figure 2: Bayes network $\mathcal{N}_{b_{\overline{a}}}$ for our running Example 1-2 and action sequence $\overline{a} = \langle move\text{-}b\text{-}right, move\text{-}left \rangle$.

3.1 Bayesian Networks

Probabilistic-FF performs a forward search in a space of belief states. The search states are belief states (that is, probability distributions over the world states w), and the search is restricted to belief states reachable from the initial belief state b_I through some sequences of actions $\overline{a}$. A key decision one should make is the actual representation of the belief states. Let b_I be our initial belief state captured by the BN $\mathcal{N}_{b_I}$, and let $b_{\overline{a}}$ be a belief state resulting from applying to b_I a sequence of actions $\overline{a}$. One of the well-known problems in the area of decision-theoretic planning is that the description of $b_{\overline{a}}$ directly over the state variables $\mathcal{X}$ becomes less and less structured as the number of (especially stochastic) actions in $\overline{a}$ increases. To overcome this limitation, we represent belief states $b_{\overline{a}}$ as a BN $\mathcal{N}_{b_{\overline{a}}}$ that *explicitly* captures the sequential application of $\overline{a}$ starting from b_I, trading the representation size for the cost of inference, compared to representing belief states directly as distributions over world states. Below we formally specify the structure of such a BN $\mathcal{N}_{b_{\overline{a}}}$, assuming that all the actions $\overline{a}$ are applicable in the corresponding belief states of their application, and later showing that Probabilistic-FF makes sure this is indeed the case. We note that these belief-state BNs are similar in spirit and structure to those proposed in the AI literature for verifying that a probabilistic plan achieves its goals with a certain probability (Dean & Kanazawa, 1989; Hanks & McDermott, 1994; Kushmerick et al., 1995).

Figure 2 illustrates the construction of $\mathcal{N}_{b_{\overline{a}}}$ for our running example with $\overline{a} = \langle move\text{-}b\text{-}right, move\text{-}left \rangle$. In general, let $\overline{a} = \langle a^1, \dots, a^m \rangle$ be a sequence of actions, numbered according to their appearance on $\overline{a}$. For $0 \leq t \leq m$, let $\mathcal{X}_{(t)}$ be a replica of our state variables $\mathcal{X}$, with $X_{(t)} \in \mathcal{X}_{(t)}$

corresponding to $X \in \mathcal{X}$. The variable set of $\mathcal{N}_{b_{\overline{a}}}$ is the union of $\mathcal{X}_{(0)}, \ldots, \mathcal{X}_{(m)}$, plus some additional variables that we introduce for the actions in $\overline{a}$.

First, for each $X_{(0)} \in \mathcal{X}_{(0)}$, we set the parents $Pa(X_{(0)})$ and conditional probability tables $T_{X_{(0)}}$ to simply copy these of the state variable X in $\mathcal{N}_{b_I}$. Now, consider an action a^t from $\overline{a}$, and let $a^t = a$. For each such action we introduce a discrete variable $Y_{(t)}$ that "mediates" between the variable layers $\mathcal{X}_{(t-1)}$ and $\mathcal{X}_{(t)}$. The domain of $Y_{(t)}$ is set to $Dom(Y_{(t)}) = \bigcup_{e \in E(a)} \Lambda(e)$, that is, to the union of probabilistic outcomes of all possible effects of a. The parents of $Y_{(t)}$ in $\mathcal{N}_{b_{\overline{a}}}$ are set to

$$Pa(Y_{(t)}) = \bigcup_{e \in E(a)} \left\{ X_{(i-1)} \mid con(e) \cap Dom(X) \neq \emptyset \right\}, \tag{3}$$

and, for each $\pi \in Dom(Pa(Y_{(t)}))$, we set

$$T_{Y(t)}(Y_{(i)} = \varepsilon \mid \pi) = \begin{cases} Pr(\varepsilon), & con\,(e(\varepsilon)) \subseteq \pi \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where $e(\varepsilon)$ denotes the effect e of a such that $\varepsilon \in \Lambda(e)$.

We refer to the set of all such variables $Y_{(t)}$ created for the actions of $\overline{a}$ as $\mathcal{Y}$. Now, let $E_X(a) \subseteq E(a)$ be the probabilistic effects of a that affect a variable $X \in \mathcal{X}$. If $E_X(a) = \emptyset$, then we set $Pa(X_{(t)}) = \{X_{(t-1)}\}$, and

$$T_{X(t)}(X_{(t)} = x \mid X_{(t-1)} = x') = \begin{cases} 1, & x = x', \\ 0, & \text{otherwise} \end{cases}. \tag{5}$$

Otherwise, if $E_X(a) \neq \emptyset$, let $x_\varepsilon \in Dom(X)$ be the value provided to X by $\varepsilon \in \Lambda(e)$, $e \in E_X(a)$. Recall that the outcomes of effects $E(a)$ are *all* mutually exclusive. Hence, we set $Pa(X_{(t)}) = \{X_{(t-1)}, Y_{(t-1)}\}$, and

$$T_{X_{(i)}}(X_{(i)} = x \mid X_{(i-1)} = x', Y_{(i-1)} = \varepsilon) = \begin{cases} 1, & e(\varepsilon) \in E_X(a) \;\wedge\; x = x_\varepsilon, \\ 1, & e(\varepsilon) \notin E_X(a) \;\wedge\; x = x', \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

where $e(\varepsilon)$ denotes the effect responsible for the outcome ε.

It is not hard to verify that Equations 4-6 capture the frame axioms and probabilistic semantics of our actions. In principle, this accomplishes our construction of $\mathcal{N}_{b_{\overline{a}}}$ over the variables $\mathcal{X}_{b_{\overline{a}}} = \mathcal{Y} \bigcup_{t=0}^{m} \mathcal{X}_{(t)}$. We note, however, that the mediating variable $Y_{(t)}$ are really needed only for truly probabilistic actions. Specifically, if a^t is a deterministic action a, let $E_X(a) \subseteq E(a)$ be the conditional effects of a that add and/or delete propositions associated with the domain of a variable $X \in \mathcal{X}$. If $E_X(a) = \emptyset$, then we set $Pa(X_{(t)}) = \{X_{(t-1)}\}$, and $T_{X(t)}$ according to Equation 5. Otherwise, we set

$$Pa(X_{(t)}) \;=\; \{X_{(t-1)}\} \bigcup_{e \in E_X(a)} \left\{ X'_{(t-1)} \mid con(e) \cap Dom(X) \neq \emptyset \right\}, \tag{7}$$

and specify $T_{X_{(t)}}$ as follows. Let $x_e \in Dom(X)$ be the value that (the only deterministic outcome of) the effect $e \in E_X(a)$ provides to X. For each $\pi \in Dom(Pa(X_{(t)}))$, if there exists $e \in E_X(a)$

such that $con(e) \subseteq \pi$, then we set

$$T_{X(t)}(X_{(t)} = x \mid \pi) = \begin{cases} 1, & x = x_e, \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Otherwise, we set

$$T_{X(t)}(X_{(t)} = x \mid \pi) = \begin{cases} 1, & x = \pi[X_{(t-1)}], \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

Due to the self-consistency of the action, it is not hard to verify that Equations 8-9 are consistent, and, together with Equation 5, capture the semantics of the conditional deterministic actions. This special treatment of deterministic actions is illustrated in Figure 2 by the direct dependencies of $\mathcal{X}_{(2)}$ on $\mathcal{X}_{(1)}$.

Proposition 1 *Let $(A, \mathcal{N}_{b_I}, G, \theta)$ be a probabilistic planning problem, and $\overline{a}$ be an m-step sequence of actions applicable in b_I. Let Pr be the joint probability distribution induced by $\mathcal{N}_{b_{\overline{a}}}$ on its variables $\mathcal{X}_{b_{\overline{a}}}$. The belief state $b_{\overline{a}}$ corresponds to the marginal distribution of Pr on $\mathcal{X}_{(m)}$, that is: $b_{\overline{a}}(\mathcal{X}) = Pr(\mathcal{X}_{(m)})$, and if $G_{(m)}$ is a partial assignment provided by G to $\mathcal{X}_{(m)}$, then the probability $b_{\overline{a}}(G)$ that $\overline{a}$ achieves G starting from b_I is equal to $Pr(G_{(m)})$.*

As we already mentioned, our belief-state BNs are constructed along the principles outlined and used by Dean and Kanazawa (1989), Hanks and McDermott (1994), and Kushmerick et al. (1995), and thus the correctness of Proposition 1 is immediate from these previous results. At this point, it is worth bringing attention to the fact that all the variables in $\mathcal{X}_{(1)}, \ldots, \mathcal{X}_{(m)}$ are completely deterministic. Moreover, the CPTs of all the variables of $\mathcal{N}_{b_{\overline{a}}}$ are all *compactly representable* due to either a low number of parents, or some local structure induced by a large amount of context-specific independence, or both. This compactness of the CPTs in $\mathcal{N}_{b_{\overline{a}}}$ is implied by the compactness of the STRIPS-style specification of the planning actions. By exploiting this compactness of the action specification, the size of the $\mathcal{N}_{b_{\overline{a}}}$'s description can be kept linear in the size of the input and the number of actions in $\overline{a}$.

Proposition 2 *Let $(A, \mathcal{N}_{b_I}, G, \theta)$ be a probabilistic planning problem described over k state variables, and $\overline{a}$ be an m-step sequence of actions from A. Then, we have $|\mathcal{N}_{b_{\overline{a}}}| = O(|\mathcal{N}_{b_I}| + m\alpha(k+1))$ where α is the largest description size of an action in A.*

The proof of Proposition 2, as well as the proofs of other formal claims in the paper, are relegated to Appendix A, pp. 613.

3.2 Weighted CNFs

Given the representation of belief states as BNs, next we should select a mechanism for reasoning about these BNs. In general, computing the probability of a query in BNs is known to be #P-complete (Roth, 1996). In addition, it is not hard to verify, using an analysis similar to the ones of Darwiche (2001) and Brafman and Domshlak (2006), that the networks arising in our work will typically exhibit large tree-width. While numerous exact algorithms for inference with BNs have been proposed in the literature (Darwiche, 2000; Dechter, 1999; Zhang & Poole, 1994), the classical algorithms do not scale well on large networks exhibiting high tree-width. On the positive side, however, an observation that guides some recent advances in the area of probabilistic reasoning is that real-world domains typically exhibit a significant degree of deterministic dependencies and context-specific independencies between the problem variables. Targeting this property of practical BNs already resulted in powerful inference techniques (Chavira & Darwiche, 2005; Sang et al., 2005). The general principle underlying these techniques is to

(i) Compile a BN $\mathcal{N}$ into a *weighted propositional logic formula* $\phi(\mathcal{N})$ in CNF, and

(ii) Perform an efficient *weighted model counting* for $\phi(\mathcal{N})$ by reusing (and adapting) certain techniques that appear powerful in enhancing backtracking DPLL-style search for SAT.

One observation we had at the early stages of developing Probabilistic-FF is that the type of networks and type of queries we have in our problems make this machinery for solving BNs by weighted CNF model counting very attractive for our needs. First, in Section 3.1 we have already shown that the BNs representing our belief states exhibit a large amount of both deterministic nodes and context-specific independence. Second, the queries of our interest correspond to computing probability of the "evidence" $G_{(m)}$ in $\mathcal{N}_{b_{\overline{a}}}$, and this type of query has a clear interpretation in terms of model counting (Sang et al., 2005). Hence, taking this route in Probabilistic-FF, we compile our belief state BNs to weighted CNFs following the encoding scheme proposed by Sang et al. (2005), and answer probabilistic queries using Cachet (Sang, Bacchus, Beame, Kautz, & Pitassi, 2004), one of the most powerful systems to date for exact weighted model counting in CNFs.

In general, the weighted CNFs and the weights of such formulas are specified as follows. Let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a set of propositional variables with $Dom(V_i) = \{v_i, \neg v_i\}$, and let $\varpi : \bigcup_i Dom(V_i) \rightarrow \mathbb{R}^{0+}$ be a non-negative, real-valued *weight* function from the literals of $\mathcal{V}$. For any partial assignment π to $\mathcal{V}$, the weight $\varpi(\pi)$ of this assignment is defined as the product of its literals' weights, that is, $\varpi(\pi) = \prod_{l \in \pi} \varpi(l)$. Finally, a propositional logic formula ϕ is called *weighted* if it is defined over such a weighted set of propositional variables. For any weighted formula ϕ over $\mathcal{V}$, the weight $\varpi(\phi)$ is defined as the sum of the weights of all the complete assignments to $\mathcal{V}$ satisfying ϕ, that is,

$$\varpi(\phi) = \sum_{\pi \in Dom(\mathcal{V})} \varpi(\pi)\delta\left(\pi \models \phi\right),$$

where $Dom(\mathcal{V}) = \times_i Dom(V_i)$. For instance, if for all variables V_i we have $\varpi(v_i) = \varpi(\neg v_i) = 1$, then $\varpi(\phi)$ simply stands for the number of complete assignments to $\mathcal{V}$ that satisfy ϕ.

Given an initial belief state BN $\mathcal{N}_{b_I}$, and a sequence of actions $\overline{a} = \langle a^1, \ldots, a^m \rangle$ applicable in b_I, here we describe how the weighted CNF encoding $\phi(\mathcal{N}_{b_{\overline{a}}})$ (or $\phi(b_{\overline{a}})$, for short) of the belief state $b_{\overline{a}}$ is built and used in Probabilistic-FF. First, we formally specify the generic scheme introduced by Sang et al. (2005) for encoding a BN $\mathcal{N}$ over variables $\mathcal{X}$ into a weighted CNF $\phi(\mathcal{N})$. The encoding formula $\phi(\mathcal{N})$ contains two sets of variables. First, for each variable $Z \in \mathcal{X}$ and each value $z \in Dom(Z)$, the formula $\phi(\mathcal{N})$ contains a *state proposition* with literals $\{z, \neg z\}$, weighted as $\varpi(z) = \varpi(\neg z) = 1$. These state propositions act in $\phi(b_{\overline{a}})$ as regular SAT propositions. Now, for each variable $Z \in \mathcal{X}_{b_{\overline{a}}}$, let $Dom(Z) = \{z_1, \ldots, z_k\}$ be an arbitrary fixed ordering of $Dom(Z)$. Recall that each row $T_Z[i]$ in the CPT of Z corresponds to an assignment ζ_i (or a set of such assignments) to $Pa(Z)$. Thus, the number of rows in T_Z is upper bounded by the number of different assignments to $Pa(Z)$, but (as it happens in our case) it can be significantly lower if the dependence of Z on $Pa(Z)$ induces a substantial local structure. Following the ordering of $Dom(Z)$ as above, the entry $T_Z[i, j]$ contains the conditional probability of $Pr(z_j \mid \zeta_i)$. For every CPT entry

procedure basic-WMC(ϕ)
  if $\phi = \emptyset$ **return** 1
  if ϕ has an empty clause **return** 0
  select a variable $V \in \phi$
    return basic-WMC$(\phi|_v) \cdot \varpi(v)$ + basic-WMC$(\phi|_{\neg v}) \cdot \varpi(\neg v)$

Figure 3: Basic DPPL-style weighted model counting.

$T_Z[i,j]$ but the last one (i.e., $T_Z[i,k]$), the formula $\phi(\mathcal{N})$ contains a *chance proposition* with literals $\{\langle z_j^i \rangle, \neg\langle z_j^i \rangle\}$. These chance variables aim at capturing the probabilistic information from the CPTs of $\mathcal{N}_{b_{\overline{a}}}$. Specifically, the weight of the literal $\langle z_j^i \rangle$ is set to $Pr(z_j \mid \zeta_i, \neg z_1, \ldots, \neg z_{j-1})$, that is to conditional probability that the entry is true, given that the row is true, and no prior entry in the row is true:

$$\begin{aligned} \varpi\left(\langle z_j^i \rangle\right) &= \frac{T_Z[i,j]}{1 - \sum_{k=1}^{j-1} T_Z[i,k]} \\ \varpi\left(\neg\langle z_j^i \rangle\right) &= 1 - \varpi\left(\langle z_j^i \rangle\right) \end{aligned} \tag{10}$$

Considering the clauses of $\phi(\mathcal{N})$, for each variable $Z \in \mathcal{X}$, and each CPT entry $T_Z[i,j]$, the formula $\phi(\mathcal{N})$ contains a clause

$$\left(\zeta_i \wedge \neg\langle z_1^i \rangle \wedge \cdots \wedge \neg\langle z_{j-1}^i \rangle \wedge \langle z_j^i \rangle\right) \rightarrow z_j, \tag{11}$$

where ζ_i is a conjunction of the literals forming the assignment $\zeta_i \in Dom(Pa(Z))$. These clauses ensure that the weights of the complete assignments to the variables of $\phi(\mathcal{N})$ are equal to the probability of the corresponding atomic events as postulated by the BN $\mathcal{N}$. To illustrate the construction in Equations 10-11, let boolean variables A and B be the parents of a ternary variable C (with $Dom(C) = \{C_1, C_2, C_3\}$) in some BN, and let $Pr(C_1|A, \neg B) = 0.2$, $Pr(C_2|A, \neg B) = 0.4$, and $Pr(C_3|A, \neg B) = 0.4$. Let the raw corresponding to the assignment $A, \neg B$ to $Pa(C)$ be the i-th row of the CPT T_C. In the encoding of this BN, the first two entries of this raw of T_C are captured by a pair of respective chance propositions $\langle C_1^i \rangle$, and $\langle C_2^i \rangle$. According to Equation 10, the weights of these propositions are set to $\varpi\left(\langle C_1^i \rangle\right) = 0.2$, and $\varpi\left(\langle C_1^i \rangle\right) = \frac{0.4}{1-0.2} = 0.5$. Then, according to Equation 11, the encoding contains three clauses

$$\begin{aligned} &\left(\neg A \vee B \vee \neg\langle C_1^i \rangle \vee C_1\right) \\ &\left(\neg A \vee B \vee \langle C_1^i \rangle \vee \neg\langle C_2^i \rangle \vee C_2\right) \\ &\left(\neg A \vee B \vee \langle C_1^i \rangle \vee \langle C_2^i \rangle \vee C_3\right) \end{aligned}$$

Finally, for each variable $Z \in \mathcal{X}$, the formula $\phi(\mathcal{N})$ contains a standard set of clauses encoding the "exactly one" relationship between the state propositions capturing the value of Z. This accomplishes the encoding of $\mathcal{N}$ into $\phi(\mathcal{N})$. In the next Section 3.3 we illustrate this encoding on the belief state BN from our running example.

The weighted CNF encoding $\phi(b_{\overline{a}})$ of the belief state BN $\mathcal{N}_{b_{\overline{a}}}$ provides the input to a weighted model counting procedure. A simple recursive DPPL-style procedure basic-WMC underlying Cachet (Sang et al., 2004) is depicted in Figure 3, where the formula $\phi|_v$ is obtained from ϕ by setting the literal v to true. Theorem 3 by Sang et al. (2005) shows that if ϕ is a weighted CNF encoding of a BN $\mathcal{N}$, and $Pr(Q|E)$ is a general query with respect to $\mathcal{N}$, query Q, and evidence E, then we have:

$$Pr(Q|E) = \frac{\textsf{basic-WMC}(\phi \wedge Q \wedge E)}{\textsf{basic-WMC}(\phi \wedge E)}, \tag{12}$$

where query Q and evidence E can in fact be arbitrary formulas in propositional logic. Note that, in a special (and very relevant to us) case of empty evidence, Equation 12 reduces to $Pr(Q) = \textsf{basic-WMC}(\phi \wedge Q)$, that is, a single call to the basic-WMC procedure. Corollary 3 is then immediate from our Proposition 1 and Theorem 3 by Sang et al. (2005).

Corollary 3 *Let (A, b_I, G, θ) be a probabilistic planning task with a BN $\mathcal{N}_{b_I}$ describing b_I, and $\overline{a}$ be an m-step sequence of actions applicable in b_I. The probability $b_{\overline{a}}(G)$ that $\overline{a}$ achieves G starting from b_I is given by:*

$$b_{\overline{a}}(G) = \textsf{WMC}\left(\phi(b_{\overline{a}}) \wedge G(m)\right), \tag{13}$$

where $G(m)$ is a conjunction of the goal literals time-stamped with the time endpoint m of $\overline{a}$.

3.3 Example: Weighted CNF Encoding of Belief States

We now illustrate the generic BN-to-WCNF encoding scheme of Sang et al. (2005) on the belief state BN $\mathcal{N}_{b_{\overline{a}}}$ from our running example in Figure 2.

For $0 \leq i \leq 2$, we introduce time-stamped state propositions $r_1(i), r_2(i), b_1(i), b_2(i)$. Likewise, we introduce four state propositions $\varepsilon_1(1), \varepsilon_2(1), \varepsilon_3(1), \varepsilon'_1(1)$ corresponding to the values of the variable $Y_{(1)}$. The first set of clauses in $\phi(b_{\overline{a}})$ ensure the "exactly one" relationship between the state propositions capturing the value of a variable in $\mathcal{N}_{b_{\overline{a}}}$:

$$\begin{array}{rl} & \left(\varepsilon_1(1) \vee \varepsilon_2(1) \vee \varepsilon_3(1) \vee \varepsilon'_1(1)\right), \\ 1 \leq i < j \leq 4 \ : & \\ & \left(\neg y_i(1) \vee \neg y_j(1)\right), \\ 0 \leq i \leq 2 \ : & \\ & \left(r_1(i) \vee r_2(i)\right),\ \left(\neg r_1(i) \vee \neg r_2(i)\right) \\ & \left(b_1(i) \vee b_2(i)\right),\ \left(\neg b_1(i) \vee \neg b_2(i)\right) \end{array} \tag{14}$$

Now we proceed with encoding the CPTs of $\mathcal{N}_{b_{\overline{a}}}$. The root nodes have only one row in their CPTs so their chance propositions can be identified with the corresponding state variables (Sang et al., 2005). Hence, for the root variable $R_{(0)}$ we need neither additional clauses nor special chance propositions, but the state proposition $r_1(0)$ of $\phi(b_{\overline{a}})$ is treated as a chance proposition with $\varpi\left(r_1(0)\right) = 0.9$.

Encoding of the variable $B_{(0)}$ is a bit more involved. The CPT $T_{B_{(0)}}$ contains two (content-wise different) rows corresponding to the "given r_1" and "given r_2" cases, and both these cases induce a non-deterministic dependence of $B_{(0)}$ on $R_{(0)}$. To encode the content of $T_{B_{(0)}}$ we introduce two chance variables $\langle b_1(0)^1 \rangle$ and $\langle b_1(0)^2 \rangle$ with $\varpi(\langle b_1(0)^1 \rangle) = 0.7$ and $\varpi(\langle b_1(0)^2 \rangle) = 0.2$. The positive literals of $\langle b_1(0)^1 \rangle$ and $\langle b_1(0)^2 \rangle$ capture the events "b_1 given r_1" and "b_1 given r_2", while the negations $\neg\langle b_1(0)^1 \rangle$ and $\neg\langle b_1(0)^2 \rangle$ capture the complementary events "b_2 given r_2" and "b_2 given r_2", respectively. Now consider the "given r_1" row in $T_{B_{(0)}}$. To encode this row, we need

$\phi(b_{\overline{a}})$ to contain $\left(r_1(0) \wedge \langle b_1(0)^1\rangle\right) \rightarrow b_1(0)$ and $\left(r_1(0) \wedge \neg\langle b_1(0)^1\rangle\right) \rightarrow b_2(0)$. Similar encoding is required for the row "given r_2", and thus the encoding of T_{B^0} introduces four additional clauses:

$$\begin{aligned} &\left(\neg r_1(0) \vee \neg\langle b_1(0)^1\rangle \vee b_1(0)\right), \ \left(\neg r_1(0) \vee \langle b_1(0)^1\rangle \vee b_2(0)\right) \\ &\left(\neg r_2(0) \vee \neg\langle b_1(0)^2\rangle \vee b_1(0)\right), \ \left(\neg r_2(0) \vee \langle b_1(0)^2\rangle \vee b_2(0)\right) \end{aligned} \tag{15}$$

Having finished with the $\mathcal{N}_{b_I}$ part of $\mathcal{N}_{b_{\overline{a}}}$, we proceed with encoding the variable $Y_{(1)}$ corresponding to the probabilistic action $move\text{-}b\text{-}right$. To encode the first row of $T_{Y_{(1)}}$ we introduce three chance propositions $\langle\varepsilon_1(1)^1\rangle$, $\langle\varepsilon_2(1)^1\rangle$, and $\langle\varepsilon_3(1)^1\rangle$; in general, no chance variables are needed for the last entries of the CPT rows. The weights of these chance propositions are set according to Equation 10 to $\varpi\left(\langle\varepsilon_1(1)^1\rangle\right) = 0.7$, $\varpi\left(\langle\varepsilon_2(1)^1\rangle\right) = \frac{0.2}{1-0.7} = 0.6(6)$, and $\varpi\left(\langle\varepsilon_3(1)^1\rangle\right) = \frac{0.1}{1-0.9} = 0.1$. Using these chance propositions, we add to $\phi(b_{\overline{a}})$ four clauses as in Equation 11, notably the first four clauses of Equation 16 below.

Proceeding the second row of $T_{Y_{(1)}}$, observe that the value of $R_{(0)}$ and $B_{(0)}$ in this case fully determines the value of $Y_{(1)}$. This deterministic dependence can be encoded without using any chance propositions using the last two clauses in Equation 16.

$$\begin{aligned} &\left(\neg r_1(0) \vee \neg b_1(0) \vee \neg\langle\varepsilon_1(1)^1\rangle \vee \varepsilon_1(1)\right), \\ &\left(\neg r_1(0) \vee \neg b_1(0) \vee \langle\varepsilon_1(1)^1\rangle \vee \neg\langle\varepsilon_2(1)^1\rangle \vee \varepsilon_2(1)\right), \\ &\left(\neg r_1(0) \vee \neg b_1(0) \vee \langle\varepsilon_1(1)^1\rangle \vee \langle\varepsilon_2(1)^1\rangle \vee \neg\langle\varepsilon_3(1)^1\rangle \vee \varepsilon_3(1)\right), \\ &\left(\neg r_1(0) \vee \neg b_1(0) \vee \langle\varepsilon_1(1)^1\rangle \vee \langle\varepsilon_2(1)^1\rangle \vee \langle\varepsilon_3(1)^1\rangle \vee \varepsilon'_1(1)\right), \\ \\ &\left(r_1(0) \vee \neg\varepsilon'_1(1)\right), \ \left(b_1(0) \vee \neg\varepsilon'_1(1)\right) \end{aligned} \tag{16}$$

Using the state/chance variables introduced for R^0, B^0, and $Y_{(1)}$, we encode the CPTs of $R_{(1)}$ and $B_{(1)}$ as:

$$\begin{aligned} R_{(1)} : &\left(\neg\varepsilon_1(1) \vee r_2(1)\right), \ \left(\neg\varepsilon_2(1) \vee r_2(1)\right), \\ &\left(\neg\varepsilon_3(1) \vee \neg r_1(0) \vee r_1(1)\right), \ \left(\neg\varepsilon'_1(1) \vee \neg r_1(0) \vee r_1(1)\right), \\ &\left(\neg\varepsilon_3(1) \vee \neg r_1(0) \vee r_1(1)\right), \ \left(\neg\varepsilon'_1(1) \vee \neg r_2(0) \vee r_2(1)\right) \\ B_{(1)} : &\left(\neg\varepsilon_1(1) \vee b_2(1)\right), \\ &\left(\varepsilon_1(1) \vee \neg b_1(0) \vee b_1(1)\right), \\ &\left(\varepsilon_1(1) \vee \neg b_2(0) \vee b_2(1)\right) \end{aligned} \tag{17}$$

Since the CPTs of both $R_{(1)}$ and $B_{(1)}$ are completely deterministic, their encoding as well is using no chance propositions. Finally, we encode the (deterministic) CPTs of $R_{(2)}$ and $B_{(2)}$ as:

$$\begin{aligned} R_{(2)} : &\left(r_1(2)\right) \\ B_{(2)} : &\left(\neg b_1(1) \vee b_1(2)\right) \\ &\left(\neg b_2(1) \vee b_2(2)\right) \end{aligned} \tag{18}$$

where the unary clause $(r_1(2))$ is a reduction of $(\neg r_1(1) \vee r_1(2))$ and $(\neg r_2(1) \vee r_1(2))$. This accomplishes our encoding of $\phi(b_{\overline{a}})$.

3.4 From Conformant-FF to Probabilistic-FF

Besides the fact that weighted model counting is attractive for the kinds of BNs arising in our context, the weighted CNF representation of belief states works extremely well with the ideas underlying Conformant-FF (Hoffmann & Brafman, 2006). This was outlined in the introduction already; here we give a few more details.

As stated, Conformant-FF does a forward search in a non-probabilistic belief space in which each belief state corresponds to a set of world states considered to be possible. The main trick of Conformant-FF is the use of CNF formulas for an implicit representation of belief states, where formulas $\phi(\overline{a})$ encode the semantics of executing action sequence $\overline{a}$ in the initial belief state. Facts known to be true or false are inferred from these formulas. This computation of only a partial knowledge constitutes a *lazy* kind of belief state representation, in comparison to other approaches that use explicit enumeration (Bonet & Geffner, 2000) or BDDs (Bertoli, Cimatti, Pistore, Roveri, & Traverso, 2001) to fully represent belief states. The basic ideas underlying Probabilistic-FF are:

(i) Define time-stamped Bayesian Networks (BN) describing probabilistic belief states (Section 3.1 above).

(ii) Extend Conformant-FF's belief state CNFs to model these BN (Section 3.2 above).

(iii) In addition to the SAT reasoning used by Conformant-FF, use weighted model-counting to determine whether the probability of the (unknown) goals in a belief state is high enough (directly below).

(iv) Introduce approximate probabilistic reasoning into Conformant-FF's heuristic function (Section 4 below).

In more detail, given a probabilistic planning task (A, b_I, G, θ), a belief state $b_{\overline{a}}$ corresponding to some applicable in b_I m-step action sequence $\overline{a}$, and a proposition $q \in \mathcal{P}$, we say that q is *known* in $b_{\overline{a}}$ if $b_{\overline{a}}(q) = 1$, *negatively known* in $b_{\overline{a}}$ if $b_{\overline{a}}(q) = 0$, and *unknown* in $b_{\overline{a}}$, otherwise. We begin with determining whether each q is known, negatively known, or unknown at time m. Re-using the Conformant-FF machinery, this classification requires up to two SAT tests of $\phi(b_{\overline{a}}) \wedge \neg q(m)$ and $\phi(b_{\overline{a}}) \wedge q(m)$, respectively. The information provided by this classification is used threefold. First, if a subgoal $g \in G$ is negatively known at time m, then we have $b_{\overline{a}}(G) = 0$. On the other extreme, if all the subgoals of G are known at time m, then we have $b_{\overline{a}}(G) = 1$. Finally, if some subgoals of G are known and the rest are unknown at time m, then we accomplish evaluating the belief state $b_{\overline{a}}$ by testing whether

$$b_{\overline{a}}(G) = \mathsf{WMC}\left(\phi(b_{\overline{a}}) \wedge G(m)\right) \geq \theta. \tag{19}$$

Note also that having the sets of all (positively/negatively) known propositions at all time steps up to m allows us to *significantly* simplify the CNF formula $\phi(b_{\overline{a}}) \wedge G(m)$ by inserting into it the corresponding values of known propositions.

After evaluating the considered action sequence $\overline{a}$, if we get $b_{\overline{a}}(G) \geq \theta$, then we are done. Otherwise, the forward search continues, and the actions that are applicable in $b_{\overline{a}}$ (and thus used to generate the successor belief states) are actions whose preconditions are all known in $b_{\overline{a}}$.

4. Heuristic Function

The key component of any heuristic search procedure is the heuristic function. The quality (informedness) and computational cost of that function determine the performance of the search. The heuristic function is usually obtained from solutions to a relaxation of the actual problem of interest (Pearl, 1984; Russell & Norvig, 2004). In classical planning, a successful idea has been to use a relaxation that ignores the delete effects of the actions (McDermott, 1999; Bonet & Geffner, 2001; Hoffmann & Nebel, 2001). In particular, the heuristic of the FF planning system is based on the notion of *relaxed plan*, which is a plan that achieves the goals while assuming that all delete lists of actions are empty. The relaxed plan is computed using a Graphplan-style (Blum & Furst, 1997) technique combining a forward chaining graph construction phase with a backward chaining plan extraction phase. The heuristic value $h(w)$ that FF provides to a world state w encountered during the search is the length of the relaxed plan from w. In Conformant-FF, this methodology was extended to the setting of conformant planning under initial state uncertainty (without uncertainty about action effects). Herein, we extend Conformant-FF's machinery to handle probabilistic initial states and effects. Section 4.1 provides background on the techniques used in FF and Conformant-FF, then Sections 4.2 and 4.4 detail our algorithms for the forward and backward chaining phases in Probabilistic-FF, respectively. These algorithms for the two phases of the Probabilistic-FF heuristic computation are illustrated on our running example in Sections 4.3 and 4.5, respectively.

4.1 FF and Conformant-FF

We specify how relaxed plans are computed in FF; we provide a coarse sketch of how they are computed in Conformant-FF. The purpose of the latter is only to slowly prepare the reader for what is to come: Conformant-FF's techniques are re-used for Probabilistic-FF anyway, and hence will be described in full detail as part of Sections 4.2 and 4.4.

Formally, relaxed plans in classical planning are computed as follows. Starting from w, FF builds 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 obtained from $P(t)$ by including the add effects (with fulfilled conditions) of the actions in $A(t)$. That is, $P(t)$ always contains those facts that will be true if one would execute (the relaxed versions of) all actions at the earlier layers up to $A(t-1)$. The relaxed planning graph is constructed either until it reaches a propositional layer $P(m)$ that contains all the goals, or until the construction reaches a fixpoint $P(t) = P(t+1)$ without reaching the goals. The latter case corresponds to (all) situations in which a relaxed plan does not exist, and because existence of a relaxed plan is a necessary condition for the existence of a real plan, the state w is excluded from the search space by setting $h(w) = \infty$. In the former case of $G \subseteq P(m)$, a relaxed plan is a subset of actions in $A(1), \dots, A(m)$ that suffices to achieve the goals (under ignoring the delete lists), and it can be extracted by a simple backchaining loop: For each goal in $P(m)$, select an action in $A(1), \dots, A(m)$ that achieves this goal, and iterate the process by considering those actions' preconditions and the respective effect conditions as new subgoals. The heuristic estimate $h(w)$ is then set to the length of the extracted relaxed plan, that is, to the number of actions selected in this backchaining process.

Aiming at extending the machinery of FF to conformant planning, in Conformant-FF, Hoffmann and Brafman (2006) suggested to extend the relaxed planning graph with additional fact layers $uP(t)$ containing the facts *unknown* at time t, and then to reason about when such unknown facts become known in the relaxed planning graph. As the complexity of this type of reasoning is prohibitive, Conformant-FF further relaxes the planning task by ignoring not only the delete lists, but also all but one of the unknown conditions of each action effect. That is, if action a appears in layer $A(t)$, and for effect e of a we have $con(e) \subseteq P(t) \cup uP(t)$ and $con(e) \cap uP(t) \neq \emptyset$, then $con(e) \cap uP(t)$ is arbitrarily reduced to contain exactly one literal, and reasoning is done as if $con(e)$ had this reduced form from the beginning.

This relaxation converts implications $(\bigwedge_{c \in con(e) \cap uP(t)} c(t)) \rightarrow q(t+1)$ that the action effects induce between unknown propositions into their 2-projections that take the form of *binary implications* $c(t) \rightarrow q(t+1)$, for arbitrary $c \in con(e) \cap uP(t)$. Due to the layered structure of the planning graph, the set of all these binary implications $c(t) \rightarrow q(t+1)$ can be seen as forming a directed acyclic graph *Imp*. Under the given relaxations, this graph captures exactly all dependencies between the truth of propositions over time. Hence, checking whether a proposition q becomes known at time t can be done as follows. First, backchain over the implication edges of *Imp* that end in $q(t)$, and collect the set *support*$(q(t))$ of leafs[4] that are reached. Then, if Φ is the CNF formula describing the possible initial states, test by a SAT check whether

$$\Phi \rightarrow \bigvee_{l \in \mathit{support}(q(t))} l$$

This test will succeed if and only if at least one of the leafs in *support*$(q(t))$ is true in every possible initial state. Under the given relaxations, this is the case if and only if, when applying all actions in the relaxed planning graph, q will always be true at time t.[5]

The process of extracting a relaxed plan from the constructed conformant relaxed planning graph is an extension of FF's respective process with machinery that selects actions responsible for relevant paths in *Imp*. The overall Conformant-FF heuristic machinery is sound and complete for relaxed tasks, and yields a heuristic function that is highly informative across a range of challenging domains (Hoffmann & Brafman, 2006).

In this work, we adopt Conformant-FF's relaxations, ignoring the delete lists of the action effects, as well as all but one of the propositions in the effect's condition. Accordingly, we adopt the following notations from Conformant-FF. Given a set of actions A, we denote by $|_1^+$ any function from A into the set of all possible actions, such that $|_1^+$ maps each $a \in A$ to the action similar to a but with empty delete lists and with all but one conditioning propositions of each effect removed; for $|_1^+(a)$, we write $a|_1^+$. By $A|_1^+$ we denote the action set obtained by applying $|_1^+$ to all the actions of A, that is, $A|_1^+ = \left\{a|_1^+ \mid a \in A\right\}$. For an action sequence $\overline{a}$ we denote by $\overline{a}|_1^+$ the sequence of actions obtained by applying $|_1^+$ to every action along $\overline{a}$, that is,

$$\overline{a}|_1^+ = \begin{cases} \langle\rangle, & \overline{a} = \langle\rangle \\ \langle a|_1^+ \rangle \cdot \overline{a}'|_1^+, & \overline{a} = \langle a \rangle \cdot \overline{a}' \end{cases}.$$

For a probabilistic planning task (A, b_I, G, θ), the task $(A|_1^+, b_I, G, \theta)$ is called a relaxation of (A, b_I, G, θ). Finally, if $\overline{a}|_1^+$ is a plan for $(A|_1^+, b_I, G, \theta)$, then $\overline{a}$ is called a relaxed plan for (A, b_I, G, θ).

4. Following the Conformant-FF terminology, by "leafs" we refer to the nodes having zero in-degree.
5. Note here that it would be possible to do a full SAT check, without any 2-projection (without relying on *Imp*), to see whether q becomes known at t. However, as indicated above, doing such a full check for every unknown proposition at every level of the relaxed planning graph for every search state would very likely be too expensive, computationally.

In the next two sections we describe the machinery underlying the Probabilistic-FF heuristic estimation. Due to the similarity between the conceptual relaxations used in Probabilistic-FF and Conformant-FF, Probabilistic-FF inherits almost all of Conformant-FF's machinery. Of course, the new contributions are those algorithms dealing with probabilistic belief states and probabilistic actions.

4.2 Probabilistic Relaxed Planning Graphs

Like FF and Conformant-FF, Probabilistic-FF computes its heuristic function in two steps, the first one chaining forward to build a relaxed planning graph, and the second step chaining backward to extract a relaxed plan. In this section, we describe in detail Probabilistic-FF's forward chaining step, building a *probabilistic relaxed planning graph* (or PRPG, for short). In Section 4.4, we then show how one can extract a (probabilistic) relaxed plan from the PRPG. We provide a detailed illustration of the PRPG construction process on the basis of our running example; since the illustration is lengthy, it is moved to a separate Section 4.3.

The algorithms building a PRPG are quite involved; it is instructive to first consider (some of) the key points before delving into the details. The main issue is, of course, that we need to extend Conformant-FF's machinery with the ability to determine when the goal set is sufficiently *likely*, rather than when it is known to be true for sure. To achieve that, we must introduce into relaxed planning some effective reasoning about both the probabilistic initial state, and the effects of probabilistic actions. It turns out that such a reasoning can be obtained by a certain *weighted* extension of the implication graph. In a nutshell, if we want to determine how likely it is that a fact q is true at a time t, then we propagate certain weights backwards through the implication graph, starting in $q(t)$; the weight of $q(t)$ is set to 1, and the weight for any $p(t')$ gives an estimate of *the probability of achieving q at t given that p holds at t'*. Computing this probability exactly would, of course, be too expensive. Our estimation is based on *assuming independence of the various probabilistic events involved.* This is a choice that we made very carefully; we experimented widely with various other options before deciding in favor of this technique.

Any simplifying assumption in the weight propagation constitutes, of course, another relaxation, on top of the relaxations we already inherited from Conformant-FF. The particularly problematic aspect of assuming independence is that it is not an under-estimating technique. The actual weight of a node $p(t')$ – the probability of achieving q at t given that p holds at t' – may be lower than our estimate. In effect, the PRPG may decide wrongly that a relaxed plan exists: even if we execute all relaxed actions contained in the successful PRPG, the probability of achieving the goal by this execution may be less than the required threshold. In other words, we lose the soundness (relative to relaxed tasks) of the relaxed planning process.

We experimented with an alternative weight propagation method, based on an opposite assumption, that the relevant probabilistic events always co-occur, and that hence the weights must be propagated according to simple maximization operations. This propagation method yielded very uninformative heuristic values, and hence inacceptable empirical behaviour of Probabilistic-FF, even in very simple benchmarks. In our view, it seems unlikely that an under-estimating yet informative and efficient weight computation exists. We further experimented with some alternative non under-estimating propagation schemes, in particular one based on assuming that the probabilistic events are completely disjoint (and hence weights should be *added*); these schemes gave better performance than maximization, but lagged far behind the independence assumption in the more challenging benchmarks.

Let us now get into the actual algorithm building a PRPG. A coarse outline of the algorithm is as follows. The PRPG is built in a layer-wise fashion, in each iteration extending the PRPG, reaching up to time t, by another layer, reaching up to time $t+1$. The actions in the new step are those whose preconditions are known to hold at t. Effects conditioned on unknown facts (note here the reduction of effect conditions to a single fact) constitute new edges in the implication graph. In difference to Conformant-FF, we don't obtain a single edge from condition to add effect; instead, we obtain edges from the condition to "chance nodes", where each chance node represents a probabilistic outcome of the effect; the chance nodes, in turn, are linked by edges to their respective add effects. The weights of the chance nodes are set to the probabilities of the respective outcomes, the weights of all other nodes are set to 1. These weights are "static weights" which are not "dynamically" modified by weight propagation; rather, the static weights form an input to the propagation.

Once all implication graph edges are inserted at a layer, the algorithm checks whether any new facts become known. This check is done very much like the corresponding check in Conformant-FF, by testing whether the disjunction of the support leafs for a proposition p at $t+1$ is implied by the initial state formula. The two differences to Conformant-FF are: (1) Only leafs are relevant whose dynamic weight is 1 (otherwise, achieving a leaf is not guaranteed to accomplish p at $t+1$). (2) Another reason for p to become known may be that all outcomes of an unconditional effect (or an effect with known condition) result in achievement of p at time $t+1$. We elegantly formulate the overall test by a single implication test over support leafs whose dynamic weight equals their own weight.

Like FF's and Conformant-FF's algorithms, the PRPG process has two termination criteria. The PRPG terminates positively if the goal probability is high enough at time t; the PRPG terminates negatively if, from t to $t+1$, nothing has changed that may result in a higher goal propability at some future t'. The goal probability in a layer t is computed based on weighted model counting over a formula derived from the support leafs of all goals not known to be true. The criteria for negative termination check: whether any new facts have become known or unknown (not negatively known); whether any possibly relevant new support leafs have appeared; and whether the goal probability has increased. If neither is the case, then we can stop safely—if the PRPG terminates unsuccessfully then we have a guarantee that there is no relaxed plan, and that the corresponding belief is hence a dead end.

Let us get into the details. Figure 4 depicts the main routine for building the PRPG for a belief state $b_{\overline{a}}$. As we already specified, the sets $P(t)$, $uP(t)$, and $A(t)$ contain the propositions that are known to hold at time t (hold at t with probability 1), the propositions that are unknown to hold at time t (hold at t with probability less than 1 but greater than 0), and actions that are known to be applicable at time t, respectively. The layers $t \geq 0$ of PRPG capture applying the relaxed actions starting from $b_{\overline{a}}$. The layers $-m$ to -1 of PRPG correspond to the m-step action sequence $\overline{a}$ leading from the initial belief state to the belief state in question $b_{\overline{a}}$. We inherit the latter technique from Conformant-FF; in a sense, the PRPG "reasons about the past". This may look confusing at first sight, but it has a simple reason. Imagine the PRPG starts at level 0 instead. Then, to check whether a proposition becomes known, we have to do SAT tests regarding support leafs *against the belief state formula, $\phi(b_{\overline{a}})$, instead of the initial state formula* (similarly for weighted model counting to test whether the goal is likely enough). Testing against $\phi(b_{\overline{a}})$ is possible, but very expensive

procedure build-PRPG($\overline{a}, A, \phi(\mathcal{N}_{b_I}), G, \theta, |_1^+$),
returns a Bool saying if there is a relaxed plan for the belief state
given by $\overline{a} = \langle a^{-m}, \dots, a^{-1} \rangle$, and
builds data structures from which a relaxed plan can be extracted
$\Phi := \phi(\mathcal{N}_{b_I})$, *Imp* $:= \emptyset$
$P(-m) := \{p \mid p$ is known in $\Phi\}$, $uP(-m) := \{p \mid p$ is unknown in $\Phi\}$
for $t := -m \cdots -1$ **do**
$A(t) := \{a^t|_1^+\} \cup NOOPS$
build-timestep($t, A(t)$)
endfor
$t := 0$
while get-P(t, G) $< \theta$ **do**
$A(t) := \{a|_1^+ \mid a \in A, pre(a) \subseteq P(t)\} \cup NOOPS$
build-timestep($t, A(t)$)
if $P(t+1) = P(t)$ and
$uP(t+1) = uP(t)$ and
$\forall p \in uP(t+1) : uP(-m) \cap$ *support*$(p(t+1)) = uP(-m) \cap$ *support*$(p(t))$ and
get-P($t+1, G$) $=$ get-P(t, G) **then**
return FALSE
endif
$t := t + 1$
endwhile
$T := t$, **return** TRUE

Figure 4: Main routine for building a probabilistic relaxed planning graph (PRPG).

computationally.[6] The negative-index layers chain the implication graph all the way back to the initial state, and hence enable us to perform SAT tests against the – typically much smaller – initial state formula.

Returning to Figure 4, the PRPG is initialized with an empty implication set *Imp*, $P(-m)$ and $uP(-m)$ are assigned the propositions that are known and unknown in the initial belief state, and a weighted CNF formula Φ is initialized with $\phi(\mathcal{N}_{b_I})$. Φ is the formula against which implication/weighted model checking tests are run when asking whether a proposition becomes known/whether the goal is likely enough. While the PRPG is built, Φ is incrementally extended with further clauses to capture the behavior of different effect outcomes.

The **for** loop builds the sets P and uP for the $\overline{a}$'s time steps $-m \cdots -1$ by iterative invocation of the build-timestep procedure that each time expands PRPG by a single time level. At each iteration $-m \leq t \leq -1$, the sets $P(t+1)$ and $uP(t+1)$ are made to contain the propositions that are known/unknown after applying the relaxed version of the action $a^t \in \overline{a}$ (remember that $\overline{a} = \langle a^1, \dots, a^m \rangle$). To simplify the presentation, each action set $A(t)$ contains a set of dummy actions $NOOPS$ that simply transport all the propositions from time layer t to time layer $t{+}1$. More formally, $NOOPS = \{\text{noop}_p \mid p \in \mathcal{P}\}$, where $pre(\text{noop}_p) = \emptyset$, $E(\text{noop}_p) = \{(\{p\}, \{\varepsilon\})\}$, and $\varepsilon = (1.0, \{p\}, \emptyset)\})$.

6. In Conformant-FF, this configuration is implemented as an option; it significantly slows down the search in most domains, and brings advantages only in a few cases.

The subsequent **while** loop constructs the relaxed planning graph from layer 0 onwards by, again, iterative invocation of the build-timestep procedure. The actions in each layer $t \geq 0$ are relaxations of those actions whose preconditions are known to hold at time t with certainty. This iterative construction is controlled by two termination tests. First, if the goal is estimated to hold at layer t with probability higher than θ, then we know that a relaxed plan estimate can be extracted. Otherwise, if the graph reaches a fix point, then we know that no relaxed (and thus, no real) plan from b_I exists. We postpone the discussion of these two termination criteria, and now focus on the time layer construction procedure build-timestep.

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
$P(t+1) := P(t), uP(t+1) := \emptyset$
for all effects e of an action $a \in A$, $con(e) \in P(t) \cup uP(t)$ **do**
  for all $\varepsilon \in \Lambda(e)$ **do**
    $uP(t+1) := uP(t+1) \cup add(\varepsilon)$
    introduce new fact $\varepsilon(t)$ with $\varpi(\varepsilon(t)) = Pr(\varepsilon)$
    $Imp := Imp \cup \{(\varepsilon(t), p(t+1)) \mid p \in add(\varepsilon)\}$
  endfor
  if $con(e) \in uP(t)$ **then**
    $Imp := Imp \cup \bigcup_{\varepsilon \in \Lambda(e)} \{(con(e)(t), \varepsilon(t))\}$
  else
    $\Phi := \Phi \wedge \left(\vee_{\varepsilon \in \Lambda(e)} \varepsilon(t)\right) \wedge \bigwedge_{\varepsilon, \varepsilon' \in \Lambda(e)} (\neg\varepsilon(t) \vee \neg\varepsilon'(t))$
  endif
endfor
for all $p \in uP(t+1)$ **do**
  build-w-impleafs$(p(t+1), Imp)$
  $support(p(t+1)) := \{l \mid l \in leafs(Imp_{\rightarrow p(t+1)}) \wedge \varpi_{p(t+1)}(l) = \varpi(l)\}$
  if $\Phi \rightarrow \bigvee_{l \in support(p(t+1))} l$ **then** $P(t+1) := P(t+1) \cup \{p\}$ **endif**
endfor
$uP(t+1) := uP(t+1) \setminus P(t+1)$

Figure 5: Building a time step of the PRPG.

The build-timestep procedure is shown in Figure 5. The first **for** loop of build-timestep proceeds over all outcomes of (relaxed) actions in the given set A that may occur at time t. For each such probabilistic outcome we introduce a new chance proposition weighted by the conditional likelihood of that outcome.[7] Having that, we extend *Imp* with binary implications from this new chance proposition to the add list of the outcome. If we are uncertain about the condition $con(e)$ of the corresponding effect at time t, that is, we have $con(e) \in uP(t)$, then we also add implications from $con(e)$ to the chance propositions created for the outcomes of e. Otherwise, if $con(e)$ is known at time t, then there is no uncertainty about our ability to make the effect e to hold at time t. In this case, we do not "ground" the chance propositions created for the outcomes of e into the implication graph, but simply extend the running formula Φ with clauses capturing the "exactly one" relationship between these chance propositions corresponding to the alternative outcomes of e

7. Of course, in our implementation we have a special case treatment for deterministic actions, using no chance nodes (rather than a single "chance node" with static weight 1).

at time t. This way, the probabilistic uncertainty about the outcome of e can be treated as if being a property of the initial belief state b_I; This is the only type of knowledge we add into the knowledge base formula Φ after initializing it in build-PRPG to $\phi(\mathcal{N}_{b_I})$.

Notation	*Description*
$Imp_{v \to u}$	The graph containing exactly all the paths from node v to node u in *Imp*.
$Imp_{\to u}$	The subgraph of *Imp* formed by node u and all the ancestors of u in *Imp*.
$leafs(Imp')$	The set of all zero in-degree nodes in the subgraph Imp' of *Imp*.
$E(Imp')$	The set of time-stamped action effects responsible for the implication edges of the subgraph Imp' of *Imp*.

Table 2: Overview of notations around the implication graph.

The second **for** loop checks whether a proposition p, unknown at time t, becomes known at time $t+1$. This part of the build-timestep procedure is somewhat more involved; Table 2 provides an overview of the main notations used in the follows when discussing the various uses of the implication graph *Imp*.

First thing in the second **for** loop of build-timestep, a call to build-w-impleafs procedure associates each node $v(t')$ in $Imp_{\to p(t+1)}$ with an estimate $\varpi_{p(t+1)}(v(t'))$ on the probability of achieving p at time $t+1$ by the effects $E(Imp_{v(t') \to p(t+1)})$, given that v holds at time t'. In other words, the dynamic weight (according to $p(t+1)$) of the implication graph nodes is computed. Note that $v(t')$ can be either a time-stamped proposition $q(t')$ for some $q \in \mathcal{P}$, or a chance proposition $\varepsilon(t')$ for some probabilistic outcome ε.

We will discuss the build-w-impleafs procedure in detail below. For proceeding to understand the second **for** loop of build-timestep, the main thing we need to know is the following lemma:

Lemma 4 *Given a node $v(t') \in Imp_{\to p(t+1)}$, we have $\varpi_{p(t+1)}(v(t')) = \varpi(v(t'))$ if and only if, given v at time t', the sequence of effects $E(Imp_{v(t') \to p(t+1)})$ achieves p at $t+1$ with probability* 1.

In words, $v(t')$ leads to $p(t+1)$ with certainty iff the dynamic weight of $v(t')$ equals its static weight. This is a simple consequence of how the weight propagation is arranged; it should hold true for any reasonable weight propagation scheme ("do not mark a node as certain if it is not"). A full proof of the lemma appears in Appendix A on pp. 613.

Re-consider the second **for** loop of build-timestep. What happens is the following. Having finished the build-w-impleafs weight propagation for p at time $t+1$, we

1. collect all the leafs $support(p(t+1))$ of $Imp_{\to p(t)}$ that meet the criteria of Lemma 4, and
2. check (by a call to a SAT solver) whether the knowledge-base formula Φ implies the disjunction of these leafs.

If the implication holds, then the examined fact p at time t is added to the set of facts known at time t. Finally, the procedure removes from the set of facts that are known to possibly hold at time $t+1$ all those facts that were just proven to hold at time $t+1$ with certainty.

To understand the above, consider the following. With Lemma 4, $support(p(t+1))$ contains exactly the set of leafs achieving which will lead to $p(t+1)$ with certainty. Hence we can basically

procedure build-w-impleafs $(p(t), \mathit{Imp})$
top-down propagation of weights $\varpi_{p(t)}$ from $p(t)$ to all nodes in $\mathit{Imp}_{\rightarrow p(t)}$
$\varpi_{p(t)}\,(p(t)) := 1$
for decreasing time steps $t' := (t-1)\ldots(-m)$ **do**
  for all chance nodes $\varepsilon(t') \in \mathit{Imp}_{\rightarrow p(t)}$ **do**
    $\alpha := \prod_{r \in add(\varepsilon), r(t'+1) \in \mathit{Imp}_{\rightarrow p(t)}} \left[1 - \varpi_{p(t)}\,(r(t'+1))\right]$
    $\varpi_{p(t)}\,(\varepsilon(t')) := \varpi\,(\varepsilon(t')) \cdot (1-\alpha)$
  endfor
  for all fact nodes $q(t') \in \mathit{Imp}_{\rightarrow p(t)}$ **do**
    $\alpha := 1$
    for all $a \in A(t'), e \in E(a), con(e) = q$ **do**
      $\alpha := \alpha \cdot \left[1 - \sum_{\varepsilon \in \Lambda(e), \varepsilon(t') \in \mathit{Imp}_{\rightarrow p(t)}} \varpi_{p(t)}\,(\varepsilon(t'))\right]$
    endfor
    $\varpi_{p(t)}\,(q(t')) := 1 - \alpha$
  endfor
endfor

Figure 6: The *build-w-impleafs* procedure for weight back-propagation over the implication graph.

use the same implication test as in Conformant-FF. Note, however, that the word "basically" in the previous sentence hides a subtle but important detail. In difference to the situation in Conformant-FF, *support*$(p(t+1))$ may contain two kinds of nodes: (1) proposition nodes at the start layer of the PRPG, i.e., at layer $-m$ corresponding to the initial belief; (2) chance nodes at later layers of the PRPG, corresponding to outcomes of effects that have no unknown conditions. This is the point where the discussed above updates on the formula Φ are needed—those keep track of alternative effect outcomes. Hence testing $\Phi \rightarrow \bigvee_{l \in support(p(t+1))} l$ is the same as testing whether either: (1) p is known at $t+1$ because it is always triggered with certainty by at least one proposition true in the initial world; or (2) p is known at $t+1$ because it is triggered by all outcomes of an effect that will appear with certainty. We get the following result:

Lemma 5 *Let $(A, \mathcal{N}_{b_I}, G, \theta)$ be a probabilistic planning task, $\overline{a}$ be a sequence of actions applicable in b_I, and $|_1^+$ be a relaxation function for A. For each time step $t \geq -m$, and each proposition $p \in \mathcal{P}$, if $P(t)$ is constructed by* build-PRPG$(\overline{a}, A, \phi(\mathcal{N}_{b_I}), G, \theta, |_1^+)$*, then p at time t can be achieved by a relaxed plan starting with $\overline{a}|_1^+$*

(1) with probability > 0 (that is, p is not negatively known at time t) if and only if $p \in uP(t) \cup P(t)$, and

(2) with probability 1 (that is, p is known at time t) if and only if $p \in P(t)$.

This is a consequence of the arguments outlined above. The full proof of Lemma 5 is given in Appendix A on pp. 614.

Let us now consider the weight-propagating[8] procedure build-w-impleafs depicted in Figure 6. This procedure performs a layered, top-down weight propagation from a given node[9] $p(t) \in \mathit{Imp}$

8. The weight propagation scheme of the build-w-impleafs procedure is similar in nature to this used in the heuristics module of the recent probabilistic temporal planner Prottle of Little, Aberdeen, and Thiébaux (2005).

9. Note that the "t" here will be instantiated with $t+1$ when called from build-timestep.

down to the leafs of $Imp_{\to p(t)}$. This order of traversal ensures that each node of $Imp_{\to p(t)}$ is processed only after all its successors in $Imp_{\to p(t)}$. For the chance nodes $\varepsilon(t')$, the dynamic weight $\varpi_{p(t)}(\varepsilon(t'))$ is set to

1. the probability that the outcome ε takes place at time t' given that the corresponding action effect $e(\varepsilon)$ does take place at t', times
2. an estimate of the probability of achieving p at time t by the effects $E(Imp_{\varepsilon(t')\to p(t)})$.

The first quantity is given by the "global", static weight $\varpi(\varepsilon(t'))$ assigned to $\varepsilon(t')$ in the first **for** loop of build-timestep. The second quantity is derived from the dynamic weights $\varpi_{p(t)}(r(t'+1))$ for $r \in add(\varepsilon)$, computed in the previous iteration of the outermost **for** loop of build-w-impleafs. Making a heuristic assumption that the effect sets $E(Imp_{r(t'+1)\to p(t)})$ for different $r \in add(\varepsilon)$ are all pairwise independent, α is then set to the probability of failure to achieve p at t by the effects $E(Imp_{\varepsilon(t')\to p(t)})$. This computation of α for $\varepsilon(t')$ is decomposed over the artifacts of ε, and this is where the weight propagation starts taking place. For the fact nodes $q(t')$, the dynamic weight $\varpi_{p(t)}(q(t'))$ is set to the probability that some action effect conditioned on q at time t' allows (possibly indirectly) achieving the desired fact p at time t. Making again the heuristic assumption of independence between various such effects conditioned on q at t', computing $\varpi_{p(t)}(q(t'))$ is decomposed over the outcomes of these effects.

procedure get-P (t, G)
estimates the probability of achieving G at time p.
if $G \not\subseteq P(t) \cup uP(t)$ **then return** 0 **endif**
if $G \subseteq P(t)$ **then return** 1 **endif**
for $g \in G \setminus P(t)$ **do**
for each $l \in leafs(Imp_{\to g(t)})$, introduce a chance proposition $\langle l_g \rangle$ with weight $\varpi_{g(t)}(l)$
$\varphi_g := (\bigvee_{l \in leafs(Imp_{\to g(t)})} l) \wedge \bigwedge_{l \in leafs(Imp_{\to g(t)}) \cap uP(-m)} (\neg l \vee \langle l_g \rangle)$
endfor
return WMC$(\Phi \wedge \bigwedge_{g \in G \setminus P(t)} \varphi_g)$

Figure 7: Estimating the goal likelihood at a given time step.

What remains to be explained of the build-PRPG procedure are the two termination criteria of the **while** loop constructing the planning graph from the layer 0 onwards. The first test is made by a call to the get-P procedure, and it checks whether the PRPG built to the time layer T contains a relaxed plan for $(A, \mathcal{N}_{b_I}, G, \theta)$. The get-P procedure is shown in Figure 7. First, if one of the subgoals is negatively known at time t, then, from Lemma 5, the overall probability of achieving the goal is 0. On the other extreme, if all the subgoals are known at time t, then the probability of achieving the goal is 1. The correctness of the latter test is implied by Lemma 5 and non-interference of relaxed actions. This leaves us with the main case in which we are uncertain about some of the subgoals. This uncertainty is either due to dependence of these subgoals on the actual initial world state, or due to achieving these subgoals using probabilistic actions, or due to both. The uncertainty about the initial state is fully captured by our weighted CNF formula $\phi(\mathcal{N}_{b_I}) \subseteq \Phi$. Likewise, the outcomes' chance propositions $\varepsilon(t')$ introduced into the implication graph by the build-timestep procedure are "chained up" in Imp to the propositions on the add lists of these outcomes, and "chained down" in *Imp* to the unknown (relaxed) conditions of these outcomes, if any. Therefore, if some action outcome ε at time $t' < t$ is relevant to achieving a subgoal $g \in G$ at time t, then the corresponding node $\varepsilon(t')$ must appear in $Imp_{\rightarrow g(t)}$, and its weight will be back-propagated by build-w-impleafs($g(t), Imp$) down to the leafs of $Imp_{\rightarrow g(t)}$. The get-P procedure then exploits these back-propagated estimates by, again, taking a heuristic assumption of independence between achieving different subgoals. Namely, the probability of achieving the unknown sub-goals $G \setminus P(t)$ is estimated by weighted model counting over the formula Φ, conjoined with probabilistic theories φ_g of achieving each unknown goal g in isolation. To understand the formulas φ_g, consider that, in order to make g true at t, we must achieve at least one of the leafs l of $Imp_{\rightarrow g(t)}$; hence the left part of the conjunction. On the other hand, if we make l true, then this achieves $g(t)$ only with (estimated) probability $\varpi_{g(t)}(l)$; hence the right part of the conjunction requires us to "pay the price" if we set l to true.[10]

As was explained at the start of this section, the positive PRPG termination test may fire even if the real goal probability is *not* high enough. That is, get-P may return a value higher than the real goal probability, due to the approximation (independence assumption) done in the weight propagation. Of course, due to the same approximation, it may also happen that get-P returns a value lower than the real goal probability.

The second PRPPG termination test comes to check whether we have reached a point in the construction of PRPG that allows us to conclude that there is no relaxed plan for $(A, \mathcal{N}_{b_I}, G, \theta)$ that starts with the given action sequence $\overline{a}$. This termination criterion asks whether, from time step t to time step $t+1$, any potentially relevant changes have occurred. A potentially relevant change would be if the goal-satisfaction probability estimate get-P grows, or if the known and unknown propositions grow, of if the support leafs of the latter propositions in *Imp* that correspond to the initial belief state grow.[11] If none occurs, then the same would hold in all future iterations $t' > t$, implying that the required goal satisfaction probability θ would never be reached. In other words, the PRPG construction is complete.

Theorem 6 *Let $(A, \mathcal{N}_{b_I}, G, \theta)$ be a probabilistic planning task, $\overline{a}$ be a sequence of actions applicable in b_I, and $|_1^+$ be a relaxation function for A. If* build-PRPG($\overline{a}, A, \phi(\mathcal{N}_{b_I}), G, \theta, |_1^+$) *returns FALSE, then there is no relaxed plan for (A, b_I, G, θ) that starts with $\overline{a}|_1^+$.*

Note that Theorem 6 holds despite the approximation done during weight propagation, making the assumption of probabilistic independence. For Theorem 6 to hold, the only requirement on the weight propagation is this: *if the real weight still grows, then the estimated weight still grows.* This requirement is met under the independence assumption. It would not be met under the assumption of co-occurence, propagating weights by maximization operations, and thereby conservatively underestimating the weights. With that propagation, if the PRPG fails then we cannot conclude that there is no plan for the respective belief. This is another good argument (besides the bad quality heuristics we observed empirically) against using the conservative estimation.

10. If we do not introduce the extra chance propositions $\langle l_g \rangle$, and instead assign the weight $\varpi_{g(t)}(l)$ to l itself, then the outcome is not correct: we have to "pay" also for setting l to false.

11. To understand the latter, note that PRPG can always be added with more and more replicas of probabilistic actions irrelevant to achieving the goals, and having effects with *known* conditions. While these action effects (since they are irrelevant) will not influence our estimate of goal-satisfaction probability, the chance propositions corresponding to the outcomes of these effects may become the support leafs of some unknown proposition p. In the latter case, the set of support leafs *support*($p(t')$) will infinitely grow with $t' \rightarrow \infty$, while the projection of *support*($p(t')$) on the initial belief state (that is, *support*($p(t)$) $\cap\, uP(t)$) is guaranteed to reach a fix point.

The full proof to Theorem 6 is given in Appendix A on pp. 615. The theorem finalizes our presentation and analysis of the process of constructing probabilistic relaxed planning graphs.

4.3 Example: PRPG Construction

To illustrate the construction of a PRPG by the algorithm in Figures 4-7, let us consider a simplification of our running Examples 1-2 in which

(i) only the actions $\{move\text{-}b\text{-}right, move\text{-}left\}$ constitute the action set A,

(ii) the goal is $G = \{r_1, b_2\}$, and the required lower bound on the probability of success $\theta = 0.9$,

(iii) the initial belief state b_I is given by the BN $\mathcal{N}_{b_I}$ as in Example 2, and

(iv) the belief state $b_{\overline{a}}$ evaluated by the heuristic function corresponds to the actions sequence $\overline{a} = \langle move\text{-}b\text{-}right \rangle$.

The effects/outcomes of the actions A considered in the construction of PRPG are described in Table 3, where e^{mbr} is a re-notation of the effect e in Table 1, the effect e' in Table 1 is effectively ignored due to the emptiness of its add effects.

a	$E(a)$	$con(e)$	$con(e)\|_1^+$	$\Lambda(e)$	$Pr(\varepsilon)$	$add(\varepsilon)$
a^{mbr} $(move\text{-}b\text{-}right)$	e^{mbr}	$\{r_1, b_1\}$	$\{r_1\}$	$\varepsilon_1^{\mathsf{mbr}}$	0.7	$\{r_2, b_2\}$
				$\varepsilon_2^{\mathsf{mbr}}$	0.2	$\{r_2\}$
				$\varepsilon_3^{\mathsf{mbr}}$	0.1	$\emptyset$
a^{ml} $(move\text{-}left)$	e^{ml}	$\{r_2\}$	$\{r_2\}$	$\varepsilon^{\mathsf{ml}}$	1.0	$\{r_1\}$
noop_{r_1}	e^{r_1}	$\{r_1\}$	$\{r_1\}$	ε^{r_1}	1.0	$\{r_1\}$
noop_{r_2}	e^{r_2}	$\{r_2\}$	$\{r_2\}$	ε^{r_2}	1.0	$\{r_2\}$
noop_{b_1}	e^{b_1}	$\{b_1\}$	$\{b_1\}$	ε^{b_1}	1.0	$\{b_1\}$
noop_{b_2}	e^{b_2}	$\{b_2\}$	$\{b_2\}$	ε^{b_2}	1.0	$\{b_2\}$

Table 3: Actions and their $|_1^+$ relaxation for the PRPG construction example.

The initialization phase of the build-PRPG procedure results in $\Phi = \phi(\mathcal{N}_{b_I})$, *Imp* $:= \emptyset$, $P(-1) = \emptyset$, and $uP(-1) = \{r_1, r_2, b_1, b_2\}$. The content of $uP(-1)$ is depicted in the first column of nodes in Figure 8. The first **for** loop of build-PRPG (constructing PRPG for the "past" layers corresponding to $\overline{a}$) makes a single iteration, and calls the build-timestep procedure with $t = -1$ and A(-1) $= \{a^{\mathsf{mbr}}\} \cup NOOPS$. (In what follows, using the names of the actions we refer to their $|_1^+$ relaxations as given in Table 3.) The add list of the outcome $\varepsilon_3^{\mathsf{mbr}}$ is empty, and thus it adds no nodes to the implication graph. Other than that, the chance nodes introduced to *Imp* by this call to build-timestep appear in the second column of Figure 8. The first outer **for** loop of build-timestep results in *Imp* given by columns 1-3 of Figure 8, $uP(0) = uP(-1)$, and no extension of Φ.

In the second outer **for** loop of build-timestep, the weight propagating procedure build-w-impleafs is called for each unknown fact $p(0) \in uP(0) = \{r_1(0), r_{2(0)}, b_1(0), b_2(0)\}$, generating the "$p(0)$-oriented" weights as in Table 4. For each $p(0) \in uP(0)$, the set of supporting leafs *support*$(p(0)) =$

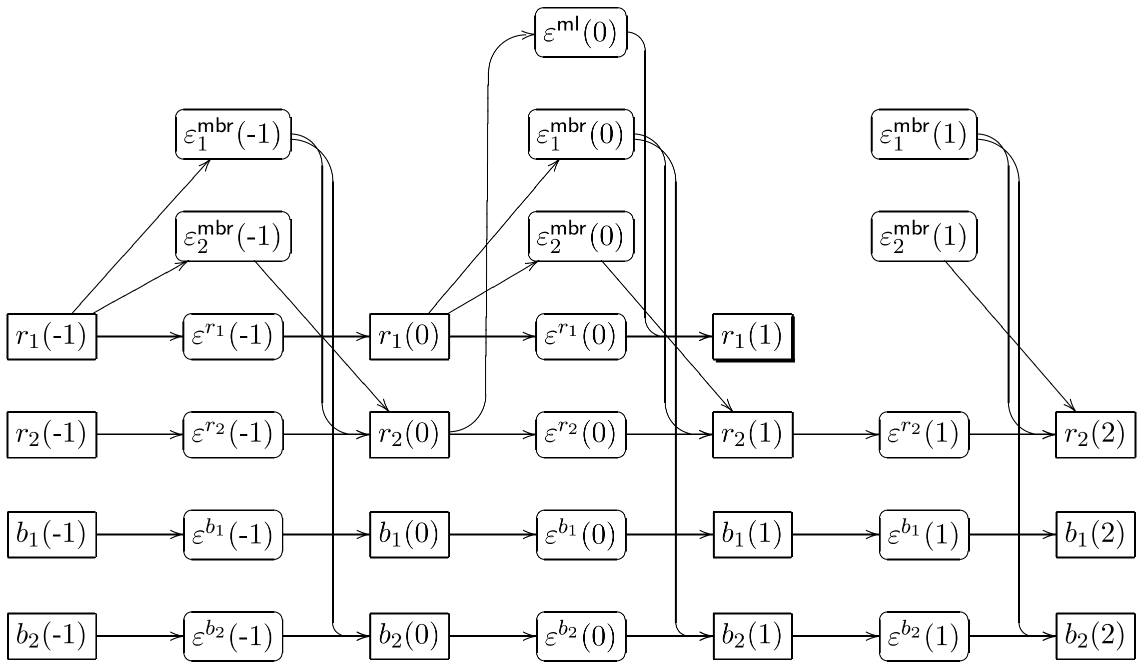


Figure 8: The implication graph *Imp*. The odd columns of nodes depict the sets of unknown propositions $uP(t)$. The even columns of nodes depict the change propositions introduced for the probabilistic outcomes of the actions $A(t)$.

$\{p(-1)\}$, none of them is implied by $\Phi = \mathcal{N}_{b_I}$, and thus the set of known facts $P(0)$ remains equal to $P(-1) = \emptyset$, and $uP(-1)$ equal to $= uP(-1)$.

	$t'=0$				$t'=-1$									
	r_1	r_2	b_1	b_2	$\varepsilon_1^{\mathsf{mbr}}$	$\varepsilon_2^{\mathsf{mbr}}$	ε^{r_1}	ε^{r_2}	ε^{b_1}	ε^{b_2}	r_1	r_2	b_1	b_2
$\varpi_{r_1(0)}$	1						1				1			
$\varpi_{r_2(0)}$		1			0.7	0.2		1			0.9	1		
$\varpi_{b_1(0)}$			1						1				1	
$\varpi_{b_2(0)}$				1	0.7					1	0.7			1

Table 4: The columns in the table correspond to the nodes in the implication graph *Imp*, and each row provides the weights $\varpi_{p(0)}$ for some $p(0) \in uP(0)$. An entry in the row of $p(0)$ is empty if and only if the node associated with the corresponding column does not belong to the implication subgraph $Imp_{\rightarrow p(0)}$.

Having finished with the **for** loop, the build-PRPG procedure proceeds with the **while** loop that builds the "future" layers of PRPG. The test of goal (un)satisficing get-$\mathsf{P}(0, G) < \theta$ evaluates to TRUE as we get get-$\mathsf{P}(0, G) = 0.63 < 0.9$, and thus the loop proceeds with its first iteration. To see the former, consider the implication graph *Imp* constructed so far (columns 1-3 in Figure 8). For our goal $G = \{r_1, b_2\}$ we have $\mathit{leafs}(\mathit{Imp}_{\rightarrow r_1(0)}) = \{r_1(-1)\}$, and $\mathit{leafs}(\mathit{Imp}_{\rightarrow b_2(0)}) = \{r_1(-1), b_2(-1)\}$. As $\{r_1(0), b_2(0)\} \subset uP(0)$ and $\Phi = \phi(\mathcal{N}_{b_I})$, we have

$$\text{get-P}(0, G) = \text{WMC}\left(\phi(\mathcal{N}_{b_I}) \wedge \varphi_{r_1} \wedge \varphi_{b_2}\right),$$

where

$$\begin{aligned} \varphi_{r_1} &= (\langle r_{1,r_1} \rangle) \wedge (r_1 \leftrightarrow \langle r_{1,r_1} \rangle), \\ \varphi_{b_2} &= (\langle r_{1,b_2} \rangle \vee \langle b_{2,b_2} \rangle) \wedge (r_1(-1) \leftrightarrow \langle r_{1,b_2} \rangle) \wedge (b_2(-1) \leftrightarrow \langle b_{2,b_2} \rangle), \end{aligned} \tag{20}$$

and

$$\begin{aligned} \varpi(\langle r_{1,r_1} \rangle) &= \varpi_{r_1(0)}(r_1(-1)) = 1 \\ \varpi(\langle b_{2,b_2} \rangle) &= \varpi_{b_2(0)}(b_2(-1)) = 1 \\ \varpi(\langle r_{1,b_2} \rangle) &= \varpi_{b_2(0)}(r_1(-1)) = 0.7 \end{aligned} \quad . \tag{21}$$

Observe that the two models of $\phi(\mathcal{N}_{b_I})$ consistent with r_2 immediately falsify the sub-formula $\phi(\mathcal{N}_{b_I}) \wedge \varphi_{r_1}$. Hence, we have

$$\begin{aligned} \text{get-P}(0, G) &= \text{WMC}\left(\phi(\mathcal{N}_{b_I}) \wedge \varphi_{r_1} \wedge \varphi_{b_2}|_{r_1(-1)=1, b_1(-1)=1}\right) + \\ &\quad\ \text{WMC}\left(\phi(\mathcal{N}_{b_I}) \wedge \varphi_{r_1} \wedge \varphi_{b_2}|_{r_1(-1)=1, b_2(-1)=1}\right) \\ &= b_I(r_1, b_1) \cdot \varpi(\langle r_{1,r_1} \rangle) \cdot \varpi(\langle r_{1,b_2} \rangle) + b_I(r_1, b_2) \cdot \varpi(\langle r_{1,r_1} \rangle) \cdot \varpi(\langle r_{1,b_2} \rangle) \cdot \varpi(\langle b_{2,b_2} \rangle) \\ &= 0.63 \cdot 1 \cdot 0.7 + 0.27 \cdot 1 \cdot 0.7 \cdot 1 \\ &= 0.63 \end{aligned}$$

In the first iteration of the **while** loop, build-PRPG calls the build-timestep procedure with $t = 0$ and $A(0) = \{a^{\mathsf{mbr}}, a^{\mathsf{ml}}\} \cup NOOPS$. The chance nodes introduced to *Imp* by this call to build-timestep appear in the forth column of Figure 8. The first outer **for** loop of build-timestep results in *Imp* given by columns 1-5 of Figure 8, $uP(1) = uP(0)$, and no extension of Φ. As before, in the second **for** loop of build-timestep, the build-w-impleafs procedure is called for each unknown fact $p(1) \in uP(1) = \{r_1(1), r_{2(1)}, b_1(1), b_2(1)\}$, generating the "$p(1)$-oriented" weights. The interesting case here is the case of weight propagation build-w-impleafs($r_1(1)$, *Imp*), resulting in weights

$$\begin{aligned} \varpi_{r_1(1)}(r_1(1)) &= 1 \\ \varpi_{r_1(1)}(\varepsilon^{\mathsf{ml}}(0)) &= 1 \\ \varpi_{r_1(1)}(\varepsilon^{r_1}(0)) &= 1 \\ \varpi_{r_1(1)}(r_1(0)) &= 1 \\ \varpi_{r_1(1)}(r_2(0)) &= 1 \end{aligned} \quad \Rightarrow \quad \begin{aligned} \varpi_{r_1(1)}(\varepsilon^{r_1}(\text{-}1)) &= 1 \\ \varpi_{r_1(1)}(\varepsilon^{r_2}(\text{-}1)) &= 1 \\ \varpi_{r_1(1)}(\varepsilon_1^{\mathsf{mbr}}(\text{-}1)) &= 0.7 \\ \varpi_{r_1(1)}(\varepsilon_2^{\mathsf{mbr}}(\text{-}1)) &= 0.2 \end{aligned} \quad \Rightarrow \quad \begin{aligned} \varpi_{r_1(1)}(r_1(\text{-}1)) &= 1 \\ \varpi_{r_1(1)}(r_2(\text{-}1)) &= 1 \end{aligned}$$

for the nodes in $\mathit{Imp}_{\rightarrow r_1(1)}$. From that, the set of supporting leafs of $r_1(1)$ is assigned to $\mathit{support}(r_1(1)) = \{r_1(-1), r_2(-1)\}$, and since $\Phi = \phi(\mathcal{N}_{b_I})$ does implies $r_1(-1) \vee r_2(-1)$, the fact r_1 is concluded to be known at time 1, and is added to $P(1)$. For all other nodes $p(1) \in uP(1)$ we still have $\mathit{support}(p(1)) = \{p(-1)\}$, and thus they all remain unknown at time $t = 1$ as well. Putting things together, this call to the build-w-impleafs procedure results with $P(1) = \{r_1(1)\}$, and

$uP(1) = \{r_{2(1)}, b_1(1), b_2(1)\}$. The **while** loop of the build-PRPG procedure proceeds with checking the fixpoint termination test, and this immediately fails due to $P(1) \neq P(0)$. Hence, the **while** loop proceeds with the next iteration corresponding to $t = 1$.

The test of goal (un)satisficing $\text{get-P}(1, G) < \theta$ still evaluates to TRUE because we have $\text{get-P}(1, G) = 0.899 < 0.9$. Let us follow this evaluation of $\text{get-P}(1, G)$ in detail as well. Considering the implication graph *Imp* constructed so far up to time $t = 1$ (columns 1-5 in Figure 8), and having $G \cap uP(1) = \{b_2(1)\}$, $\mathit{leafs}(\mathit{Imp}_{\rightarrow b_2(1)}) = \{r_1(-1), b_2(-1)\}$, and (still) $\Phi = \phi(\mathcal{N}_{b_I})$, we obtain

$$\text{get-P}(1, G) = \mathsf{WMC}\left(\phi(\mathcal{N}_{b_I}) \wedge \varphi_{b_2}\right),$$

with

$$\varphi_{b_2} = \left(\langle r_{1,b_2}\rangle \vee \langle b_{2,b_2}\rangle\right) \wedge \left(r_1(-1) \leftrightarrow \langle r_{1,b_2}\rangle\right) \wedge \left(b_2(-1) \leftrightarrow \langle b_{2,b_2}\rangle\right), \tag{22}$$

While the structure of φ_{b_2} in Equation 22 is identical to this in Equation 20, the weights associated with the auxiliary chance propositions are different, notably

$$\begin{array}{l} \varpi\left(\langle b_{2,b_2}\rangle\right) = \varpi_{b_2(1)}\left(b_2(-1)\right) = 1 \\ \varpi\left(\langle r_{1,b_2}\rangle\right) = \varpi_{b_2(1)}\left(r_1(-1)\right) = 0.91 \end{array} . \tag{23}$$

The difference in $\varpi\left(\langle r_{1,b_2}\rangle\right)$ between Equation 21 and Equation 23 stems from the fact that $r_1(-1)$ supports $b_2(1)$ not only via the effect e^{mbr} at time -1 but also via the a different instance of the same effect at time 0. Now, the only model of $\phi(\mathcal{N}_{b_I})$ that falsify φ_{b_2} is the one that sets both r_1 and b_2 to false. Hence, we have

$$\begin{aligned} \text{get-P}(1, G) &= b_I(r_1, b_1) \cdot \varpi\left(\langle r_{1,b_2}\rangle\right) + \\ &\quad\; b_I(r_1, b_2) \cdot \varpi\left(\langle r_{1,b_2}\rangle\right) \cdot \varpi\left(\langle b_{2,b_2}\rangle\right) + \\ &\quad\; b_I(r_2, b_2) \cdot \varpi\left(\langle b_{2,b_2}\rangle\right) \\ &= 0.63 \cdot 0.91 + 0.27 \cdot 0.91 \cdot 1 + 0.08 \cdot 1 \\ &= 0.899 \end{aligned}$$

Having verified $\text{get-P}(1, G) < \theta$, the **while** loop proceeds with the construction for time $t = 2$, and calls the build-timestep procedure with $t = 1$ and $A(1) = \{a^{\mathsf{mbr}}, a^{\mathsf{ml}}\} \cup NOOPS$. The chance nodes introduced to *Imp* by this call to build-timestep appear in the sixth column of Figure 8. The first outer **for** loop of build-timestep results in *Imp* given by columns 1-7 of Figure 8, and

$$\begin{aligned} \Phi = \phi(\mathcal{N}_{b_I}) &\wedge \left(\varepsilon_1^{\mathsf{mbr}}(1) \vee \varepsilon_2^{\mathsf{mbr}}(1) \vee \varepsilon_3^{\mathsf{mbr}}(1)\right) \wedge \\ &\wedge \left(\neg\varepsilon_1^{\mathsf{mbr}}(1) \vee \neg\varepsilon_2^{\mathsf{mbr}}(1)\right) \wedge \left(\neg\varepsilon_1^{\mathsf{mbr}}(1) \vee \neg\varepsilon_3^{\mathsf{mbr}}(1)\right) \wedge \left(\neg\varepsilon_2^{\mathsf{mbr}}(0) \vee \neg\varepsilon_3^{\mathsf{mbr}}(0)\right) \end{aligned} \tag{24}$$

Next, the build-w-impleafs procedure is called as usual for each unknown fact $p(2) \in uP(2) = \{r_{2(2)}, b_1(2), b_2(2)\}$. The information worth detailing here is that now we have $\mathit{leafs}(\mathit{Imp}_{\rightarrow b_2(2)}) = \{b_2(-1), r_1(-1), \varepsilon_1^{\mathsf{mbr}}(1)\}$, and $\mathit{support}(b_2(2)) = \{b_2(-1), \varepsilon_1^{\mathsf{mbr}}(1)\}$. However, we still have $\Phi \rightarrow \bigvee_{l \in \mathit{support}(p(2))} l$ for no $p(2) \in uP(2)$, and thus the set of known facts $P(2)$ remains equal to $P(1) = \{r_1\}$.

Returning from the call to the build-w-impleafs procedure, build-PRPG proceeds with checking the fixpoint termination condition. This time, the first three equalities of the condition do hold, yet the condition is not satisfied due to get-P$(2, G) >$ get-P(t, G). To see the latter, notice that we have

$$\text{get-P}(2, G) = \mathsf{WMC}\left(\Phi \wedge \varphi_{b_2}\right),$$

where Φ is given by Equation 24,

$$\varphi_{b_2} = \Big(\langle r_{1,b_2}\rangle \vee \langle b_{2,b_2}\rangle \vee \varepsilon_1^{\mathsf{mbr}}(1)\Big) \wedge (r_1(-1) \leftrightarrow \langle r_{1,b_2}\rangle) \wedge (b_2(-1) \leftrightarrow \langle b_{2,b_2}\rangle), \tag{25}$$

and

$$\begin{aligned} \varpi\left(\langle b_{2,b_2}\rangle\right) &= \varpi_{b_2(1)}\left(b_2(-1)\right) = 1 \qquad . \\ \varpi\left(\langle r_{1,b_2}\rangle\right) &= \varpi_{b_2(1)}\left(r_1(-1)\right) = 0.91 \\ \varpi(\varepsilon_1^{\mathsf{mbr}}(1)) &= \varpi_{b_2(1)}(\varepsilon_1^{\mathsf{mbr}}(1)) = 0.7 \end{aligned} \tag{26}$$

It is not hard to verify that

$$\begin{aligned} \text{get-P}(2, G) &= \text{get-P}(1, G) + b_I(r_2, b_1) \cdot \varpi(\varepsilon_1^{\mathsf{mbr}}(1)) \\ &= 0.899 + 0.02 \cdot 0.7 \\ &= 0.913 \end{aligned}$$

Note that now we do have get-P$(2, G) \geq \theta$, and therefore build-PRPG aborts the **while** loop by passing the goal satisficing test, and sets $T = 2$. This finalizes the construction of PRPG, and thus, our example.

4.4 Extracting a Probabilistic Relaxed Plan

If the construction of the PRPG succeeds in reaching the goals with the estimated probability of success get-P(T, G) exceeding θ, then we extract a relaxed plan consisting of $A' \subseteq A(0), \dots, A(T-1)$, and use the size of A' as the heuristic value of the evaluated belief state $b_{\overline{a}}$.

Before we get into the technical details, consider that there are some key differences between relaxed (no delete lists) probabilistic planning on the one hand, and both relaxed classical and relaxed qualitative conformant planning on the other hand. In relaxed probabilistic planning, it might make sense to execute the same action numerous times in consecutive time steps. In fact, this might be essential – just think of throwing a dice in a game until a "6" appears. In contrast, in the relaxed classical and qualitatively uncertain settings this is not needed – once an effect has been executed, it remains true forever. Another complication in probabilistic planning is that the required goal-achievement probability is specified over a conjunction (or, possibly, some more complicated logical combination) of different facts. While increasing the probability of achieving each individual sub-goal $g \in G$ in relaxed planning will always increase the overall probability of achieving G, choosing the right distribution of effort among the sub-goals to pass the required threshold θ for the whole goal G is a non-trivial problem.

A fundamental problem is the aforementioned lack of guarantees of the weight propagation. On the one hand, the construction of PRPG and Lemma 5 imply that $\overline{a}|_1^+$ concatenated with an arbitrary linearization $\overline{a}^R$ of $A(0), \dots, A(T-1)$ is executable in b_I. On the other hand, due to the independence assumption made in the build-w-impleafs procedure, get-P$(T, G) \geq \theta$ does *not* imply that the probability of achieving G by $\overline{a}|_1^+$ concatenated with $\overline{a}^R$ exceeds θ. A "real" relaxed plan, in that sense, might not even exist in the constructed PRPG.

Our answer to the above difficulties is to extract relaxed plans that are correct *relative to the weight propagation.* Namely, we use an implication graph "reduction" algorithm that computes a minimal subset of that graph which still – according to the weight propagation – sufficiently supports the goal. The relaxed plan then corresponds to that subset. Obviously, this "solves" the difficulty with the lack of "real" relaxed plans; we just do the relaxed plan extraction according to the independence assumption (besides ignoring deletes and removing all but one condition of each effect). The mechanism also naturally takes care of the need to apply the same action several times: this corresponds to several implication graph edges which are all needed in order to obtain sufficient weight. The choice of how effort is distributed among sub-goals is circumvented in the sense that all sub-goals are considered in conjunction, that is, the reduction is performed once and for all. Of course, there remains a choice in which parts of the implication graph should be removed. We have found that it is a useful heuristic to make this choice based on which actions have already been applied on the path to the belief. We will detail this below.

Making another assumption on top of the previous relaxations can of course be bad for heuristic quality. The "relaxed plans" we extract are not guaranteed to actually achieve the desired goal probability. Since the relaxed plans are used only for search guidance, per se this theoretical weakness is only of marginal importance. However, an over-estimation of goal probability might result in a bad heuristic because the relaxed plan does not include the right actions, or does not apply them often enough. In Section 5, we will discuss an example domain where Probabilistic-FF fails to scale for precisely this reason.

Figure 9 shows the main routine extract-PRPlan for extracting a relaxed plan from a given PRPG (note that T is the index of the highest PRPG layer, c.f. Figure 4). The sub-routines of extract-PRPlan are shown in Figures 10-11. At a high level, the extract-PRPlan procedure consists of two parts:

1. *Reduction* of the implication graph, aiming at identifying a set of time-stamped action effects that can be ignored without decreasing our estimate of goal-achievement probability get-P(T, G) below the desired threshold θ, and

2. *Extraction* of a valid relaxed plan $\overline{a}^r$ such that (schematically) constructing PRPG with $\overline{a}^r$ instead of the full set of $A(0), \ldots, A(T)$ would still result in get-P$(T, G) \geq \theta$.

The first part is accomplished by the reduce-implication-graph procedure, depicted in Figure 10. As of the first step in the algorithm, the procedure considers only the parts of the implication graph that are relevant to achieving the unknown sub-goals. Next, reduce-implication-graph performs a greedy iterative elimination of actions from the "future" layers $0, \ldots, T-1$ of PRPG until the probability estimate get-P(T, G) over the reduced set of actions goes below θ. While, in principle, any action from $A(0), \ldots, A(T-1)$ can be considered for elimination, in reduce-implication-graph we examine only *repetitions of the actions that already appear in* $\overline{a}$. Specifically, reduce-implication-graph iterates over the actions a in $\overline{a}|_1^+$, and if a repeats somewhere in the "future" layers of PRPG, then one such repetition $a(t')$ is considered for removal. If removing this repetition of a is found safe with respect to achieving θ,[12] then it is effectively removed by eliminating all the edges in *Imp* that are induced by $a(t')$. Then the procedure considers the next repetition of a. If removing another

12. Note here that the formula for WMC is constructed exactly as for the get-P function, c.f. Figure 7.

procedure extract-PRPlan($PRPG(\overline{a}, A, \phi(\mathcal{N}_{b_I}), G, \theta, |_1^+)$),
selects actions from $A(0), \ldots, A(T-1)$
Imp$'$:= reduce-implication-graph()
extract-subplan(*Imp*$'$)
sub-goal($G \cap P(T)$)
for decreasing time steps $t := T, \ldots, 1$ **do**
  for all $g \in G(t)$ **do**
    if $\exists a \in A(t-1), e \in E(a), con(e) \in P(t-1), \forall \varepsilon \in \Lambda(e) : g \in add(\varepsilon)$ **then**
      add-to-relaxed-plan one such a at time t
      sub-goal($pre(a) \cup con(e)$)
    else
      Imp$^{g(t)}$:= construct-support-graph(*support*($g(t)$))
      extract-subplan(*Imp*$^{g(t)}$)
    endif
  endfor
endfor

Figure 9: Extracting a probabilistic relaxed plan.

copy of a is not safe anymore, then the procedure breaks the inner loop and considers the next action.

procedure reduce-implication-graph()
operates on the PRPG;
returns a sub-graph of *Imp*.
Imp$' := \cup_{g \in G \setminus P(T)}$ *Imp*$_{\rightarrow g(T)}$
for all actions $a \in \overline{a}|_1^+$ **do**
  for all edges $(\varepsilon(t'), p(t'+1)) \in$ *Imp*$''$, induced by $a(t') \in A(t')$, for some $t' \geq 0$ **do**
    Imp$'' :=$ *Imp*$'$
    remove from *Imp*$''$ all the edges induced by $a \in A(t')$
    for all $g \in G \setminus P(t)$ **do**
      for each $l \in$ *leafs*(*Imp*$''_{\rightarrow g(T)}$), introduce a chance proposition $\langle l_g \rangle$ with weight $\varpi_{g(T)}(l)$
      $\varphi_g := (\bigvee_{l \in leafs(Imp''_{\rightarrow g(T)})} l) \wedge \bigwedge_{l \in leafs(Imp''_{\rightarrow g(T)}) \cap uP(-m)} (\neg l \vee \langle l_g \rangle)$
    endfor
    if WMC($\Phi \wedge \bigwedge_{g \in G \setminus P(T)} \varphi_g) \geq \theta$ **then** *Imp*$' :=$ *Imp*$''$ **else break endif**
  endfor
endfor
return *Imp*$'$

Figure 10: The procedure reducing the implication graph.

To illustrate the intuition behind our focus on the repetitions of the actions from $\overline{a}$, let us consider the following example of a simple logistics-style planning problem with probabilistic actions. Suppose we have two locations A and B, a truck that is known to be initially in A, and a heavy and uneasy to grab package that is known to be initially on the truck. The goal is to have the package unloaded in B with a reasonably high probability, and there are two actions we can use – moving the truck from A to B (a^{m}), and unloading the package (a^{u}). Moving the truck does not necessarily move the truck to B, but it does that with an extremely high probability. On the other hand, unloading the bothersome package succeeds with an extremely low probability, leaving the package on the truck otherwise. Given this data, consider the belief state $b_{\overline{a}}$ corresponding to "after trying to move the truck once", that is, to the action sequence $\langle a^{\mathsf{m}} \rangle$. To achieve the desired probability of success, the PRPG will have to be expanded to a very large time horizon T, allowing the action a^{u} to be applied sufficiently many times. However, the fact "truck in B" is not known in the belief state $b_{\overline{a}}$, and thus the implication graph will also contain the same amount of applications of a^{m}. Trimming away most of these applications of a^{m} will still keep the probability sufficiently high.

The reader might ask at this point what we hope to achieve by "trimming away most of the applications of a^{m}". The point is, intuitively, that the implication graph reduction mechanism is a means to *understand what has been accomplished already, on the path to* $b_{\overline{a}}$. Without such an understanding, the relaxed planning can be quite indiscriminative between search states. Consider the above example, and assume we have not one but two troubled packages, $P1$ and $P2$, on the truck, with unload actions $a^{\mathsf{u}1}$ and $a^{\mathsf{u}2}$. The PRPG for $b_{\overline{a}}$ contains copies of $a^{\mathsf{u}1}$ and $a^{\mathsf{u}2}$ at layers up to the large horizon T. Now, say our search starts to unload $P1$. In the resulting belief, the PRPG still has T steps because the situation has not changed for $P2$. Each step of the PRPG still contains copies of both $a^{\mathsf{u}1}$ and $a^{\mathsf{u}2}$ – and hence the heuristic value remains the same as before! In other words, without an implication graph reduction technique, relevant things that are accomplished may remain hidden behind other things that have not yet been accomplished. In the above example, this is not really critical because, as soon as we have tried an unload for *each* of $P1$ and $P2$, the time horizon T decreases by one step, and the heuristic value is reduced. It is, however, often the case that some sub-task must be accomplished before some other sub-task can be attacked. In such situations, without implication graph reduction, the search staggers across a huge plateau until the first task is completed. We observed this in a variety of benchmarks, and hence designed the implication graph reduction to make the relaxed planning aware of what has already been done.

Of course, since our weight propagation may over-estimate true probabilities, and hence over-estimate what was achieved in the past, the implication graph reduction may conclude prematurely that a sub-task has been "completed". This leads us to the main open question in this research; we will get back to this at the end of Section 5, where we discuss this in the context of an example where Probabilistic-FF's performance is bad.

Let us get back to explaining the extract-PRPlan procedure. After the implication graph reduction, the procedure proceeds with the relaxed plan extraction. The process makes use of proposition sets $G(1), \dots, G(T)$, which are used to store time-stamped sub-goals arising at layers $1 \leq t \leq T$ during the relaxed plan extraction. The sub-routine extract-subplan (Figure 11)

1. adds to the constructed relaxed plan all the time-stamped actions responsible for the edges of the reduced implication graph *Imp*′, and

2. subgoals everything outside the implication graph that condition the applicability of the effects responsible for the edges of *Imp*′.

Here and in the later phases of the process, the sub-goals are added into the sets $G(1), \dots, G(T)$ by the sub-goal procedure that simply inserts each given proposition as a sub-goal at the first layer of its appearance in the PRPG. Having accomplished this extract-and-subgoal pass of extract-subplan over *Imp*′, we also subgoal all the goal conjuncts known at time T.

In the next phase of the process, the sub-goals are considered layer by layer in decreasing order of time steps $T \geq t \geq 1$. For each sub-goal g at time t, certain supporting actions are selected into

procedure extract-subplan(Imp')
actions that are helpful for achieving uncertain goals $G \cap uP(T)$ and
subgoals all the essential conditions of these actions
for each edge $(\varepsilon(t), p(t+1)) \in Imp'$ such that $t \geq 0$ **do**
  if action a and its effect $e \in E(a)$ be responsible for ε at time t **time**
    add-to-relaxed-plan a at time t
    sub-goal($(pre(a) \cup con(e)) \cap P(t)$)
  endif endfor

procedure sub-goal(P)
inserts the propositions in P as sub-goals
at the layers of their first appearance in the PRPG
for all $p \in P$ **do**
  $t_0 := \operatorname{argmin}_t \{p \in P(t)\}$
  if $t_0 \geq 1$ **then** $G(t_0) := G(t_0) \cup \{p\}$ **endif**
endfor

procedure construct-support-graph($support(g(t))$)
takes a subset $support(g(t))$ of $leafs(Imp_{\to g(t)})$ weighted according to $g(t)$;
returns a sub-graph Imp' of Imp.
$Imp' := \emptyset$
$open := support(g(t))$
while $open \neq \emptyset$ **do**
  $open := open \setminus \{p(t')\}$
  choose $a \in A(t'), e \in E(a), con(e) = \{p\}$ such that
    $\forall \varepsilon \in \Lambda(e) : (p(t'), \varepsilon(t')) \in Imp_{g(t)} \wedge \varpi_{g(t)}(\varepsilon(t')) = \varpi(\varepsilon(t'))$
  for each $\varepsilon \in \Lambda(e)$ **do**
    choose $q \in add(\varepsilon)$ such that $\varpi_{g(t)}(q(t'+1)) = 1$
    $Imp' := Imp' \cup \{(p(t'), \varepsilon(t')), (\varepsilon(t'), q(t'+1))\}$
    $open := open \cup \{q(t'+1)\}$
  endfor endwhile
return Imp'

Figure 11: Sub-routines for extract-PRPlan.

the relaxed plan. If there is an action a and some effect $e \in E(a)$ that are known to be applicable at time $t-1$, and guarantee to achieve g with certainty, then a is added to the constructed relaxed plan at $t-1$. Otherwise, we

1. use the construct-support-graph procedure to extract a sub-graph $Imp^{g(t)}$ consisting of a set of implications that together ensure achieving g at time t, and

2. use the already discussed procedure extract-subplan to

   (a) add to the constructed relaxed plan all the time-stamped actions responsible for the edges of $Imp^{g(t)}$, and

   (b) subgoal everything outside this implication graph $Imp^{g(t)}$ that condition the applicability of the effects responsible for the edges of $Imp^{g(t)}$.

Processing this way all the sub-goals down to $G(1)$ finalizes the extraction of the relaxed plan estimate. Section 4.5 provides a detailed illustration of this process on the PRPG constructed in Section 4.3. In any event, it is easy to verify that the relaxed plan we extract is sound relative to our weight propagation, in the following sense.

Proposition 7 *Let* $(A, \mathcal{N}_{b_I}, G, \theta)$ *be a probabilistic planning task,* $\overline{a}$ *be a sequence of actions applicable in* b_I*, and* $|_1^+$ *be a relaxation function for* A *such that* build-PRPG$(\overline{a}, A, \phi(\mathcal{N}_{b_I}), G, \theta, |_1^+)$ *returns TRUE. Let* $A(0)^s, \ldots, A(T-1)^s$ *be the actions selected from* $A(0), \ldots, A(T-1)$ *by* extract-PRPlan*. When constructing a relaxed planning graph using only* $A(0)^s, \ldots, A(T-1)^s$*, then* get-P$(T, G) \geq \theta$*.*

Proof: By construction: reduce-implication-graph leaves enough edges in the graph so that the weight propagation underlying get-P still concludes that the goal probability is high enough. ■

4.5 Example: Extracting a Relaxed Plan from PRPG

We illustrate the process of the relaxed plan extraction on the PRPG as in Figure 8, constructed for the belief state and problem specification as in example in Section 4.3. In this example we have $T = 2$, $G \cap uP(2) = \{b_2\}$, and thus the implication graph *Imp* gets immediately reduced to its sub-graph *Imp*$'$ depicted in Figure 12a. As the plan $\overline{a}$ to the belief state in question consists of only a single action a^{mbr}, the only action instances that are considered for elimination by the outer **for** loop of reduce-implication-graph are $a^{\mathsf{mbr}}(0)$ and $a^{\mathsf{mbr}}(1)$. If $a^{\mathsf{mbr}}(0)$ is chosen to be examined, then the implication sub-graph *Imp*$'' =$ *Imp*$'$ is further reduced by removing all the edges due to $a^{\mathsf{mbr}}(0)$, and the resulting *Imp*$''$ appears[13] in Figure 12b. The Φ and φ_{b_2} components of the evaluated formula $\Phi \wedge \varphi_{b_2}$ are given by Equation 24 and Equation 25, respectively, and the weights associated with the chance propositions in Equation 25 over the reduced implication graph *Imp*$''$ are

$$\begin{aligned} \varpi\left(\langle b_{2,b_2}\rangle\right) &= \varpi_{b_2(1)}\left(b_2(-1)\right) = 1 \\ \varpi\left(\langle r_{1,b_2}\rangle\right) &= \varpi_{b_2(1)}\left(r_1(-1)\right) = 0.7 \\ \varpi(\varepsilon_1^{\mathsf{mbr}}(1)) &= \varpi_{b_2(1)}(\varepsilon_1^{\mathsf{mbr}}(1)) = 0.7 \end{aligned} \quad . \tag{27}$$

The weight model counting of $\Phi \wedge \varphi_{b_2}$ evaluates to $0.724 < \theta$, and thus *Imp*$''$ does not replace *Imp*$'$. The only alternative action removal is this of $a^{\mathsf{mbr}}(1)$, and it can be seen from the example in Section 4.3 that this attempt for action elimination will also result in probability estimate lower than θ. Hence, the only effect of reduce-implication-graph on the PRPG processed by the extract-PRPlan procedure is the reduction of the implication graph to only the edges relevant to achieving $\{b_2\}$ at time $T = 2$. The reduced implication sub-graph *Imp*$'$ returned by the reduce-implication-graph procedure is depicted in Figure 12a.

Next, the extract-subplan procedure iterates over the edges of *Imp*$'$ and adds to the initially empty relaxed plan applications of a^{mbr} at times 0 and 1. The action a^{mbr} has no preconditions, and the condition r_1 of the effect $\varepsilon_1^{\mathsf{mbr}} \in E(a^{\mathsf{mbr}})$ is known at time 1. Hence, extract-subplan invokes the sub-goal procedure on $\{r(1)\}$, and the latter is added into the proposition set $G(1)$. The subsequent call sub-goal$(G \cap P(T))$ = sub-goal$(\{r_1\})$ leads to no further extensions of $G(2), G(1)$

13. The dashed edges in Figure 12b can be removed from *Imp*$''$ either now or at a latter stage if *Imp*$''$ is chosen to replace *Imp*$'$.

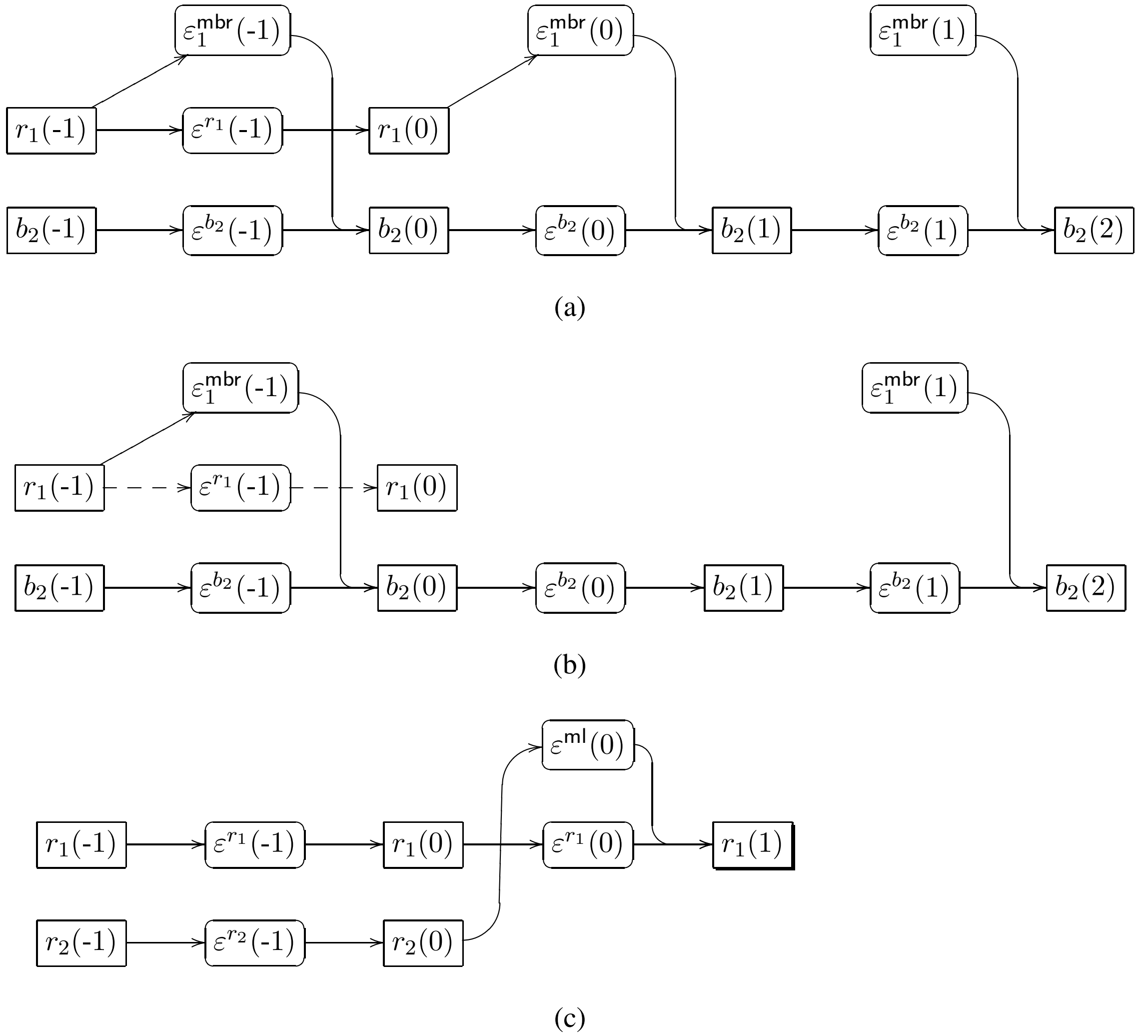


Figure 12: Illustrations for various steps of the relaxed plan extraction from the PRPG constructed in Section 4.3, and, in particular, from the implication graph of the latter, depicted in Figure 8.

as we already have $r_1 \in G(1)$. Hence, the outer **for** loop of extract-PRPlan starts with $G(2) = \emptyset$, and $G(1) = \{r_1\}$.

Since $G(2)$ is empty, the first sub-goal considered by extract-PRPlanis r_1 from $G(1)$. For r_1 at time 1, no action effect at time 0 passes the test of the **if** statement—the condition r_2 of $\varepsilon^{\mathsf{ml}}$ is not known at time 0, and the same is true[14] for ε^{r_1}. Hence, the subgoal $r_1(1)$ is processed by extracting a sub-plan to support achieving it with certainty. First, the construct-support-graph procedure is called with *support*$(r_1(1)) = \{r_1(-1), r_2(-1)\}$ (see Section 4.3). The extracted sub-

14. In fact, it is easy to see from the construction of the sub-goal procedure that if p belongs to $G(t)$, then the condition of the noop's effect ε^p cannot be known at time $t - 1$.

graph $\mathit{Imp}^{r_1(1)}$ of the original implication graph Imp is depicted in Figure 12c, and invoking the procedure extract-subplan on $\mathit{Imp}^{r_1(1)}$ results in adding (i) application of a^{ml} at time 0, and (ii) no new subgoals. Hence, the proposition sets $G(1), G(2)$ get emptied, and thus we end up with extracting a relaxed plan $\langle a^{\mathsf{mbr}}(0), a^{\mathsf{ml}}(0), a^{\mathsf{mbr}}(1)\rangle$.

5. Empirical Evaluation

We have implemented Probabilistic-FF in C, starting from the Conformant-FF code. With $\theta = 1.0$, Probabilistic-FF behaves exactly like Conformant-FF (except that Conformant-FF cannot handle non-deterministic effects). Otherwise, Probabilistic-FF behaves as described in the previous sections, and uses Cachet (Sang et al., 2005) for the weighted model counting. To better home in on strengths and weaknesses of our approach, the empirical evaluation of Probabilistic-FF has been done in two steps. In Section 5.1 we evaluate Probabilistic-FF on problems having non-trivial uncertain initial states, but only deterministic actions. In Section 5.2 we examine Probabilistic-FF on problems with probabilistic action effects, and with both sources of uncertainty. We compare Probabilistic-FF's performance to that of the probabilistic planner POND (Bryce et al., 2006). The reasons for choosing POND as the reference point are twofold. First, similarly to Probabilistic-FF, POND constitutes a forward-search planner guided by a non-admissible heuristic function based on (relaxed) planning graph computations. Second, to our knowledge, POND clearly is the most efficient probabilistic planner reported in the literature.[15]

The experiments were run on a PC running at 3GHz with 2GB main memory and 2MB cache running Linux. Unless stated otherwise, each domain/problem pair was tried at four levels of desired probability of success $\theta \in \{0.25, 0.5, 0.75, 1.0\}$. Each run of a planner was time-limited by 1800 seconds of user time. Probabilistic-FF was run in the default configuration inherited from FF, performing one trial of enforced hill-climbing and switching to best-first search in case of failure. In domains without probabilistic effects, we found that Probabilistic-FF's simpler relaxed plan extraction developed for that case (Domshlak & Hoffmann, 2006), performs better than the one described in here. We hence switch to the simpler version in these domains.[16]

Unlike Probabilistic-FF, the heuristic computation in POND has an element of randomization; namely, the probability of goal achievement is estimated via sending a set of random particles through the relaxed planning graph (the number of particles is an input parameter). For each problem instance, we averaged the runtime performance of POND over 10 independent runs. In special cases where POND timed out on some runs for a certain problem instance, yet not on all of the 10 runs, the average we report for POND uses the lower-bounding time threshold of 1800s to replace the missing time points. In some cases, POND's best-case performance differs a lot from its average performance; in these cases, the best-case performance is also reported. We note that, following the suggestion of Dan Bryce, POND was run in its default parameter setting, and, in par-

15. In our experiments we have used a recent version 2.1 of POND that significantly enhances POND2.0 (Bryce et al., 2006). The authors would like to thank Dan Bryce and Rao Kambhampati for providing us with a binary distribution of POND2.1.

16. Without probabilistic effects, relaxed plan extraction proceeds very much like in Conformant-FF, with an additional straightforward backchaining selecting support for the unknown goals. The more complicated techniques developed in here to deal with relaxed plan extraction under probabilistic effects appear to have a more unstable behavior than the simpler techniques. If there *are* probabilistic effects, then the simple backchaining is not meaningful because it has no information on how many times an action must be applied in order to sufficiently support the goal.

		$\theta = 0.25$	$\theta = 0.5$	$\theta = 0.75$	$\theta = 1.0$
Instance	#actions/#facts/#states	$t/\|S\|/l$	$t/\|S\|/l$	$t/\|S\|/l$	$t/\|S\|/l$
Safe-uni-70	70/71/140	1.39/19 /18	4.02/36/35	8.06/54/53	4.62/71 /70
Safe-cub-70	70/70/138	0.28/6/5	0.76/13/12	1.54/22/21	4.32/70/69
Cube-uni-15	6/90/3375	3.25/145/26	3.94/150/34	5.00/169/38	25.71/296/42
Cube-cub-15	6/90/3375	0.56/41/8	1.16/70/13	1.95/109/18	26.35/365/42
Bomb-50-50	2550/200/$> 2^{100}$	0.01/1/0	0.10/17/16	0.25/37/36	0.14/51/50
Bomb-50-10	510/120/$> 2^{60}$	0.00/1/0	0.89/248/22	4.04/778/62	1.74/911/90
Bomb-50-5	255/110/$> 2^{55}$	0.00/1/0	1.70/468/27	4.80/998/67	2.17/1131/95
Bomb-50-1	51/102/$> 2^{51}$	0.00/1/0	2.12/662/31	6.19/1192/71	2.58/1325/99
Log-2	3440/1040/$> 20^{10}$	0.90/117/54	1.07/152/62	1.69/205/69	1.84/295/78
Log-3	3690/1260 /$> 30^{10}$	2.85/159/64	8.80/328/98	4.60/336/99	4.14/364/105
Log-4	3960/1480/$> 40^{10}$	2.46/138/75	8.77/391/81	6.20/377/95	8.26/554/107
Grid-2	2040/825 /$> 36^{10}$	0.07/39/21	1.35/221/48	6.11/1207/69	6.14/1207/69
Grid-3	2040/841 /$> 36^{10}$	16.01/1629/76	15.8/1119/89	82.24/3974/123	66.26/3974/123
Grid-4	2040/857 /$> 36^{10}$	28.15/2167/96	51.58/2541/111	50.80/2541/115	193.47/6341/155
Rovers-7	393/97 /$> 6^3 * 3^8$	0.01/ 37/18	0.01/ 37/18	0.01/ 37/18	0.01/ 37/18
RoversP-7	393/133 /$> 6^3 * 3^8$	2.15/942/65	2.23/983/75	2.37/1008/83	2.29/1008/83
RoversPP-7	393/133 /$> 6^3 * 3^8$	8.21/948/65	12.48/989/75	12.53/994/77	16.20/1014/83
RoversPPP-7	395/140 /$> 6^3 * 3^8$	25.77/950/67	41.18/996/79	0.01/UNSAT	0.01/UNSAT

Table 5: Empirical results for problems with probabilistic initial states. Times t in seconds, search space size $|S|$ (number of calls to the heuristic function), plan length l.

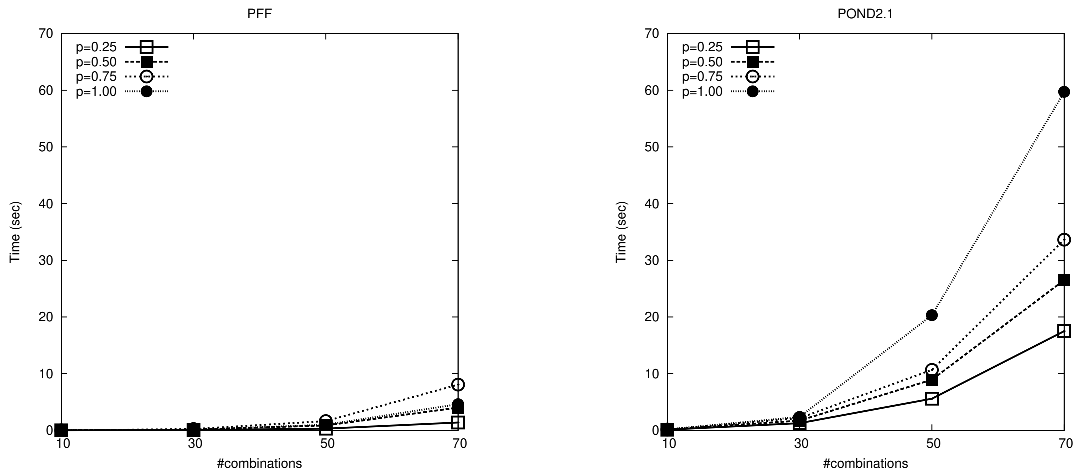
ticular, this includes the number of random particles (64) selected for computing POND's heuristic estimate (Bryce et al., 2006).

5.1 Initial State Uncertainty and Deterministic Actions

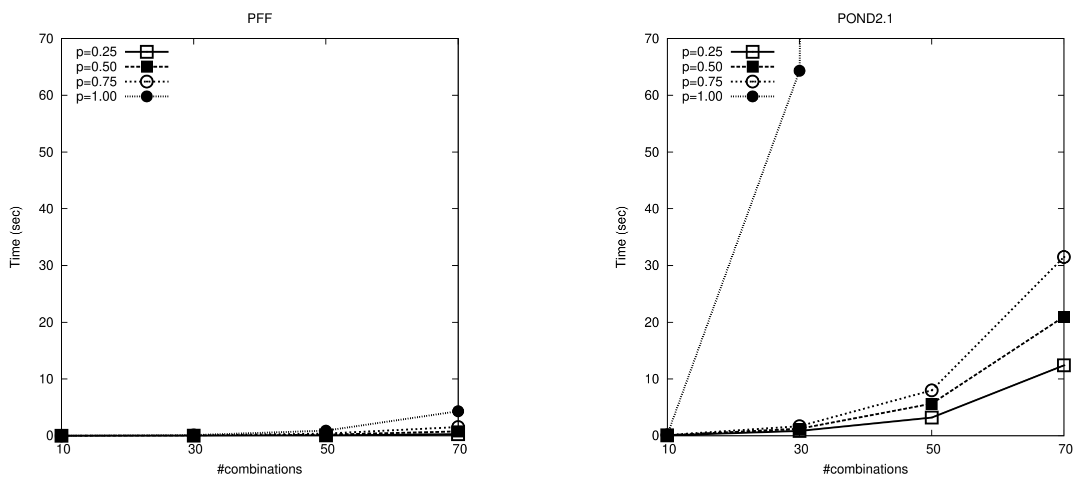
We now examine the performance of Probabilistic-FF and POND in a collection of domains with probabilistic initial states, but with deterministic action effects. We will consider the domains one by one, discussing for each a set of runtime plots. For some of the problem instances, Table 5 shows more details, providing features of the instance size as well as detailed results for Probabilistic-FF, including the number of explored search states and the plan length.

Our first three domains are probabilistic versions of traditional conformant benchmarks: "Safe", "Cube", and "Bomb". In Safe, out of n combinations one opens the safe. We are given a probability distribution over which combination is the right one. The only type of action in Safe is trying a combination, and the objective is to open the safe with probability $\geq \theta$. We experimented with two probability distributions over the n combinations, a uniform one ("Safe-uni") and a distribution that declines according to a cubic function ("Safe-cub"). Table 5 shows that Probabilistic-FF can solve this very efficiently even with $n = 70$. Figure 13 compares between Probabilistic-FF and POND, plotting their time performance on an identical linear scale, where x-axes show the number of combinations.

From the graphs it is easy to see that Probabilistic-FF outperforms POND by at least an order of magnitude on both Safe-uni and Safe-cub. But a more interesting observation here is not necessarily the difference in time performance, but the relative performance of each planner on Safe-uni and Safe-cub. Note that Safe-cub is somewhat "easier" than Safe-uni in the sense that, in Safe-cub, fewer combinations must be tried to guarantee a given probability θ of opening the safe. This because the



(a) Uniform prior distribution over the combinations.



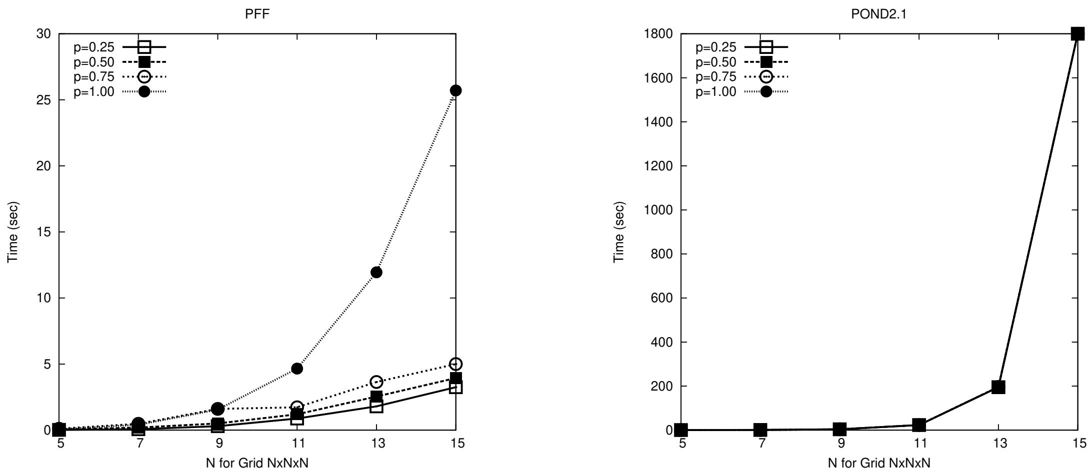
(b) Cubic decay prior distribution over the combinations.

Figure 13: The Safe domain, Probabilistic-FF (left) vs. POND (right).

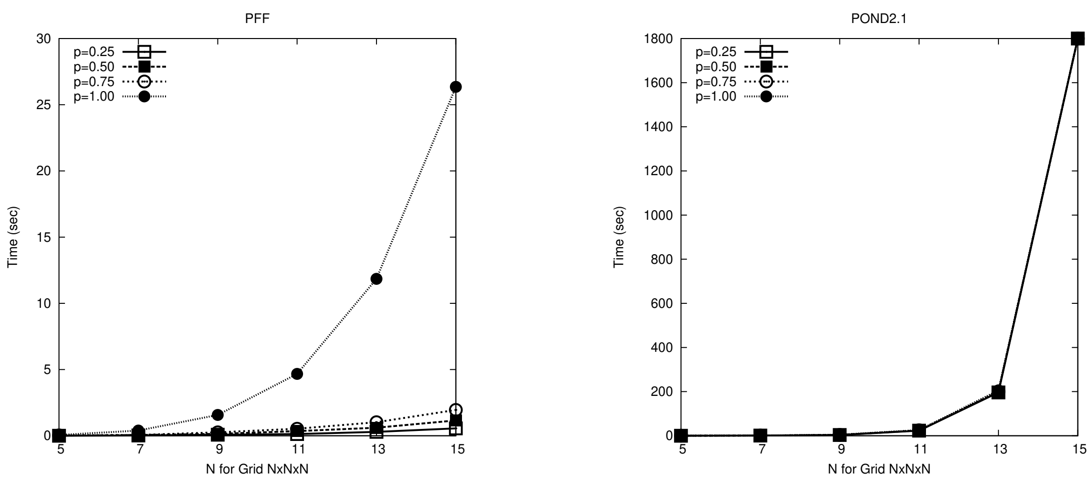
dominant part of the probability mass lies on the combinations at the head of the cubic distribution (the last combination has probability 0 to be the right combination, and thus it needs not be tried even when $\theta = 1.0$). The question is now whether the heuristic functions of Probabilistic-FF and POND exploit this difference between Safe-uni and Safe-cub. Table 5 and Figure 13 provide an affirmative answer for this question for the heuristic function of Probabilistic-FF. The picture with POND was less clear as the times spent by POND on (otherwise identical) instances of Safe-uni and Safe-cub were roughly the same.[17]

Another interesting observation is that, for both Probabilistic-FF and POND, moving from $\theta = 1.0$ to $\theta < 1.0$, that is, from planning with qualitative uncertainty to truly probabilistic planning,

17. On Safe-cub with $n = 70$ and $\theta \in \{0.75, 1.0\}$, POND undergoes an exponential blow-up that is not shown in the graphs since these data points would obscure the other data points; anyway, we believe that this blow-up is due only to some unfortunate troubles with numerics.



(a) Uniform prior distribution over the initial position.



(b) Cubic decay prior distribution over the initial position.

Figure 14: The Cube domain, Probabilistic-FF (left) vs. POND (right).

typically did not result in a performance decline. We even get *improved* performance (except for $\theta = 0.75$ in Safe-uni). The reason seems to be that the plans become shorter. This trend can be observed also in most other domains. The trend is particularly remarkable for Probabilistic-FF, since moving from $\theta = 1.0$ to $\theta < 1.0$ means to move from a case where no model counting is needed to a case where it is needed. (In other words, Probabilistic-FF automatically "specializes" itself for the qualitative uncertainty, by not using the model counting. To our knowledge, the same is not true of POND, which uses the same techniques in both cases.)

In Cube, the task is to move into a corner of a 3-dimensional grid, and the actions correspond to moving from the current cube cell to one of the (up to 6) adjacent cube cells. Again, we created problem instances with uniform and cubic distributions (over the initial position in each dimension), and again, Probabilistic-FF scales well, easily solving instances on a $15 \times 15 \times 15$ cube. Within our time limit, POND was capable of solving Cube problems with cube width ≤ 13. Figure 14 compares between Probabilistic-FF and POND in more detail, plotting their time performance on *different* linear scales (with x-axes capturing the width of the grid in each dimension), and showing at least an order of magnitude advantage for Probabilistic-FF. Note that,

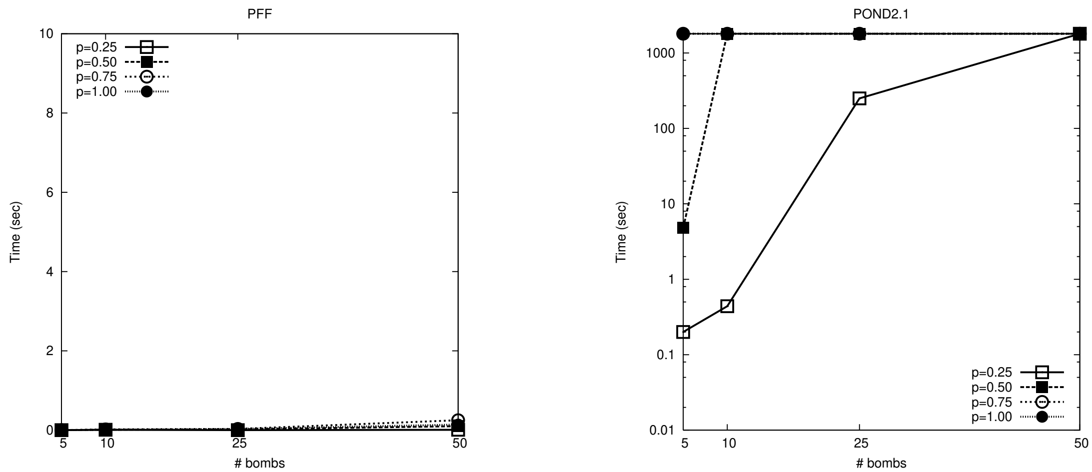
- Probabilistic-FF generally becomes faster with decreasing θ (with decreasing hardness of achieving the objective), while θ does not seem to have a substantial effect on the performance of POND,
- Probabilistic-FF exploits the relative easiness of Cube-cub (e.g., see Table 5), while the time performance of POND on Cube-cub and Cube-uni is qualitatively identical.

We also tried a version of Cube where the task is to move into the grid *center*. Probabilistic-FF is bad at doing so, reaching its performance limit at $n = 7$. This weakness in the Cube-center domain is inherited from Conformant-FF. As detailed by Hoffmann and Brafman (2006), the reason for the weakness lies in the inaccuracy of the heuristic function in this domain. There are two sources of this inaccuracy. First, to solve Cube-center in reality, one must start with moving into a corner in order to establish her position; in the relaxation, without delete lists, this is not necessary. Second, the relaxed planning graph computation over-approximates not only what can be achieved in future steps, but also what has already been achieved on the path to the considered belief state. For even moderately long paths of actions, the relaxed planning graph comes to the (wrong) conclusion that the goal has already been achieved, so the relaxed plan becomes empty and there is no heuristic information.

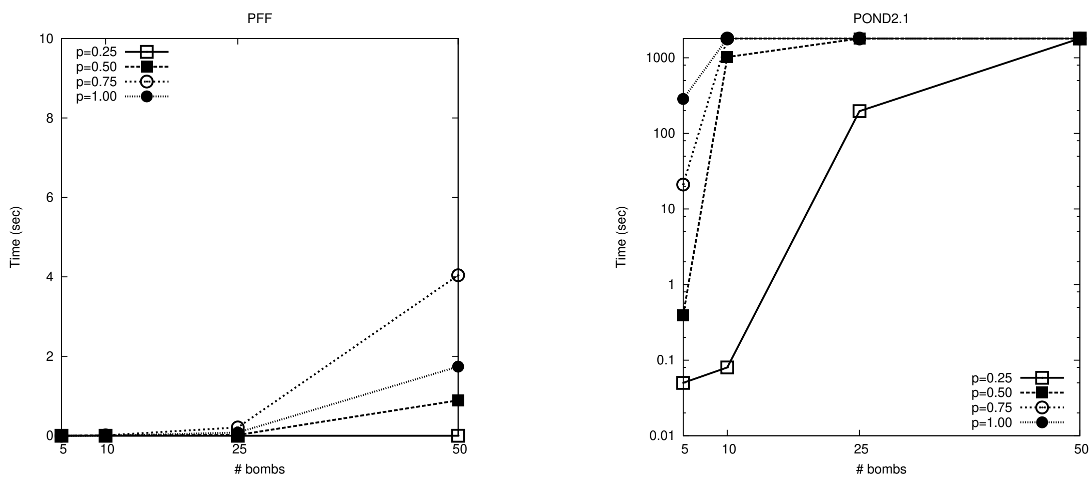
Next we consider the famous Bomb-in-the-Toilet domain (or Bomb, for short). Our version of Bomb contains n bombs and m toilets, where each bomb may be armed or not armed *independently* with probability $1/n$, resulting in huge numbers of initially possible world states. Dunking a bomb into an unclogged toilet disarms the bomb, but clogs the toilet. A toilet can be unclogged by flushing it. Table 5 shows that Probabilistic-FF scales nicely to $n = 50$, and becomes faster as m increases. The latter is logical and desirable as having more toilets means having more “disarming devices”, resulting in shorter plans needed. Figures 15 and 16 compare between Probabilistic-FF and POND, plotting the time performance of Probabilistic-FF on a linear scale, and that of POND on a logarithmic scale. The four pairs of graphs correspond to four choices of number of toilets $m \in \{50, 10, 5, 1\}$. The x-axes in all these graphs correspond to the number of potentially armed bombs, where we checked problems with $n \in \{5, 10, 25, 50\}$. Figure 15 shows that this time Probabilistic-FF is at least four orders of magnitude faster than POND; At the extremes, while the hardest combination of $n = 50$, $m = 1$, and $\theta = 0.75$ took Probabilistic-FF less than 7 seconds, POND timed-out on most of the problem instances. In addition,

- In Bomb as well, Probabilistic-FF exhibit the nice pattern of improved performance as we move from non-probabilistic ($\theta = 1.0$) to probabilistic planning (specifically, $\theta \leq 0.5$; for $\theta \leq 0.25$, the initial state is good enough already).
- While the performance of Probabilistic-FF improves with the number of toilets, POND seems to exhibit the inverse dependence, that is, being more sensitive to the number of states in the problem (see Table 5) rather to the optimal solution depth.

Finally, we remark that, though length-optimality is not explicitly required in probabilistic conformant planning, for all of Safe, Cube, and Bomb, Probabilistic-FF’s plans are optimal (the shortest possible).



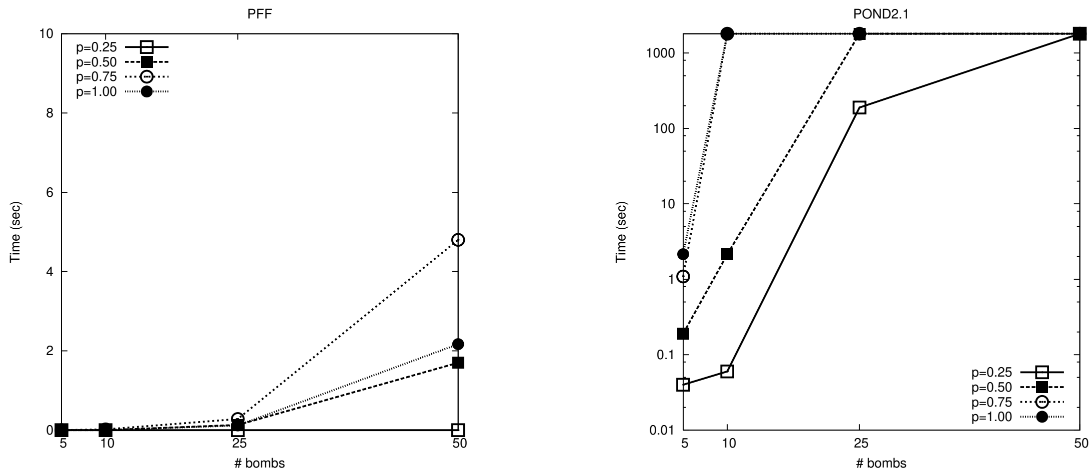
(a) 50 toilets



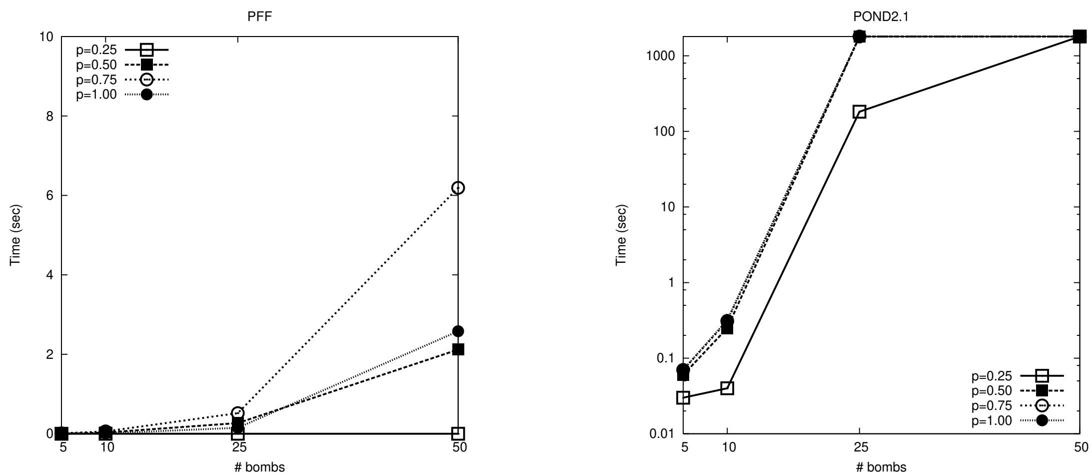
(b) 10 toilets

Figure 15: The Bomb domain, Probabilistic-FF (left) vs. POND (right).

Our next three domains are adaptations of benchmarks from deterministic planning: “Logistics”, “Grid”, and “Rovers”. We assume that the reader is familiar with these domains. Each Logistics-x instance contains 10 cities, 10 airplanes, and 10 packages, where each city has x locations. The packages are with chance 0.88 at the airport of their origin city, and uniformly at any of the other locations in that city. The effects of all loading and unloading actions are conditional on the (right) position of the package. Note that higher values of x increase not only the space of world states, but also the initial uncertainty. Grid is the complex grid world run in the AIPS’98 planning competition (McDermott, 1998), featuring locked positions that must be opened with matching keys. Each Grid-x here is a modification of instance nr. 2 (of 5) run at AIPS’98, with a 6×6 grid, 8 locked positions, and 10 keys of which 3 must be transported to a goal position. Each lock has x possible, uniformly distributed shapes, and each of the 3 goal keys has x possible, uniformly distributed initial positions. The effects of pickup-key, putdown-key, and open-lock actions are conditional.



(c) 5 toilets



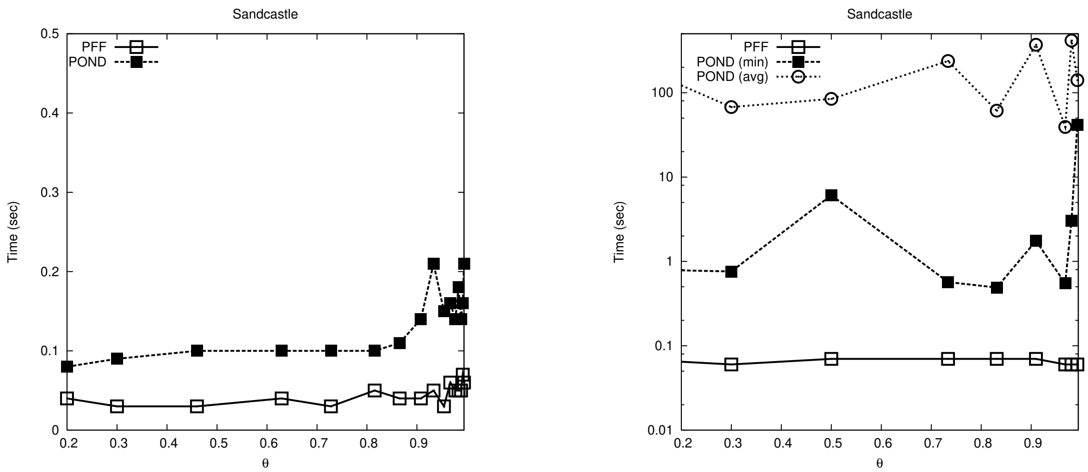
(d) 1 toilet

Figure 16: The Bomb domain, Probabilistic-FF (left) vs. POND (right).

Finally, our last set of problems comes from three cascading modifications of instance nr. 7 (of 20) of the Rovers domain used at the AIPS'02 planning competition. This problem instance has 6 waypoints, 3 rovers, 2 objectives, and 6 rock/soil samples. From Rovers to RoversPPP we modify the instance/domain as follows.

- Rovers is the original AIPS'02 problem instance nr. 7, and we use it hear mainly for comparison.

- In RoversP, each sample is with chance 0.8 at its original waypoint, and with chance 0.1 at each of the others two waypoints. Each objective may be visible from 3 waypoints with uniform distribution (this is a probabilistic adaptation of the domain suggested by Bryce & Kambhampati, 2004).



(a) (b)

Figure 17: Probabilistic-FF and POND on problems from (a) Sand-Castle, and (b) Slippery-Gripper.

- RoversPP enhances RoversP by *conditional* probabilities in the initial state, stating that whether or not an objective is visible from a waypoint depends on whether or not a rock sample (intuition: a large piece of rock) is located at the waypoint. The probability of visibility is much higher if the latter is not the case. Specifically, the visibility of each objective depends on the locations of two rock samples, and if a rock sample is present, then the visibility probability drops to 0.1.

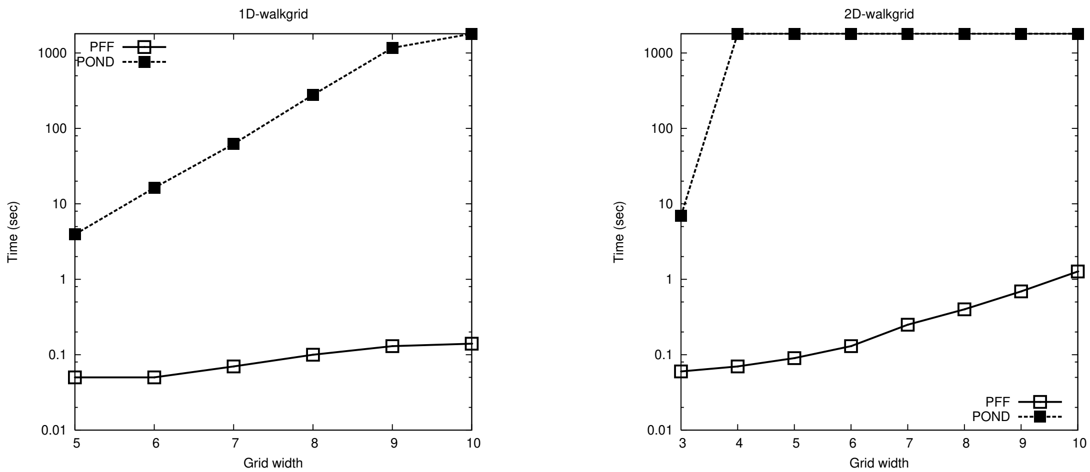
- RoversPPP extends RoversPP by introducing the need to collect data about water existence. Each of the soil samples has a certain probability (< 1) to be "wet". For communicated sample data, an additional operator tests whether the sample was wet. If so, a fact "know-that-water" contained in the goal is set to true. The probability of being wet depends on the location of the sample.

We show no runtime plots for Logistics, Grid, and Rovers, since POND runs out of either time or memory on all considered instances of these domains. Table 5 shows that the scaling behavior of Probabilistic-FF in these three domains is similar to that observed in the previous domains. The goals in the RoversPPP problem cannot be achieved with probabilities $\theta \in \{0.75, 1.0\}$. This is proved by Probabilistic-FF's *heuristic function*, providing the correct answer in split seconds.

5.2 Probabilistic Actions

Our first two domains with probabilistic actions are the famous "Sand-Castle" (Majercik & Littman, 1998) and "Slippery-Gripper" (Kushmerick et al., 1995) domains. The domains are simple, but they posed the first challenges for probabilistic planners; our performance in these domains serves an indicator of the progress relative to previous ideas for probabilistic planning.

In Sand-Castle, the states are specified by two boolean variables $moat$ and $castle$, and state transitions are given by two actions $dig\text{-}moat$ and $erect\text{-}castle$. The goal is to erect the castle.



(a) (b)

Figure 18: Probabilistic-FF and POND on problems from (a) 1D-WalkGrid with $\theta = 0.9$, and (b) 2D-WalkGrid with $\theta = 0.01$.

Building a moat with $dig\text{-}moat$ might fail with probability 0.5. Erecting a castle with $erect\text{-}castle$ succeeds with probability 0.67 if the moat has already been built, and with probability 0.25, otherwise. If failed, $erect\text{-}castle$ also destroys the moat with probability 0.5. Figure 17(a) shows that both Probabilistic-FF and POND solve this problem in less than a second for arbitrary high values of θ, with the performance of both planners being almost independent of the required probability of success.

Slippery-Gripper is already a bit more complicated domain. The states in Slippery-Gripper are specified by four boolean variables $grip\text{-}dry$, $grip\text{-}dirty$, $block\text{-}painted$, and $block\text{-}held$, and there are four actions dry, $clean$, $paint$, and $pickup$. In the initial state, the block is neither painted nor held, the gripper is clean, and the gripper is dry with probability 0.7. The goal is to have a clean gripper holding a painted block. Action dry dries the gripper with probability 0.8. Action $clean$ cleans the gripper with probability 0.85. Action $paint$ paints the block with probability 1, but makes the gripper dirty with probability 1 if the block was held, and with probability 0.1 if it was not. Action $pickup$ picks up the block with probability 0.95 if the gripper is dry, and with probability 0.5 if the gripper is wet.

Figure 17(b) depicts (on a log-scale) the relative performance of Probabilistic-FF and POND on Slippery-Gripper as a function of growing θ. The performance of Probabilistic-FF is nicely flat around 0.06 seconds. This time, the comparison with POND was somewhat problematic, because, for any fixed θ, POND on Slippery-Gripper exhibited a huge variance in runtime. In Figure 17(b) we plot the best runtimes for POND, as well as its average runtimes. The best run-times for POND for different values of θ vary around a couple of seconds, but the average runtimes are significantly worse. (For some high values of θ POND timed-out on some sample runs, and thus the plot provides a lower bound on the average runtimes.)

In the next two domains, "1D-WalkGrid" and "2D-WalkGrid", the robot has to pre-plan a sequence of conditional movements taking it from a corner of the grid to the farthest (from the initial position) corner (Hyafil & Bacchus, 2004). In 1D-WalkGrid the grid is one-dimensional, while in 2D-WalkGrid the grid is two-dimensional. Figure 18(a) depicts (on a log-scale) a snapshot of the relative performance of Probabilistic-FF and POND on one-dimensional grids of width n and $\theta = 0.9$. The robot is initially at $(1, 1)$, should get to $(1, n)$, and it can try moving in each of the two possible directions. Each of the two movement actions moves the robot in the right direction with probability 0.8, and keeps it in place with probability 0.2. It is easy to see from Figure 18(a) that the difference between the two planners in this domain is substantial—while runtime of Probabilistic-FF grows only linearly with x, the same dependence for POND is seemingly exponential.

The 2D-WalkGrid domain is already much more challenging for probabilistic planning. In all 2D-WalkGrid problems with $n \times n$ grids the robot is initially at $(1, 1)$, should get to (n, n), and it can try moving in each of the four possible directions. Each of the four movement actions advances the robot in the right direction with probability 0.8, in the opposite direction with probability 0, and in either of the other two directions with probability 0.1. Figure 18(a) depicts (on a log-scale) a snapshot of the relative performance of Probabilistic-FF and POND on 2D-WalkGrid with very low required probability of success $\theta = 0.01$, and this as a function of the grid's width n. The plot shows that Probabilistic-FF still scales well with increasing n (though not linearly anymore), while POND time-outs for all grid widths $n > 3$. For higher values of θ, however, Probabilistic-FF does reach the time-out limit on rather small grids, notably $n = 6$ and $n = 5$ for $\theta = 0.25$ and $\theta = 0.5$, respectively. The reason for this is that Probabilistic-FF's heuristic function is not good enough at estimating how many times, at an early point in the plan, a probabilistic action must be applied in order to sufficiently support a high goal threshold at the end of the plan. We explain this phenomenon in more detail at the end of this section, where we find that it also appears in a variant of the well-known Logistics domain.

Our last set of problems comes from the standard Logistics domain. Each problem instance x-y-z contains x locations per city, y cities, and z packages. We will see that Probabilistic-FF scales much worse, in Logistics, in the presence of probabilistic effects than if there is "only" initial state uncertainty (we will explain the reason for this at the end of this section). Hence we use much smaller instances than the ones used above in Section 5.1. Namely, to allow a direct comparison to previous results in this domain, we closely follow the specification of Hyafil and Bacchus (2004). We use instances with configurations x-y-z = 2-2-2, 4-2-2, and 2-2-4, and distinguish between two levels of uncertainty.

- L-x-y-z correspond to problems with uncertainty only in the outcome of the $load$ and $unload$ actions. Specifically, the probabilities of success for $load$ are 0.875 for trucks and 0.9 for airplanes, and for $unload$, 0.75 and 0.8, respectively.

- LL-x-y-z extends L-x-y-z with independent uniform priors for each initial location of a package within its start city.

Figure 19 depicts (on a log scale) runtimes of Probabilistic-FF and POND on L-2-2-2, L-4-2-2, and L-2-2-4, as a function of growing θ. On these problems, both planners appear to scale well, with the runtime of Probabilistic-FF and the optimal runtime of POND being roughly the same, and the average runtime of POND somewhat degrading from 2-2-2 to 4-2-2 to 2-2-4. This shows that both planners are much more efficient in this domain than the previously known SAT and CSP based techniques. However, moving to LL-x-y-z changes the picture for both planners. The results are as follows:

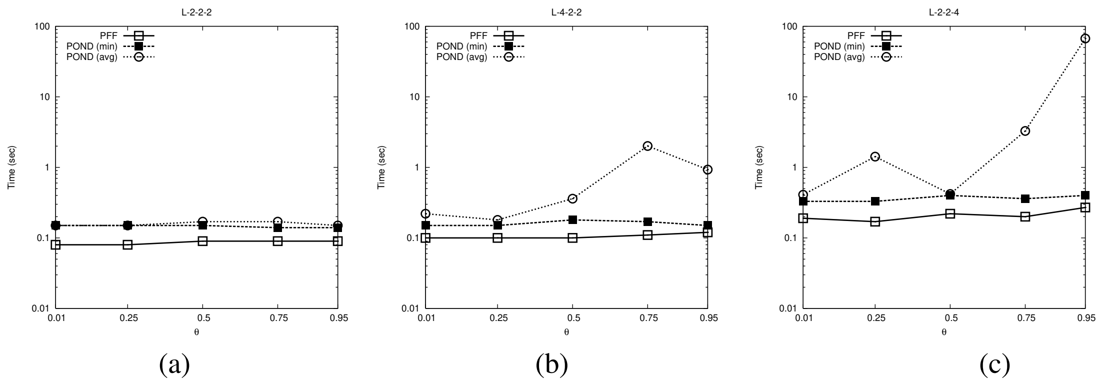


Figure 19: Probabilistic-FF and POND on problems from Logistics (a) L-2-2-2, (b) L-4-2-2, and (c) L-2-2-4.

1. On LL-2-2-2, the runtimes of Probabilistic-FF were identical to those on L-2-2-2, and the optimal runtimes of POND only slightly degraded to $2-8$ seconds. However, for all examined values of θ, some runs of POND resulted in timeouts.

2. On LL-4-2-2, the runtimes of Probabilistic-FF were identical to those on L-4-2-2 for $\theta \in \{0.01, 0.25, 0.5, 0.75\}$, yet Probabilistic-FF time-outed on $\theta = 0.95$. The optimal runtimes of POND degraded from those for L-4-2-2 only to $9 - 18$ seconds, and again, for all values of θ, some runs of POND resulted in timeouts.

3. On LL-2-2-4, Probabilistic-FF experienced hard times, finishing in 0.19 seconds for $\theta = 0.01$, and time-outing for all other examined values of θ. The optimal runtimes of POND degraded from those for L-2-2-4 to $120 - 700$ seconds, and here as well, for all values of θ, some runs of POND resulted in timeouts.

We also tried a variant of LL-x-y-z with non-uniform priors over the initial locations of the packages, but this resulted in a qualitatively similar picture of absolute and relative performance.

The LL-x-y-z domain remains challenging, and deserves close attention in the future developments for probabilistic planning. In this context, it is interesting to have a close look at what the reasons for the failure of Probabilistic-FF is. It turns out that Probabilistic-FF is not good enough at estimating how many times, at an early point in the plan, a probabilistic action must be applied in order to sufficiently support a high goal threshold at the end of the plan. To make this concrete, consider a Logistics example with uncertain effects of all load and unload actions. Consider a package P that must go from a city A to a city B. Let's say that P is initially not at A's airport. If the goal threshold is high, this means that, to be able to succeed, the package has to be brought to A's airport with a high probability *before* loading it onto an airplane. This is exactly the point where Probabilistic-FF's heuristic function fails. The relaxed plan contains too few actions unloading P at A's airport. The effect is that the search proceeds too quickly to loading P onto a plane and bringing it to B. Once the search gets to the point where B should be unloaded to its goal location, the goal threshold cannot be achieved no matter how many times one unloads P. At this point, Probabilistic-FF's enforced hill-climbing enters a loop and eventually fails because the relaxed plan (which over-estimates the past achievements) becomes empty.[18]

The challenge here is to devise methods that are better at recognizing how many times P has to be unloaded at A's airport in order to sufficiently support the goal threshold. The error made by Probabilistic-FF lies in that our propagation of weights on the implication graph over-estimates the goal probability. Note here that this is much more critical for actions that must be applied early on in the plan, than for actions that are applied later. If an action a appears early on in a plan, then the relaxed plan, when a is executed, will be long. Recall that the weight propagation proceeds backwards, from the goal towards the current state. At each single backwards step, the propagation makes an approximation that might lose precision of the results. Over several backwards steps, these imprecisions accumulate. Hence the quality of the approximation decreases quickly over the number of backwards steps. The longer the distance between goal and current state is, the more information is lost. We have observed this phenomenon in detailed experiments with different weight propagation schemes, that is, with different underlying assumptions. Of the propagation schemes we tried, the independence assumption, as presented in this paper, was by far the most accurate one. All other schemes failed to deliver good results even for much shorter distances between the goal and the current state.

It is interesting to consider how this issue affects POND, which uses a very different method for estimating the probability of goal achievement: instead of performing a backwards propagation and aggregation of weight values, POND sends a set of random particles through the relaxed planning graph in a forward fashion, and stops the graph building if enough particles end up in the goal. From our empirical results, it seems that this method suffers from similar difficulties as Probabilistic-FF, but not to such a large extent. POND's optimal runtimes for LL-x-y-z are much higher than those for L-x-y-z. This indicates that it is always challenging for POND to "recognize" the need for applying some action a many times early on in the plan. More interestingly, POND never times-out in L-x-y-z, but it does often time-out in LL-x-y-z. This indicates that, to some extent, it is a matter of chance whether or not POND's random particles recognize the need for applying an action a many times early on in the plan. An intuitive explanation is that the "good cases" are those where sufficiently many of the particles failed to reach the goal due to taking the "wrong effect" of a. Based on this intuition, one would expect that it helps to increase the number of random particles in POND's heuristic function. We did so, running POND on LL-x-y-z with an increased number of particles, 200 and 600 instead of the default value of 64. To our surprise, the qualitative behavior of POND did not change, time-outing in a similar number of cases. It is unclear to us what the reason for this phenomenon is. Certainly, it can be observed that the situation encoded in LL-x-y-z is not solved to satisfaction by either of Probabilistic-FF's weight propagation or POND's random particle methods, in their current configurations.

At the time of writing, it is unclear to the authors how better methods could be devised. It seems unlikely that a weight propagation – at least one that does not resort to expensive reasoning – exists which manages long distances better than the independence assumption. An alternative way out might be to simply define a weaker notion of plans that allows to repeat certain kinds of actions –

18. This does not happen in the above L-2-2-2, L-4-2-2, and L-2-2-4 instances simply because they are too small and a high goal probability can be achieved without thinking too much about the above problem; if one increases the size of these instances, the problem appears. The problem appears earlier in the presence of initial state uncertainty – even in small instances such as LL-2-2-2, LL-4-2-2, and LL-2-2-4 – because with uncertainty about the start position of the packages one needs to try unloading them at the start airports more often.

throwing a dice or unloading a package – arbitrarily many times. However, since our assumption is that we do not have any observability during plan execution, when executing such a plan there would still arise the question how often an action should be tried. Since Logistics is a fairly well-solved domain in simpler formalisms – by virtue of Probabilistic-FF, even in the probabilistic setting as long as the effects are deterministic – we consider addressing this problem as a quite pressing open question.

6. Conclusion

We developed a probabilistic extension of Conformant-FF's search space representation, using a synergetic combination of Conformant-FF's SAT-based techniques with recent techniques for weighted model counting. We further provided an extension of conformant relaxed planning with approximate probabilistic reasoning. The resulting planner scales well on a range of benchmark domains. In particular it outperforms its only close relative, POND, by at least an order of magnitude in almost all of the cases we tried.

While this point may be somewhat obvious, we would like to emphasize that our achievements do *not* solve the (this particular) problem once and for all. Probabilistic-FF inherits strengths *and* weaknesses from FF and Conformant-FF, like domains where FF's or Conformant-FF's heuristic functions yield bad estimates (e.g. the mentioned Cube-center variant). What's more, the probabilistic setting introduces several new potential impediments for FF's performance. For one thing, weighted model counting is inherently harder than SAT testing. Though this did not happen in our set of benchmarks, there are bound to be cases where the cost for exact model counting becomes prohibitive even in small examples. A promising way to address this issue lies in recent methods for *approximate* model counting (Gomes, Sabharwal, & Selman, 2006; Gomes, Hoffmann, Sabharwal, & Selman, 2007). Such methods are much more efficient than exact model counters. They provide high-confidence lower bounds on the number of models. The lower bounds can be used in Probabilistic-FF in place of the exact counts. It has been shown that good lower bounds with very high confidecne can be achieved quickly. The challenge here is to extend the methods – which are currently designed for non-weighted CNFs – to handle *weighted* model counting.

More importantly perhaps, in the presence of probabilistic effects there is a fundamental weakness in Probabilistic-FF's – and POND's – heuristic information. This becomes a pitfall for performance even in a straightforward adaptation of the Logistics domain, which is otherwise very easy for this kind of planners. As outlined, the key problem is that, to obtain a high enough confidence of goal achievement, one may have to apply particular actions several times early on in the plan. Neither Probabilistic-FF's nor POND's heuristics are good enough at identifying *how many* times. In our view, finding techniques that address this issue is currently the most important open topic in this area.

Apart from addressing the latter challenge, we intend to work towards applicability in real-word settings. Particularly, we will look at the space application settings that our Rovers domain hints at, at medication-type treatment planning domains, and at the power supply restoration domain (Bertoli, Cimatti, Slaney, & Thiébaux, 2002).

Acknowledgments

The authors would like to thank Dan Bryce and Rao Kambhampati for providing a binary distribution of POND2.1. Carmel Domshlak was partially supported by the Israel Science Foundations grant 2008100, as well as by the C. Wellner Research Fund. Some major parts of this research have been accomplished at the time that Jörg Hoffmann was employed at the Intelligent Information Systems Institute, Cornell University.

Appendix A. Proofs

Proposition 2 *Let $(A, \mathcal{N}_{b_I}, G, \theta)$ be a probabilistic planning problem described over k state variables, and $\overline{a}$ be an m-step sequence of actions from A. Then, we have $|\mathcal{N}_{b_{\overline{a}}}| = O(|\mathcal{N}_{b_I}|+m\alpha(k+1))$ where α is the largest description size of an action in A.*

Proof: The proof is rather straightforward, and it exploits the local structure of $\mathcal{N}_{b_{\overline{a}}}$'s CPTs. The first nodes/CPTs layer $\mathcal{X}_{(0)}$ of $\mathcal{N}_{b_{\overline{a}}}$ constitutes an exact copy of $\mathcal{N}_{b_I}$. Then, for each $1 \leq t \leq m$, the t-th layer of $\mathcal{N}_{b_{\overline{a}}}$ contains $k+1$ node $\{Y_{(t)}\} \cup \mathcal{X}_{(t)}$.

First, let us consider the "action node" $Y_{(t)}$. While specifying the CPT $T_{Y(t)}$ in a straightforward manner as if prescribed by Eq. 4 might result in an exponential blow up, the same Eq. 4 suggests that the original description of a^t is by itself a compact specification of $T_{Y(t)}$. Therefore, $T_{Y(t)}$ can be described in space $O(\alpha)$, and this description can be efficiently used for answering queries $T_{Y(t)}(Y_{(i)} = \varepsilon \mid \pi)$ as in Eq. 4. Next, consider the CPT $T_{X(t)}$ of a state-variable node $X(t) \in \mathcal{X}_{(t)}$. This time, it is rather evident from Eq. 5 that $T_{X(t)}$ can be described in space $O(\alpha)$ so that queries $T_{X(t)}(X_{(t)} = x \mid X_{(t-1)} = x')$ could be efficiently answered. Thus, summing up for all layers $1 \leq t \leq m$, the description size of $|\mathcal{N}_{b_{\overline{a}}}| = O(|\mathcal{N}_{b_I}| + m\alpha(k+1))$ ■

Lemma 4 *Given a node $v(t') \in$ Imp$_{\rightarrow p(t)}$, we have $\varpi_{p(t)}(v(t')) = \varpi(v(t'))$ if and only if, given v at time t', the sequence of effects $E($Imp$_{v(t') \rightarrow p(t)})$ achieves p at t with probability 1.*

Proof: The proof of Lemma 4 is by a backward induction on the time layers of *Imp*$_{v(t') \rightarrow p(t)}$. For time t, the only node of *Imp*$_{\rightarrow p(t)}$ time-stamped with t is $p(t)$ itself. For this node we do have $\varpi_{p(t)}(p(t)) = \varpi(p(t)) = 1$, but, given p at time t, an empty plan corresponding to (empty) $E($*Imp*$_{p(t) \rightarrow p(t)})$ trivially "re-establishes" p at t with certainty. Assuming now that the claim holds for all nodes of *Imp*$_{\rightarrow p(t)}$ time stamped with $t'+1, \ldots, t$, we now show that it holds for the nodes time stamped with t'.

It is easy to see that, for any node $v(t') \in$ *Imp*$_{\rightarrow p(t)}$, we get $\varpi_{p(t)}(v(t')) = \varpi(v(t'))$ only if α goes down to zero. First, consider the chance nodes $\varepsilon(t') \in$ *Imp*$_{v \rightarrow p(t)}$. For such a node, lb is set to zero if and only if we have $\varpi_{p(t)}(r(t'+1)) = 1$ for some $r \in add(\varepsilon)$. However, by our inductive assumption, in this and only in this case the effects $E($*Imp*$_{\varepsilon(t') \rightarrow p(t+1)})$ achieve p at t with probability 1, given the occurrence of ε at time t'.

Now, consider the fact nodes $q(t') \in$ *Imp*$_{v \rightarrow p(t)}$. For such a node, α can get nullified only by some effect $e \in E(a), a \in A(t'), con(e) = q$. The latter happens if only if, for *all* possible outcomes of e, (i) the node $\varepsilon(t')$ belongs to *Imp*$_{\rightarrow p(t)}$, and (ii) and the estimate $\varpi_{p(t)}(\varepsilon(t')) = \varpi(\varepsilon(t'))$. In other words, by our inductive assumption, given *any* outcome $\varepsilon \in \Lambda(e)$ at time t', the effects $E($*Imp*$_{\varepsilon(t') \rightarrow p(t)})$ achieve p at t with probability 1. Thus, given q at time t', the effects $E($*Imp*$_{q(t') \rightarrow p(t)})$ achieve p at t with probability 1 *independently* of the actual outcome of e. Alternatively, if for $q(t')$ we have $lb > 0$, then for each effect e conditioned on $q(t)$, there exists an outcome ε of e such that, according to what we just proved for the chance nodes time-stamped with t', the effects $E(Imp_{\varepsilon(t') \to p(t+1)})$ do not achieve p at t with probability 1. Hence, the whole set of effects $E(Imp_{q(t') \to p(t+1)})$ does not achieve p at t with probability 1. ■

Lemma 5 *Let $(A, \mathcal{N}_{b_I}, G, \theta)$ be a probabilistic planning task, $\overline{a}$ be a sequence of actions applicable in b_I, and $|_1^+$ be a relaxation function for A. For each time step $t \geq -m$, and each proposition $p \in \mathcal{P}$, if $P(t)$ is constructed by* build-PRPG$(\overline{a}, A, \phi(\mathcal{N}_{b_I}), G, \theta, |_1^+)$*, then p at time t can be achieved by a relaxed plan starting with $\overline{a}|_1^+$*

(1) with probability > 0 (that is, p is not negatively known at time t) if and only if $p \in uP(t) \cup P(t)$, and

(2) with probability 1 (that is, p is known at time t) if and only if $p \in P(t)$.

Proof: The proof of the "if" direction is by a straightforward induction on t. For $t = -m$ the claim is immediate by the direct initialization of $uP(-m)$ and $P(-m)$. Assume that, for $-m \leq t' < t$, if $p \in uP(t') \cup P(t')$, then p is not negatively known at time t', and if $p \in P(t')$, then p is known at time t'.

First, consider some $p(t) \in uP(t) \cup P(t)$, and suppose that p is egatively know at time t. By the inductive assumption, and the property of the PRPG construction that $uP(t-1) \cup P(t-1) \subseteq uP(t) \cup P(t)$, we have $p \notin uP(t-1) \cup P(t-1)$. Therefore, p has to be added into $uP(t)$ (and then, possibly, moved from there to $P(t)$) in the first **for** loop of the build-timestep procedure. However, if so, then there exists an action $a \in A(t-1)$, $e \in E(a)$, and $\varepsilon \in \Lambda(e)$ such that (i) $con(e) \in uP(t-1) \cup P(t-1)$, and (ii) $p \in add(\varepsilon)$. Again, by the assumption of the induction we have that $pre(a)$ is known at time $t-1$, and $con(e)$ is not negatively known at time $t-1$. Hence, the non-zero probability of ε occurring at time t implies that p can be achieved at time t with probability greater than 0, contradicting that p is negatively know at time t.

Now, let us consider some $p(t) \in P(t)$. Notice that, for $t > -m$, we have $p(t) \in P(t)$ if and only if

$$\Phi \longrightarrow \bigvee_{l \in support(p(t))} l \quad . \tag{28}$$

Thus, for each world state w consistent with b_I, we have either $q \in w$ for some fact proposition $q(-m) \in support(p(t))$, or, for some effect e of an action $a(t') \in A(t')$, $t' < t$, we have $con(e) \in P(t')$ and $\{\varepsilon(t') \mid \varepsilon \in \Lambda(e)\} \subseteq support(p(t))$. In this first case, Lemma 4 immediately implies that the concatenation of $\overline{a}|_1^+$ with an arbitrary linearization of the (relaxed) actions $A(0), \ldots, A(t-1)$ achieves p at t with probability 1, and thus p is known at time t. In the second case, our inductive assumption implies that $con(e)$ is known at time t, and together with Lemma 4 this again implies that the concatenation of $\overline{a}|_1^+$ with an arbitrary linearization of the (relaxed) actions $A(0), \ldots, A(t-1)$ achieves p at t with probability 1.

The proof of the "only if" direction is by induction on t as well. For $t = -m$ this claim is again immediate by the direct initialization of $P(-m)$. Assume that, for $-m \leq t' < t$, if p is not negatively known at time t', then $p \in uP(t') \cup P(t')$, and if p is known at time t', then $p \in P(t')$. First, suppose that p is not negatively known at time t, and yet we have $p \notin uP(t) \cup P(t)$. From our inductive assumption plus that $A(t-1)$ containing all the NOOP actions for propositions in $uP(t-1) \cup P(t-1)$, we know that p *is* negatively known at time $t-1$. If so, then p can become not negatively known at time t only due to some $\varepsilon \in \Lambda(e)$, $e \in E(a)$, such that $pre(a)$ is known at time $t-1$, and $con(e)$ is not negatively known at time $t-1$. By our inductive assumption, the latter conditions imply $con(e) \in uP(t-1) \cup P(t-1)$, and $pre(a) \in P(t-1)$. But if so, then p has to be added to $uP(t) \cup P(t)$ by the first **for** loop of the build-timestep procedure, contradicting our assumption that $p \notin uP(t) \cup P(t)$.

Now, let us consider some p known at time t. By our inductive assumption, $P(t-1)$ contains all the facts known at time $t-1$, and thus $A(t-1)$ is the maximal subset of actions $A|_1^+$ applicable at time $t-1$. Let us begin with an exhaustive classification of the effects e of the actions $A(t-1)$ with respect to our p at time t.

(I) $\forall \varepsilon \in \Lambda(e) : p \in add(\varepsilon)$, and $con(e) \in P(t-1)$

(II) $\forall \varepsilon \in \Lambda(e) : p \in add(\varepsilon)$, and $con(e) \in uP(t-1)$

(III) $\exists \varepsilon \in \Lambda(e) : p \notin add(\varepsilon)$ or $con(e) \notin P(t-1) \cup uP(t-1)$

If the set (I) is not empty, then, by the construction of build-w-impleafs($p(t)$, *Imp*), we have

$$\{\varepsilon(t-1) \mid \varepsilon \in \Lambda(e)\} \subseteq \textit{support}(p(t)),$$

for each $e \in$ (I). Likewise, by the construction of build-timestep (notably, by the update of Φ), for each $e \in$ (I), we have

$$\Phi \longrightarrow \bigvee_{\{\varepsilon(t-1) \mid \varepsilon \in \Lambda(e)\}} \varepsilon(t-1).$$

Putting these two facts together, we have that Eq. 28 holds for p at time t, and thus we have $p \in P(t)$.

Now, suppose that the set (I) is empty. It is not hard to verify that no subset of *only* effects (III) makes p known at time t. Thus, the event "at least one of the effects (II) occurs" must hold with probability 1. First, by the construction of build-w-impleafs($p(t)$, *Imp*), we have

$$\textit{support}\,(p(t)) \supseteq \bigcup_{e \in \text{(II)}} \textit{support}\,(con(e)(t-1))$$

Then, and 4 from Lemma 4 we have that the event "at least one of the effects (II) occurs" holds with probability 1 if and only if

$$\Phi \longrightarrow \bigvee_{\substack{e \in \text{(II)} \\ l \in \textit{support}(con(e)(t-1))}} l$$

Putting these two facts together, we have that Eq. 28 holds for p at time t, and thus we have $p \in P(t)$. ■

Theorem 6 *Let $(A, \mathcal{N}_{b_I}, G, \theta)$ be a probabilistic planning task, $\overline{a}$ be a sequence of actions applicable in b_I, and $|_1^+$ be a relaxation function for A. If* build-PRPG($\overline{a}, A, \phi(\mathcal{N}_{b_I}), G, \theta, |_1^+$) *returns FALSE, then there is no relaxed plan for (A, b_I, G, θ) that starts with $\overline{a}|_1^+$.*

Proof: Let $t > 0$ be the last layer of the PRPG upon the termination of build-PRPG. For every $-m \leq t' \leq t$, by the construction of PRPG and Lemma 5, the sets $P(t')$ and $uP(t')$ contain all (and only all) propositions that are known (respectively unknown) after executing all the actions in the action layers up to and including $A(t'-1)$.

First, let us show that if build-PRPG returns FALSE, then the corresponding termination criterion would hold in all future iterations. If $P(t+1) = P(t)$, then we have $A(t+1) = A(t)$. Subsequently, since $P(t+1) \cup uP(t+1) = P(t) \cup uP(t)$ and $A(t+1) = A(t)$, we have $P(t+2) \cup uP(t+2) = P(t+1) \cup uP(t+1)$. Given that, we now show that $P(t+2) = P(t+1)$ and $uP(t+2) = uP(t+1)$.

Assume to the contrary that there exists $p(t+2) \in P(t+2)$ such that $p(t+1) \notin P(t+1)$, that is $p(t+1) \in uP(t+1)$. By the construction of the sets $P(t+1)$ and $P(t+2)$ in the build-timestep procedure, we have

$$\begin{aligned} \Phi &\longrightarrow \bigvee_{l \in \mathit{support}(p(t+2))} l \ , \\ \Phi &\not\longrightarrow \bigvee_{l \in \mathit{support}(p(t+1))} l \end{aligned} \tag{29}$$

Consider an exhaustive classification of the effects e of the actions $A(t+1)$ with respect to our p at time $t+2$.

(I) $\forall \varepsilon \in \Lambda(e) : p \in add(\varepsilon)$, and $con(e) \in P(t+1)$

(II) $\forall \varepsilon \in \Lambda(e) : p \in add(\varepsilon)$, and $con(e) \in uP(t+1)$

(III) $\exists \varepsilon \in \Lambda(e) : p \notin add(\varepsilon)$ or $con(e) \notin P(t+1) \cup uP(t+1)$

Suppose that the set (I) is not empty, and let $e \in$ (I). From $P(t) = P(t+1)$ we have that $con(e) \in P(t)$, and thus $\{\varepsilon(t) \mid \varepsilon \in \Lambda(e)\} \subseteq \mathit{support}(p(t+1))$. By the update of Φ in build-timestep we then have $\Phi \longrightarrow \bigvee_{\{\varepsilon(t) \mid \varepsilon \in \Lambda(e)\}} \varepsilon(t)$, and thus $\Phi \longrightarrow \bigvee_{l \in \mathit{support}(p(t+1))} l$, contradicting Eq. 29.

Alternatively, assume that the set (I) is empty. Using the arguments similar to these in the proof of Lemma 5, $p(t+2) \in P(t+2)$ and $p(t+1) \notin P(t+1)$ in this case imply that

$$\begin{aligned} \Phi &\longrightarrow \bigvee_{\substack{e \in \text{(II)} \\ l \in \mathit{support}(con(e)(t+1))}} l \\ \Phi &\not\longrightarrow \bigvee_{\substack{e \in \text{(II)} \\ l \in \mathit{support}(con(e)(t))}} l \end{aligned} \tag{30}$$

However, $A(t+1) = A(t)$, $uP(t+1) = uP(t)$, and $P(t+1) = P(t)$ together imply that all the action effects that can possibly take place at time $t+1$ are also feasible to take place at time t. Therefore, since for each $e \in$ (II) we have $con(e) \in uP(t+1)$ by the definition of (II), Eq. 30 implies that

$$\bigcup_{e \in \text{(II)}} \mathit{support}\,(con(e)(t+1)) \cap uP(-m) \neq \bigcup_{e \in \text{(II)}} \mathit{support}\,(con(e)(t)) \cap uP(-m), \tag{31}$$

contradicting our termination condition. Hence, we arrived into contradiction with our assumption that $p(t+1) \notin P(t+1)$.

Having shown that $P(t+2) = P(t+1)$ and $uP(t+2) = uP(t+1)$, we now show that the termination criteria implies that, for each $q(t+2) \in uP(t+2)$, we have

$$uP(-m) \cap \mathit{support}(p(t+2)) = uP(-m) \cap \mathit{support}(p(t+1)).$$

Let $E_{p(t+2)}$ be the set of all effects of actions $A(t+1)$ such that $con(e) \in uP(t+1)$, and, for each outcome $\varepsilon \in \Lambda(e)$, we have $p \in add(\varepsilon)$. Given that, we have

$$\begin{aligned} uP(-m) \cap \mathit{support}(p(t+2)) &= uP(-m) \cap \bigcup_{e \in E_{p(t+2)}} \mathit{support}(con(e)(t+1)) \\ &= uP(-m) \cap \bigcup_{e \in E_{p(t+2)}} \mathit{support}(con(e)(t)) \qquad , \\ &= uP(-m) \cap \mathit{support}(p(t+1)) \end{aligned} \tag{32}$$

where the first and third equalities are by the definition of *support* sets via Lemma 4, and the second equation is by our termination condition.

The last things that remains to be shown is that our termination criteria implies get-P$(t+2, G)$ =get-P$(t+1, G)$. Considering the simple cases first, if $G \not\subseteq P(t+1) \cup uP(t+1)$, from $P(t+2) \cup uP(t+2) = P(t+1) \cup uP(t+1)$ we have get-P$(t+2, G)$ =get-P$(t+1, G) = 0$. Otherwise, if $G \subseteq P(t+1)$, from $P(t+2) = P(t+1)$ we have get-P$(t+2, G)$ =get-P$(t+1, G) = 1$.

This leaves us with the case of $G \subseteq P(t+1) \cup uP(t+1)$ and $G \cap uP(t+1) \neq \emptyset$. From $P(t+2) = P(t+1)$, $uP(t+2) = uP(t+1)$, and the termination condition, we have

$$G \cap uP(t) = G \cap uP(t+1) = G \cap uP(t+2).$$

From get-P$(t+1, G)$ =get-P(t, G) we know that action effects that become feasible only in $A(t)$ do not increase our estimate of probability of achieving any $g \in G \cap uP(t+1)$ from time t to time $t+1$. However, from $P(t+1) = P(t)$, $uP(t+1) = uP(t)$, and $A(t+1) = A(t)$, we have that no action effect will become feasible at time $t+1$ if it is not already feasible at time t, and thus get-P$(t+1, G)$ =get-P(t, G) will imply get-P$(t+2, G)$ =get-P$(t+1, G)$.

To this point we have shown that if build-PRPG returns FALSE, then the corresponding termination criterion would hold in all future iterations. Now, assume to the contrary to the claim of the theorem that build-PRPG returns FALSE at some iteration t, yet there exists a relaxed plan for (A, b_I, G, θ) that starts with $\overline{a}|_1^+$. First, if $\theta = 1$, then Lemma 5 implies that there exists time T such that $G \subseteq P(T)$. If so, then the persistence of our "negative" termination condition implies $G \subseteq P(t)$. However, in this case we would have get-P$(t, G) = 1$ (see the second **if** of the get-P procedure), and thus build-PRPG would return TRUE before ever getting to check the "negative" termination condition in iteration t. Alternatively, if $\theta = 0$, then build-PRPG would have terminated with returning TRUE before the "negative" termination condition is checked even once.

This leaves us with the case of $0 < \theta < 1$ and get-P$(t, G) < \theta$. (get-P$(t, G) \geq \theta$ will again contradict reaching the negative termination condition at iteration t.) We can also assume that $G \subseteq P(t) \cup uP(t)$ because $P(t) \cup uP(t)$ contains all the facts that are not negatively known at time t, and thus persistence of the negative termination condition together with $G \not\subseteq P(t) \cup uP(t)$ would imply that there is no relaxed plan for any $\theta > 0$. Let us consider the sub-goals $G \cap uP(t) \neq \emptyset$.

(1) If for all subgoals $g \in G \cap uP(t)$, the implications in $Imp_{\rightarrow g(t)}$ are *only* due to deterministic outcomes of the effects $E(Imp_{\rightarrow g(t)})$, then the uncertainty about achieving $G \cap uP(t)$ at time t is *only* due to the uncertainty about the initial state. Since the initial belief state is reasoned about with no relaxation, in this case get-P$(t, G) = \mathsf{WMC}(\Phi \wedge \bigwedge_{g \in G \setminus P(t)} \varphi_g)$ provides us with an *upper bound* on the probability of achieving our goal G by $\overline{a}|_1^+$ concatenated with

an arbitrary linearization of an arbitrary subset of $A(0), \ldots, A(t-1)$. The termination sub-condition get-P$(t+1, G)$ =get-P(t, G) and the persistence of the action sets $A(T)$, $T \geq t$, imply then that get-P(t, G) provides us with an upper bound on the probability of achieving G by $\overline{a}|_1^+$ concatenated with an arbitrary linearization of an arbitrary subset of $A(0), \ldots, A(T)$, for all $T \geq t$. Together with get-P$(t, G) < \theta$, the latter conclusion contradicts our assumption that a desired relaxed plan exists.

(2) If there exists a subgoal $g \in G \cap uP(t)$ such that some implications in $Imp_{\rightarrow g(t)}$ are due to truly probabilistic outcomes of the effects $E(Imp_{\rightarrow g(t)})$, then repeating the (relaxed) actions $A(t)$ in $A(t+1)$ will *necessarily* result in $\mathsf{WMC}(\Phi \wedge \bigwedge_{g \in G \setminus P(t+1)} \varphi_g) > \mathsf{WMC}(\Phi \wedge \bigwedge_{g \in G \setminus P(t)} \varphi_g)$, contradicting our termination sub-condition condition get-P$(t+1, G)$ =get-P(t, G).

Hence, we arrived into contradiction that our assumption that build-PRPG returns FALSE at time t, yet there exists a relaxed plan for (A, b_I, G, θ) that starts with $\overline{a}|_1^+$. ■

References

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

Bertoli, P., Cimatti, A., Slaney, J., & Thiébaux, S. (2002). Solving power supply restoration problems with planning via symbolic model-checking. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)*, pp. 576–580, Lion, France.

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1-2), 279–298.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1–2), 5–33.

Bonet, B., & Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS)*, pp. 52–61, Breckenridge, CO.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 115–123, Portland, OR.

Brafman, R. I., & Domshlak, C. (2006). Factored planning: How, when, and when not. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pp. 809–814, Boston, MA.

Bryce, D., & Kambhampati, S. (2004). Heuristic guidance measures for conformant planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 365–374, Whistler, BC, Canada.

Bryce, D., Kambhampati, S., & Smith, D. (2006). Sequential Monte Carlo in probabilistic planning reachability heuristics. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 233–242, Cumbria, UK.

Chavira, M., & Darwiche, A. (2005). Compiling Bayesian networks with local structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1306–1312, Edinburgh, Scotland.

Darwiche, A. (2000). Recursive conditioning. *Artificial Intelligence*, *125*(1-2), 5–41.

Darwiche, A. (2001). Constant-space reasoning in dynamic Bayesian networks. *International Journal of Approximate Reasoning*, *26*(3), 161–178.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*, 142–150.

Dechter, R. (1999). Bucket elimination: A unified framework for reasoning. *Artificial Intelligence*, *113*, 41–85.

Domshlak, C., & Hoffmann, J. (2006). Fast probabilistic planning through weighted model counting. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 243–252, Cumbria, UK.

Gomes, C. P., Hoffmann, J., Sabharwal, A., & Selman, B. (2007). From sampling to model counting. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India.

Gomes, C. P., Sabharwal, A., & Selman, B. (2006). Model counting: A new strategy for obtaining good bounds. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-06)*, pp. 54–61, Boston, MA.

Hanks, S., & McDermott, D. (1994). Modeling a dynamic and uncertain world I: Symbolic and probabilistic reasoning about change. *Artificial Intelligence*, *66*(1), 1–55.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Hoffmann, J., & Brafman, R. (2006). Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, *170*(6–7), 507–541.

Huang, J. (2006). Combining knowledge compilation and search for efficient conformant probabilistic planning. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 253–262, Cumbria, UK.

Hyafil, N., & Bacchus, F. (2004). Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 1033–1034, Valencia, Spain.

Jensen, F. (1996). *An Introduction to Bayesian Networks*. Springer Verlag, New York.

Kushmerick, N., Hanks, S., & Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, *78*(1-2), 239–286.

Little, I., Aberdeen, D., & Thiébaux, S. (2005). Prottle: A probabilistic temporal planner. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pp. 1181–1186, Pittsburgh, PA.

Littman, M. L., Goldsmith, J., & Mundhenk, M. (1998). The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, *9*, 1–36.

Majercik, S. M., & Littman, M. L. (1998). MAXPLAN: A new approach to probabilistic planning. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 86–93, Pittsburgh, PA.

Majercik, S. M., & Littman, M. L. (2003). Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, *147*(1-2), 119–162.

McDermott, D. (1998). The 1998 AI Planning Systems Competition. *AI Magazine*, *2*(2), 35–55.

McDermott, D. V. (1999). Using regression-match graphs to control search in planning. *Artificial Intelligence*, *109*(1-2), 111–159.

Onder, N., Whelan, G. C., & Li, L. (2006). Engineering a conformant probabilistic planner. *Journal of Artificial Intelligence Research*, *25*, 1–15.

Pearl, J. (1984). *Heuristics - Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA.

Rintanen, J. (2003). Expressive equivalence of formalisms for planning with sensing. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 185–194, Trento, Italy.

Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence*, *82*(1-2), 273–302.

Russell, S., & Norvig, P. (2004). *Artificial Intelligence: A Modern Approach* (2 edition). Pearson.

Sang, T., Bacchus, F., Beame, P., Kautz, H., & Pitassi, T. (2004). Combining component caching and clause learning for effective model counting. In *(Online) Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, Vancouver, BC, Canada.

Sang, T., Beame, P., & Kautz, H. (2005). Solving Bayes networks by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pp. 475–482, Pittsburgh, PA.

Shimony, S. E. (1993). The role of relevance in explanation I: Irrelevance as statistical independence. *International Journal of Approximate Reasoning*, *8*(4), 281–324.

Shimony, S. E. (1995). The role of relevance in explanation II: Disjunctive assignments and approximate independence. *International Journal of Approximate Reasoning*, *13*(1), 27–60.

Zhang, N. L., & Poole, D. (1994). A simple approach to Bayesian network computations. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, pp. 171–178, Banff, Alberta, Canada.