Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

A Compilation Based Approach to Conformant Probabilistic Planning with Stochastic Actions

Ran Taig and **Ronen I. Brafman**
Computer Science Department
Ben Gurion University of The Negev
Beer-Sheva, Israel 84105
taig,brafman@cs.bgu.ac.il

Abstract

We extend RBPP, the state-of-the-art, translation-based planner for conformant probabilistic planning (CPP) with deterministic actions, to handle a wide set of CPPs with stochastic actions. Our planner uses relevance analysis to divide a probabilistic "failure-allowance" between the initial state and the stochastic actions. Using its "initial-state allowance," it uses relevance analysis to select a subset of the set of initial states on which planning efforts will focus. Then, it generates a deterministic planning problem using all-outcome determinization in which action cost reflects the probability of the modeled outcome. Finally, a cost-bounded classical planner generates a plan with failure probability lower than the "stochastic-effect allowance." Our compilation method is sound, but incomplete, as it may underestimates the success probability of a plan. Yet, it scales up much better than the state-of-the-art PFF planner, solving larger problems and handling tighter probabilistic bounds on existing benchmarks.

Introduction

In conformant probabilistic planning (CPP) we are given a distribution over initial states, a set of actions, a goal condition, and a real value $0 < \theta \leq 1$. A valid plan is one that achieves the goal with probability $\geq \theta$. Few natural problems fit the framework, yet important ideas developed in CP were later extended to the richer framework of contingent planning, including techniques for representing and reasoning about belief states (Hoffmann and Brafman 2005) and compilation schemes (Albore, Palacios, and Geffner 2009).

Compilation methods have been very effective on CPP with deterministic actions (Taig and Brafman 2014), but no compilation-based planner supports stochastic actions due to two fundamental problems. First, these methods have difficulty handling non-deterministic actions, in general. Second, in CPP, one must monitor two types of probabilistic uncertainty within the translation: uncertainty regarding the initial state and uncertainty regarding action effects. Consequently, few CPP solvers handle stochastic actions, and even those that do, such as *Probabilistic-FF* (Domshlak and Hoffmann 2007) and *POND* (Bryce, Kambhampati, and Smith 2006) have difficulty handling larger problems and problems in which θ is high, especially when the plan requires repeating an action a number of times in order to increase its success probability.

In this paper we present a new compilation-based CPP planner, *RBPP+*, which extends our *RBPP* planner (Taig and Brafman 2014) with support for stochastic actions. *RBPP+* scales up to much larger problem instances than those handled so far, and can handle tighter probabilistic bounds. *RBPP+* integrates techniques developed in earlier compilation-based methods. First, it performs a dedicated relevance-based analysis, building on ideas developed in *RBPP*. Based on this analysis, the $1-\theta$ "failure-allowance" is divided into $1-\theta_I$ and $1-\theta_E$, representing failure due to the uncertainty about the initial state and due to the uncertainty regarding action effects, respectively. Next, *RESTRICT+*, a slightly modified version of *RBPP*'s *RESTRICT* procedure, selects a promising set of initial states with probability $\geq \theta_I$ to plan on, ignoring all other possible initial states. In parallel, we determinize probabilistic actions using all-outcome determinization (Yoon, Fern, and Givan 2007). In this determinization, each possible probabilistic outcome of the original action is represented by a separate deterministic action. We associate a cost with each such action that captures the probability of the corresponding stochastic effect, following the ideas of (Jiménez et al. 2006). The resulting problem is a deterministic conformant planning problem with costs. We now seek a bounded cost plan to this problem. If the bound is suitably selected, the plan has success probability of θ_E or higher.

Overall, the resulting plan has at least one possible execution path that will succeed with probability θ. Because this approach ignores alternative successful execution path of this plan, it under-estimates the success probability and is incomplete. To partially address this problem, we augment the determinized problem using a "repeatable action" mechanism, described later. With this enhancement, we are able to handle most current benchmark domains.

Our empirical evaluation shows that this approach is effective and dominates existing state-of-the-art planners on most problem instances: it scales up to problem sizes that *PFF* cannot handle, and is able to solve problems with tighter probabilistic bounds (higher θ values). However, it also reveals the limitation of existing benchmark domains for which *RBPP+*'s reasoning on limited executions suffices.

Background

We assume familiarity with classical planning notation, with (V, A, I, G) corresponding to a set of *propositions*, *actions*, *initial world state*, and *goal*, respectively. A CP problem, (V, A, b_I, G), generalizes this framework, replacing the single initial state with a set of initially possible states, called the *initial belief state* b_I, and non-deterministic actions. A plan is an action sequence $\overline{a}$ such that $\overline{a}(w_I) \subseteq G$ for every $w_I \in b_I$. CPP extend CP by quantifying the uncertainty regarding b_I using a probability distribution b_{π_I} and associating probabilities with action outcomes.

Following (Domshlak and Hoffmann 2007), a CPP task is a 5-tuples $(V, A, b_{\pi_I}, G, \theta)$, corresponding to the *propositions set*, *action set*, *initial belief state*, *goals*, and *acceptable goal satisfaction probability*. G is a conjunction of propositions. b_{π_I} is a probability distribution over world states, where $b_{\pi_I}(w)$ is the probability that w is the true initial world state. Throughout this paper, we use b_I to denote the set of states to which b_{π_I} assigns a positive probability. The effect set $E(a)$ for an action $a \in A$ 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 $\epsilon \in \Lambda(e)$ is a triplet $(Pr(\epsilon), add(\epsilon), del(\epsilon))$, where add and delete lists are as usual, and $Pr(\epsilon)$ 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, $\sum_{\epsilon \in \Lambda(e)} Pr(\epsilon) = 1$. In the special case of deterministic effect e, we have $\Lambda(e) = \epsilon$ and $Pr(\epsilon) = 1$. Unconditional effects satisfy $con(e) = \emptyset$. If a is not applicable in a state 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 $con(e) \subseteq s$, and for each $\epsilon \in \Lambda(e)$, applying a to w results in $w \cup add(\epsilon) \setminus del(\epsilon)$ with probability $Pr(\epsilon)$. This means we assume the conditional effects of an action are mutually exclusive.

In such settings, achieving G with certainty is often impossible. CPP introduces the parameter θ, which specifies the required *lower bound* on the probability of achieving G. Thus, a sequence of actions $\overline{a}$ is a *valid plan* if the probability $\overline{a}$ reaches the goal (taking into account both initial uncertainty and stochastic effects) is at least θ. We note that some approaches to CPP require, in addition, that the plan be executable in all initial states. Our algorithm can be easily adjusted to support either semantics, but in this paper we follow a strictly probabilistic semantics which requires executability and success with probability at least θ.

A CPP specification must represent the initial distribution b_{π_I}. *PFF's* approach contains two parts. First, a definition of an induced set of multi-valued variables, corresponding to a set of literals, only one of which can be true at a time (e.g., literals denoting possible locations of an object). And second, $\mathcal{N}_{b_I}$, which is a Bayes Net (BN) defined over this set of multi-valued variables.

Related Work

Probabilistic FF (PFF) (Domshlak and Hoffmann 2007) is the best current CPP solver that handles stochastic actions. It uses a time-stamped BN to describe probabilistic belief states, extending Conformant-FF's (Hoffmann and Brafman 2006) belief state encoding to model this BN. It uses both SAT reasoning and weighted model-counting to determine whether the goal probability is at least θ. In addition, it introduces approximate probabilistic reasoning into *CFF* heuristic function. While PFF performs well on many domains, its reasoning mechanism is complicated and is sensitive to the order by which effect conditions appear. Another CPP planner which supports stochastic actions is *POND* which performs inadmissible heuristic forward search in belief space. It differs from Probabilistic-FF in the belief representation method and uses random particles for the probabilistic reasoning. Empirically, PFF dominates *POND* significantly, and both planners share a significant disadvantage: they create a complex relaxed probabilistic planning graph which limits the size of instances that can be handled. When stochastic actions exist, the graph is too complex even for medium-sized problems. Another problem both planners suffer from is their inability to detect the need to apply some action a few times early in the plan. Thus, they often generate plan prefixes that cannot be extended to a valid plan, wasting futile time extending them. Our reduction approach is much simpler. It avoids complex repeated inference on graphical models, requiring only a simple computation of the initial state probability and keeping track of action cost. The enables us to handle problems that are both larger and require higher values of θ, but at the price of incompleteness.

(Jiménez et al. 2006) suggested the use of action cost to model probability in the context of replanning with determinization, which we adapt. They used all-outcome determinization, setting the cost of determinized actions to $-log$(probability-of-failure). Thus, the sum of costs is the log product of these probabilities and the minimal cost plan is also the plan representing the execution branch with maximal success probability. They applied this method to fully observable MDPs with a known initial state. We combine it with the other techniques to handle fully unobservable stochastic planning with an uncertain initial state.

Algorithm 1 RBPP+ $(P, \textit{classical-planner}, \epsilon_1, \epsilon_2)$

$\psi_I, \theta_E \Leftarrow$ RESTRICT+(P);
$P' \Leftarrow$ COMPILE$(V, A, \psi_I, G, \theta_E)$;
return *classical-planner*$(P')\setminus$ *merge-actions;*

Algorithm 2 RESTRICT+ (P, ϵ)

$Q \Leftarrow$ SORT-CLAUSES(P);
$\psi_I \Leftarrow \varphi_I$;
$C \Leftarrow$ *Extract-First*(Q);
while $[(b_{\pi_I}(\psi_I) \geq \theta + \epsilon_1)||RL(C) > 1]\&[b_{\pi_I}(\psi_I) \geq \theta + \epsilon_2]$ **do**
  $\psi_I \Leftarrow$ *RESTRICT-CLAUSE*(C, ψ_I, P);
  $C \Leftarrow$ *Extract-First*(Q);
end while
$\theta_E \Leftarrow \theta / b_{\pi_I}(\psi_I)$;
return ψ_I, θ_E;

Algorithm 3 COMPILE$(V, A, \psi_I, G, \theta_E)$

$\widetilde{A} \leftarrow$ All-outcome-determinization(A);
$P' = (V', A', I', G') \leftarrow$ CP-To-Classical-with-costs$(\widetilde{P} = (V, \hat{A}, \psi_I, G))$;
return P'

Compiling CPPs with Stochastic Actions

RBPP The core observation behind *RBPP* (Taig and Brafman 2014) is that a deterministic CPP $\mathcal{CP} = (V, A, b_{\pi_I}, G, \theta)$ is solvable iff there exists a solvable CP problem $\mathcal{C} = (V, A, b_I, G)$ such that $b_{\pi_I}(\{w \in b_I\}) \geq \theta$. To exploit this, *RBPP* takes a preprocessing approach: relevance analysis identifies those states which would be most profitable to ignore, and a CP problem is defined with an initial state containing those states that are not ignored. This is given to an off-the-shelf CP solver. The identification process is done by the $RESTRICT$ procedure. which exploits the limited failure "allowance," $1 - \theta$, to ignore initial states whose removal would simplify the problem. The most important goal of this process, which we adapt here, is to reduce the *conformant-width* of the problem, if possible. Reduced width leads to exponentially smaller classical problems, and is often the most crucial factor affecting success and failure. Additional restrictions can be useful even with width 1, but have much less impact.

RBPP+ RBPP+ generalizes RBPP to handle stochastic actions. Let $P = (V, A, b_{\pi_I}, G, \theta)$ be the input CPP. Algorithm 1 describes its high-level structure. *RESTRICT+* uses *RBPP*'s restriction mechanism to heuristically choose a subset of initial states to plan from. It returns a set ψ_I of initial states satisfying $b_{\pi_I}(\psi_I) \geq \theta_I$. *RESTRICT+* differs from *RESTRICT* in that it must also decide how much of the allowed failure probability will be devoted to initial state restriction and how much will be devoted to handling uncertainty about action failure – which we refer to as "plan risk."

RESTRICT+ attempts to ensure all clauses C in the initial state description satisfy $RL(C) = 1$. $RL(C)$, introduced in the RBPP planner, measures, for all sub-goals g' to which C is relevant, the maximal number of clauses relevant to g'. This parameter is closely related to the *conformant-width* of the problem. The ability to restrict $RL(C)$ to 1 plays a crucial role in *RBPP*'s ability to solve CPP problems. When $RL(C) = 1$ for all clauses, the translation will result in a polynomial-sized classical problem, increasing significantly the likelihood that it will be solved by the classical planner. If this restriction goal is achieved, "left-over" risk allowance is devoted to plan risk. This is especially important given that our current estimate of plan risk is pessimistic.

To this effect we define two user defined parameters: $\epsilon_1 \geq \epsilon_2 \geq 1 - \theta$, representing the minimal and maximal initial state restriction allowance,. *RESTRICT+* will restrict initial states with weight of at least $1 - (\theta + \epsilon_1)$ (unless no suitable restriction exists). It restricts more states, if this is required to reduce $RL(C)$ to 1, but no more than $1 - (\theta + \epsilon_2)$, thus leaving a "plan risk" of at least $1 - \frac{\theta}{\theta + \epsilon_2}$.

Compile determinizes all actions using all-outcome determiniziation (Yoon, Fern, and Givan 2007), creating a separate deterministic action for each stochastic effect of each action. Next, we translate the resulting deterministic CP problem into classical planning using the *K1* translation (Palacios and Geffner 2009). Now, we integrate the probabilistic information into the classical planning problem, setting the cost of the determinized actions as follows: If a_ϵ is a determinized action originating in a probabilistic outcome ϵ, set cost$(a_\epsilon) = -log(Pr(\epsilon))$. At this point, we seek a plan with cost no higher than $-log(\theta_E)$. This ensures that the probability of the execution branch captured by the classical plan $\geq \theta_E$. To see this, note that a classical plan $\Pi = \langle a_{1_{\epsilon_{i_1}}}, ..., a_{n_{\epsilon_{i_n}}} \rangle$, represents the execution branch of the conformant plan $\Pi = \langle a_1, ..., a_n \rangle$, where the actual effect of action a_j is ϵ_{i_j}. The probability that this branch takes place is $\prod_{k=1}^{n} Pr(\epsilon_{i_k})$ and it is $\geq \theta_E$ iff $\sum_{k=1}^{n} -log(Pr(\epsilon_{i_k}) \leq -log(\theta_E)$. Finally, since we plan for initial states in ψ_I only, our success probability is at least $b_{\pi_I}(\psi_I) \times \theta_E \geq \theta_I \times \theta_E \geq \theta$. This is a conservative estimate because branches other than the branch accounted for by the classical plan could reach the goal as well.

Repeatable Actions In CPP, it is often necessary to execute an action repeatedly to improve its success probability. For example, a block may slip when picking it up (as in the *slippery gripper* domain), but we can try to pick it up again. More specifically, we say that a stochastic outcome $\epsilon \in \Lambda(e)$ of action a is *repeatable* if $(\bigcup_{\hat{\epsilon} \in \Lambda(e) \setminus \epsilon} del(\hat{\epsilon})) \cap (pre(a) \cup con(e) \cup eff(\epsilon)) = \emptyset$. We say that a determinized action a_ϵ, s.t. ϵ is a repeatable outcome, is a **repeatable action**. Repeatable actions can be repeated a few times to increase the success probability of ϵ. Identification and treatment of repeatable actions is important for success in domain with repeatable outcomes.

(Domshlak and Hoffmann 2007) point out that action repetition is a serious challenge for *PFF* and *POND*. Their probability calculation mechanism fails to recognize the need to increase the probability of some fact early on in the plan by repeating it a few times in a row. This results in failure of their search later on, and prevents them from solving many problems for large values of θ. We handle this issue by identifying repeatable actions during the determinization process. Once ϵ has been identified as repeatable, we add k copies of the determinized action a_ϵ to the determinized problem: $\{a_{\epsilon_i} \mid 1 \leq i \leq k\}$ s.t. $cost(a_{\epsilon_i}) = -log(1 - (1 - Pr(\epsilon))^i)$. k is user specified, set in our experiments to ensure that $1 - (1 - Pr(\epsilon))^k \leq 0.0001$, i.e., to ensure that further repetitions will have little effect on the success probability. We treat the new repetitive actions as macro-actions: when the classical plan is mapped back to the CPP plan we replace each instance of a_{ϵ_i} by i instances of a. Although the planner can select i to be its maximal value, we ensure that the minimal number of repetitions is selected by exploiting the underlying metric planner. We add a numeric fluent which action a_{ϵ_i} increase by i, and minimize its value. Using these ideas, we are able to solve all instances marked by the *PFF* authors as challenging.

With the introduction of repeatable actions, the deterministic conformant plan potentially captures multiple execution paths that correspond to different numbers of repetitions of each repeatable action. However, our current mechanism cannot capture the need to repeat a sequence of length greater than one, which may lead to failure in domains where such repetition is needed (e.g "start the engine","drive").

Properties Our algorithm is sound given a sound underlying solver (we omit the proof due to lack of space):

Lemma 1 *Let Π' be a plan for P' (the classical planning problem generated) with $cost(\Pi') \leq -log(\theta_E)$. Let Π be the corresponding plan for P. Let w be the belief state achieved by executing Π in b_{π_I}. Then $Pr(w \models G) \geq \theta$.*

However, our algorithm is incomplete for four reasons. First, RBPP does not systematically consider all initial state restrictions. Second, $K1$ is incomplete for problems with width > 1, and we cannot guarantee restriction to width-1 problems for every θ. Third, we do not perform systematic search over all possible legal choices of θ_I and θ_E. Finally, as noted before, our computation ignores the probability that the plan will succeed on one of the initial states ignored, and that a branch, other than that captured by the determinized plan, will reach the goal. The first three sources of incompleteness are easy to overcome, in theory, by using exhaustive search where required and replacing $K1$ by a complete translation scheme. Such changes, however, are unlikely to have any positive practical impact because of the computational overhead. The last source of incompleteness is more fundamental. It represents a trade-off between accuracy and tractability. Because we avoid maintaining and reasoning about the current distribution during planning, we can scale up much better than planners that devote more effort to accurate belief tracking. Our empirical results demonstrate that while, in principle, we ignore some valid solution plans, the nature of the CPP benchmarks allows us, in practice, to find alternative plans much faster.

	$\theta = 0.25$		$\theta = 0.5$		$\theta = 0.75$		$\theta = 0.9$	
	t/l		*t/l*		*t/l*		*t/l*	
	PFF	RBPP+	PFF	RBPP+	PFF	RBPP+	PFF	RBPP+
slip-grip-1b	**0.19/2**	0.41/9	**0.19/2**	0.53/9	**0.23/3**	0.59/4	**0.36/4**	0.6/10
slip-grip-4b	1.05/**15**	**0.46**/23	3.33/**18**	**0.83/23**	err	**1.03/18**	err	**1.03/23**
SG-10b	22.11/**42**	**0.72**/55	OOT	**0.74/62**	OOT	**1.85/70**	err	**1.23/63**
sand-castle-1	**0.02/1**	0.9/3	**0.3/3**	0.9/**3**	**0.2/5**	0.94/6	**0.9/9**	1.07/**9**
sand-castle-5	1.6/26	**1.4/24**	OOT	**1.84/31**	OOT	**3.6/55**	OOT	**4.1/70**
Pcube-7-uni	1.17/**13**	**1.05/13**	1.17/**18**	**1.11/18**	**1.92/21**	2.07/29	5.48/**25**	**3.55**/37
Pcube-11-uni	**3.42/23**	**4.03**/36	**2.66/32**	5.22/49	OOT	**5.33/84**	OOT	**19.27/93**
LogL-2-2-2	**0.1/9**	0.6/13	**0.1/13**	0.7/19	**0.2/17**	1.1/21	OOT	**1.2/24**
LogL-4-2-2	0.2/17	0.8/35	0.2/24	1.05/48	0.3/32	1.06/44	OOT	**2.01/61**
LogL-4-5-5	11.7/57	**2.71/76**	OOT	**3.16/72**	OOT	**3.39/96**	OOT	**4.26/111**
grid	OOT	**6.71/37**	OOT	**10.25/41**	OOT	**124.23/80**	OOT	OOT

Table 1: Empirical results. t: time in seconds. l: plan length. *OOT:* no result after 30 minutes. 'err'-*PFF* returned error or collapsed.

Empirical Evaluation

We evaluated our algorithm on domains with stochastic actions from the *PFF* repository and on larger versions of these domains. See (Domshlak and Hoffmann 2007) for domain descriptions. We did not experiment with deterministic benchmarks, where $RBPP+$ reduces to $RBPP$, the current state of the art. We modified Palacios and Geffner's $cf2cs$ code for relevance analysis and used *NORSYS NETICA java api* for probabilistic reasoning during the restriction process. We solved the compiled problems by using *METRIC-FF* as a cost-bounded planner. We could have used a cost optimal planner, verifying that its solution is within the required bound, but our compiled problems are too large for current cost-optimal classical planners. Other cost-bounded planners tested were not effective, either. Moreover, by using a metric planner we were also able to optimize the use of repeatable actions, as described earlier.

RBPP+ was compared with *PFF*, the CPP planner that best handles stochastic actions to date. Because *PFF* is known to outperform *POND*, we did not compare against it. Results are presented in Table 1. Each task was tested with four different θ values, both in terms of plan quality and execution time. To make the comparison as fair as possible, we preprocessed the PPDDL inputs to *PFF* adding to them repeatable actions. We note that *RBPP+* results include the overhead of automatically identifying and compiling repeatable actions. Nevertheless, the results clearly show that our method scales by an order of magnitude better than PFF both in terms of the problem size and θ value. In *logistics* and *send-castle* our method solved instances exponentially larger than *PFF* which performs better only on small instances of simple problems. *RBPP+* does output longer plans, which do not grow monotonically with θ. This is because *RBPP+* attempts to maximize the success probability of a specific trajectory while *PFF* can capture the success probability of all trajectories, which allows it to validate shorter plans. We note *RBPP+*'s success on *slippery-gripper* and the larger θ instances of *logistics*, marked most challenging for previous planners due to the existence of repeatable actions. The introduction of repeatable actions to *PFF*'s input improves its performance on problems it could solve before, but does not improve its scalability beyond them.

Benchmarks in the *PFF* repository have width 1. To examine our techniques for handling larger width problems, we modified the deterministic CPP benchmark *grid* into a probabilistic one. *RBPP+* is the only planner that can handle this domain. We also examined the sensitivity of *RBPP+* to the choices of ϵ_1 and ϵ_2. In our experiments they were set arbitrarily: $\epsilon_1 = \frac{1-\theta}{4}$ and $\epsilon_2 = \frac{1-\theta}{10}$. Additional experiments we made show that any choice of values that allows a significant initial state restriction will keep our performance similar. But if the parameter choice forces the restriction process to leave some initial state clauses C with $RL(C) > 1$, the performance is weaker and problems can become unsolvable. Thus, a future improvement would be to automatically set parameters according to needs of the restriction process.

Summary and Future Work

We presented a new algorithm for CPP with stochastic actions that combines diverse techniques from previous compilation-based planners with new ideas. Our planner is sound, but incomplete and limited to a specific family of problems, yet scales up much better than previous CPP solvers on all existing benchmarks. In future work we hope to investigate improved determinization that exploits relevance analysis to focus on relevant outcomes only, reducing the size of the compiled problem and methods for exploiting the metric planner in order to obtain more accurate estimates of the success probability, allowing us to detect shorter valid plans. Another important direction is to extend the repeatable action technique to handle repeatable action sequences.

Acknowledgments The authors were supported in part by ISF grant 933/13, the Lynn and William Frankel Center for Computer Science and the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *J. Artif. Intell. Res. (JAIR)* 26:35–99.

Domshlak, C., and Hoffmann, J. 2007. Probabilistic planning via heuristic forward search and weighted model counting. *J. Artif. Intell. Res. (JAIR)* 30:565–620.

Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search witn implicit belief states. In *ICAPS*, 71–80.

Hoffmann, J., and Brafman, R. I. 2006. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.* 170(6-7):507–541.

Jiménez, S.; Coles, A.; Smith, A.; and Madrid, I. 2006. Planning in probabilistic domains using a deterministic numeric planner. In *The 25th PlanSig WS*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR* 35:623–675.

Taig, R., and Brafman, R. I. 2014. A relevance-based compilation method for conformant probabilistic planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2374–2380.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, 352–.