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

Compiling Conformant Probabilistic Planning Problems into Classical Planning

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

In CPP, we are given a set of actions (assumed deterministic in this paper), a distribution over initial states, a goal condition, and a real value $0 < \theta \leq 1$. We seek a plan π such that following its execution, the goal probability is at least θ. Motivated by the success of the translation-based approach for conformant planning, introduced by Palacios and Geffner, we suggest a new compilation scheme from CPP to classical planning. Our compilation scheme maps CPP into cost-bounded classical planning, where the cost-bound represents the maximum allowed probability of failure. Empirically, this technique shows mixed, but promising results, performing very well on some domains, and less so on others when compared to the state of the art PFF planner. It is also very flexible due to its generic nature, allowing us to experiment with diverse search strategies developed for classical planning. Our results show that compilation-based technique offer a new viable approach to CPP and, possibly, more general probabilistic planning problems.

Introduction

An important trend in research on planning under uncertainty is the emergence of planners that utilize an underlying classical, deterministic planner to solve more complex problems. Two highly influential examples are the replanning approach (Yoon, Fern, and Givan 2007) in which an underlying classical planner is used to solve MDPs by repeatedly generating plans for a determinized version of the domain, and the translation-based approach for conformant (Palacios and Geffner 2009) and contingent planning (Albore, Palacios, and Geffner 2009), where a problem featuring uncertainty about the initial state is transformed into a classical problem on a richer domain. Both approaches have drawbacks: replanning can yield bad results given dead-ends and low-valued, less likely states. The translation-based approach can blow-up in size given complex initial belief states and actions. In both cases, however, there are efforts to improve these methods, and the reliance on fast, off-the-shelf, classical planners seem to be very useful.

This paper continues this trend, leveraging the translation-based approach of Palacios and Geffner (Palacios and Geffner 2009) to handle a version of conformant planning in which there is a probability distribution over the initial state of the world. The task is to reach the goal with certain minimal probability, rather than with certainty. In general, conformant probabilistic planning (CPP) allows for stochastic actions, but as in most earlier work, we focus on the simpler case of deterministic actions. In earlier work (Brafman and Taig 2011), we reduced CPP into Metric Planning. Here, we offer a reduction into a more common variant of classical planning – cost bounded planning.

The key to understanding our translation is the observation that a successful solution plan for a deterministic CPP problem is a conformant plan w.r.t. a (probabilistically) sufficiently large set of initial states. Hence, a possible solution method would be to "guess" this subset, and then solve the conformant planning problems it induces. Our method generates a classical planning problem that manipulates the knowledge state of the agent, but includes additional operators that allow the planner to select the set of states on which it will "plan" for and the set of states it will "ignore". To capture the probabilistic aspect of the problem, our translation scheme uses action costs. Although most actions have identical zero cost, the special actions that tell the planner to "ignore" an initial state have positive cost. More specifically, an action that ignores a state s costs $Pr(s)$. Consequently, the cost-optimal plan is a plan with the highest probability of success – as it ignores the fewest possible initial states.

Given a problem with a conformant (probability 1) solution, a cost-optimal planner applied to the transformed CPP will return this conformant solution, even when it could return a much shorter plan that achieves the desired success probability, more quickly. Worse, a cost-optimal planner may get stuck trying to prove the optimality of a solution even if it encounters a legal plan during search. Indeed, typically, existing cost-optimal classical planners cannot handle the classical planning problem our compilation scheme generates. For this reason, we use a cost-bounded classical planner – i.e., a planner that seeks a classical plan with a given cost bound (Stern, Puzis, and Felner 2011). In our setting, the classical planner tries to find a plan with a cost no greater than $1 - \theta$, which implies that the probability of the initial states that are *not* ignored is at least θ. We also consider a close variant of this translation scheme where the cost bound is modeled as a single resource that, again, captures the probability of states ignored. This variant performs well on a few domains, but is generally weaker, partially because resource-bounded planners are not as well developed.

To assess our proposed translation scheme, we compare our planner to *PFF* (Domshlak and Hoffmann 2007) and our older *PTP* (Brafman and Taig 2011) planner, which are the state of the art in CPP. The results are mixed, showing no clear winner. On some classical domains, we are better, whereas on others PFF is better. However, we have found that PFF is quite sensitive to domain reformulation. For example, it is well known that PFF depends on the order of conditions in conditional effects. In addition, when domains such as logistics are modified to require slightly more complex or longer plan (for example, by splitting the *load* action in logistics into a number of actions, such as: *open-cargo-door*, *load*, *close-cargo-door*), PFF's performance is adversely affected, whereas our compilation method is not. Overall, our results indicate that the translation approach offers a useful technique for solving probabilistic planning problems, worthy of additional attention.

Next, we give some required background and describe our new proposed compilation scheme. Then, we discuss its theoretical properties and a few modifications required to make it practical. We conclude with an empirical evaluation and a discussion of our results and potential future work.

Background

Conformant Probabilistic Planning

The probabilistic planning framework we consider adds probabilistic uncertainty to a subset of the classical ADL language, namely (sequential) STRIPS with conditional effects. STRIPS planning tasks are defined over a set of propositions $\mathcal{P}$. These fluents define a set of possible worlds, the set of truth assignments over $\mathcal{P}$. In addition, for notational convenience, we assume a distinguished world state $\perp$ which represents the *undefined state* or *failure*. A STRIPS planning task is a triple (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 $w \supseteq G$. An action a is a pair $(pre(a), E(a))$ 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 $\perp$. 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 $w \cup add(e) \setminus del(e)$, which we denote by $a(w)$. We will use $\bar{a}(w)$ to denote the state resulting from the sequence of actions $\bar{a}$ in world state w.

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) effects $e, e' \in E(a)$, the result of applying a in w is undefined. Thus, actions cannot be 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 $\overline{a}(w_I) \supseteq G$.

Conformant planning (A, b_I, G) generalizes the above framework, replacing the single initial state with an initial belief state b_I, where the initial belief state is simply the set of states considered possible initially. Now, a plan is an action sequence $\overline{a}$ such that $\overline{a}(w_I) \supseteq G$ for every $w_I \in b_I$.

Conformant probabilistic planning farther extends this, by quantifying the uncertainty regarding the initial state using a probability distribution b_{π_I}. In its most general form, CPP allows for stochastic actions, but we leave this to future work. Thus, throughout this paper we assume deterministic actions only. Conformant probabilistic planning tasks are 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*. As before, G is a conjunction of propositions. b_{π_I} denotes a probability distribution over the world states, where $b_{\pi_I}(w)$ is the probability that w is the true initial world state.

A note on notation and terminology. In the conformant planning case, a belief state refers to a set of states, and the initial belief state is denoted b_I. In the case of CPP, the initial belief state is a distribution over a set of states, and the initial belief state is denoted b_{π_I}. In both cases we use the term *belief state*. Sometimes, in CPP we will use b_I to denote the set of states to which b_{π_I} assigns a positive probability. Note also that there is no change in the definition of actions and their applications in states of the world. But since we now work with belief states, actions can also be viewed as transforming one belief state to another. We use the notation $[b, a]$ to denote the belief state obtained by applying a in belief state b. The likelihood $[b, a](w')$ of a world state w' in the belief state $[b, a]$, resulting from applying action a in belief state b, is given by $[b, a](w') = \sum_{a(w)=w'} b(w)$.

We will also use the notation $[b, a](\varphi)$ to denote $\sum_{a(w)=w', w' \models \varphi} b(w)$, and we somewhat abuse notation and write $[b, a] \models \varphi$ for the case where $[b, a](\varphi) = 1$.

For any action sequence $\overline{a} \in A^*$, and any belief state b, the new belief state $[b, \overline{a}]$ resulting from applying $\overline{a}$ at b is given by
$$[b, \overline{a}] = \begin{cases} b, & \overline{a} = \langle\rangle \\ [b, a], & \overline{a} = \langle a \rangle, a \in A \\ [[b, a], \overline{a}'], & \overline{a} = \langle a \rangle \cdot \overline{a}', a \in A, \overline{a}' \neq \emptyset \end{cases}$$

In many settings achieving G with certainty is impossible. CPP introduces the parameter θ, which 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 $b_{\overline{a}} = [b_{\pi_I}, \overline{a}]$. Because our actions are deterministic, this is essentially saying that $\overline{a}$ is a *plan* if $Pr(\{w : \overline{a}(w) \models G\}) \geq \theta$, i.e,. the weight of the initial states from which the plan reaches the goal is at least θ.

Related Work

The best current probabilistic conformant planner is *Probabilistic FF* (PFF) (Domshlak and Hoffmann 2007). The basic ideas underlying Probabilistic-FF are:

1. Define time-stamped Bayesian Networks (BN) describing probabilistic belief states.

2. Extend Conformant-FF's belief state to model these BN.
3. In addition to the SAT reasoning used by Conformant-FF (Hoffmann and Brafman 2006), use weighted model-counting to determine whether the probability of the (unknown) goals in a belief state is high enough.
4. Introduce approximate probabilistic reasoning into Conformant-FF's heuristic function.

In some domains, *PFF*'s results were improved by our older *PTP* planner (Brafman and Taig 2011). This work is close in spirit to *PTP*. *PTP* compiles CPP into a metric planning problem in which the numeric variables represent the probabilities of various propositions and actions update this information. For every variable p, *PTP* maintains a numeric variable, Pr_p, that holds the probability of p in the current state. *PTP* also maintains variables of the form p/t that capture conditional knowledge. If an action adds p/t, then the value of Pr_t is increased by the probability of t. Similar information about the probability of the goal is updated with a Pr_{goal} variable, and the goal in this metric planning problem is: $Pr_{goal} \geq \theta$. We compare our methods to *PTP* below. The main *a-priori* advantage of our current approach is the simpler translation it offers and the choice of target problem: classical planning received much more attention than metric planning, and many more heuristics are available.

An earlier attempt to deal with probabilities by reducing it to action costs appears in (Jiménez et al. 2006) in the context of probabilistic planning problems where actions have probabilistic effects but there is no uncertainty about the initial state. The probabilistic problem is compiled into a (non-equivalent) classical problem where each possible effect e is represented by a unique action and the cost associated with this action is set to be $1 - Pr(e)$. That value captures the amount of risk the planner takes when choosing that action, which equals the probability that the effect won't take place when the original action is executed. This value is then minimized by the cost-optimal planner. Our compilation scheme uses related ideas but deals with uncertainty about the initial state, and comes with correctness guarantees. In (Keyder and Geffner 2008) the authors also observed the possibility to integrate probabilistic information and original action costs into a new cost function and used it in order to solve the relaxed underlying MDP of a probabilistic planning problem and gain, by that, valuable heuristic information.

Closely related to our work is the *CLG+* planner (Albore and Geffner 2009). This planner attempts to solve contingent planning problems in which goal achievement cannot be guaranteed. This planner makes assumptions, gradually, that reduce the uncertainty, and allow it to plan. This is achieved by adding special actions, much like ours, that "drop" a tag, i.e., assume its value is impossible. These actions are associated with a high cost. The main difference is that the cost we associate with assumption-making actions reflects the probability of the states ruled out, allowing us to model probabilistic planning problems as cost-optimal planning. Furthermore, our planner may decide (depending on the search procedure used) to come up with a sub-optimal plan, albeit one that meets the desired probabilistic threshold, even when a full conformant plan exists. This flexibility allows us to trade-off computational efficiency with probability of success.

Of similar flavor to the above is the assumption-based planning approach introduced recently (Davis-Mendelow, Baier, and McIlraith 2012). This work considers the problem of solving conformant and contingent planning under various assumptions, that may be selected according to various preference criteria. However, it does not consider an explicit probabilistic semantics that addresses CPP.

Cost bounded classical planning

In cost bounded classical planning, a classical planning problem is extended with a constant parameter $c \in \mathbb{R} > 0$. The task is to find a plan with cost $\leq c$ as fast as possible. In this setting the optimal plan cost and the distance of the resulting plan from optimal does not matter, as opposed to notions such as sub-optimal search. One way to solve this problem is to use an optimal planner and then confirm that the cost bound is met. Another method is to use an anytime planner that gradually improves the plan cost until the cost bound is met. However, these methods do not make real use of the bound during the search, e.g., for pruning nodes that cannot lead to a legal solution. Recently, a number of algorithms that deal directly with this problem were suggested (Stern, Puzis, and Felner 2011),(Thayer et al. 2012). The common ground of all these algorithms is the consideration of the bound c within the heuristic function. One example is *Potential Search*. It uses heuristic estimates to calculate the probability that a solution of cost no more than c exists below a given node (Stern, Puzis, and Felner 2011). This idea was extended by the *Beeps* algorithm (Thayer et al. 2012) which chooses which node to expand next, based on the node's potential, which combines admissible and inadmissible estimates of the node's h value as well as an inadmissible estimate of the number of actions left to the goal (distance estimate). This algorithm is currently the state of the art for this problem.

Resource constrained classical planning

Resource constrained planning is a well known extension of classical planning that models problems in which actions consume resources, such as time and energy, and the agent must achieve the goal using some initial amount of resources. Here we follow the formulation of (Nakhost, Hoffmann, and Müller 2010) and (Haslum and Geffner 2001) where a constrained resource planning task extends a simple classical planning task with a set R of resource identifiers as well as two functions:

- $i : R \to \mathbb{R}_{\geq 0}$, i(r) is the initial level of resource $r \in R$.
- $u : (A \times R) \to \mathbb{R}_{\geq 0}$, for each action $a \in A$ and each resource $r \in R$, $u(a, r)$ is the amount of r consumed by an execution of a.

A state $\bar{s}$ is a pair (s, rem) where $rem \in {\mathbb{R}_{\geq 0}}^{|R|}$ holds the remaining amount of each resource when reaching s. To execute action a in $\bar{s}$, its preconditions must hold in s, and for every resource r, its value in rem must be at least as high as the amount of this resource consumed by a.

The Translation Approach

The idea behind the translation approach (Palacios and Geffner 2009) to conformant planning, implemented in the T_0 planner, is to reason by cases. The different cases correspond to different conditions on the initial state, or, equivalently, different sets of initial states. These sets of states, or conditions, are captured by tags. Each tag is identified with a subset of b_I. Below we use slightly different notation from that of Palacios and Geffner, and we often abuse it, treating a tag as the set of initial states it defines.

The translation approach associates a set of tags T_p with every proposition p. Intuitively, these tags represent information about the initial state that is sufficient to determine the value of p at every point of time during the plan. The set of propositions is augmented with new propositions of the form p/t, where t is one of the possible tags for p. p/t holds the current value of p given that the initial state satisfies the condition t. The value of each proposition p/t in the initial state is simply the value of p in the state(s) represented by t. To understand this idea, it is best to equate tags with possible initial states, although this is inefficient in practice. Then, it is clear that what this transformation seeks to represent is the belief state of the agent – if we know what is the current value of each proposition for every initial world state, we have a representation of the belief state of the agent. Thus, the classical planning problem generated is one that emulates planning in belief space.

To succeed, the plan must eventually reach a belief state in which all states satisfy G. It is often said that in such a state, the agent *knows* G. Thus, the classical planner state must also represent the agent's knowledge. For this purpose, the translation method adds new variables of the form Kp for every proposition p which represent the fact that p holds in every currently possible world. Kp is initially true, if p is true in all initial world states with positive probability.

The fact that the new variables are initialized correctly is not sufficient to ensure a correct representation of the agent's knowledge state throughout the plan. For this purpose, the actions used must be modified. If the original actions are deterministic, then the change to the agent's state of knowledge following the execution of an action is also deterministic, and we can reflect it by altering the action description appropriately. Thus, one replaces each original action with a similar action that also correctly updates the value of the p/t and Kp propositions using suitable conditional effects.

Intuitively, Kp should hold whenever p/t holds for all possible tags $t \in T_p$, i.e., p is true now for every possible initial world state. To ensure this, we must add special "inference" actions – actions that do not change the state of the world, but rather deduce new information from existing information – called *merge* actions. To see why this is needed, imagine that g/t holds for all tags $t \in T_g$ except for some t', and that the last action finally achieved g/t'. Thus, now g holds in all possible world states. With the current set of actions, this information is not reflected in the planner's state, i.e., Kg does not hold. Merge actions allow us to add (i.e., deduce) Kg in this case. The preconditions of such an action are simply g/t for every $t \in T_g$, and the effect is Kg.

The resulting problem is a classical planning problem defined on a larger set of variables which reflects the state of the world as well as the state of information (belief state) of the agent. The size of this set depends on the original set of variables and the number of tags we need to add. Hence, an efficient tag generation process is important. As noted above, a trivial set of tags is one that contains one tag for each possible initial state. This leads to an exponential blowup in the number of propositions. However, we can often do much better, as the value of each proposition at the current state depends only on a small number of propositions in the initial state. This allows us to use many fewer tags (=cases). In fact, the current value of different propositions depends on different aspects of the initial state. Thus, in practice, we select different tags for each proposition.

New Translation Scheme for CPP

The basic motivation for the method we present now is the understanding that we can solve a CPP problem by identifying a set b' of initial states whose total probability is greater or equal to θ, such that there exists a conformant plan for b'. This plan is a solution to the CPP problem, too. This is formalized in the following simple lemma whose proof is immediate, but nevertheless underlies our ideas. Please recall that we consider deterministic actions only.

Lemma 1 *A CPP $\mathcal{CP} = (V, A, b_{\pi_I}, G, \theta)$ is solvable iff there exists a solvable conformant planning problem $\mathcal{C} = (V, A, b_I, G)$ such that $Pr(\{w \in b_I\}) \geq \theta$. Moreover, $\overline{a}$ is a solution to $\mathcal{CP}$ iff it is a solution to $\mathcal{P}$.*

Proof: $\mathcal{CP} \Rightarrow \mathcal{C}$: Let $\overline{a}$ be a solution to $\mathcal{CP}$. Choose $b_I = \{w \in b_{\pi_I} | \overline{a}(w) \supseteq G\}$.
$\mathcal{C} \Rightarrow \mathcal{CP}$: Let $\overline{a}$ be a solution to $\mathcal{C}$. By definition, $Pr(\{w \in b_I\}) \geq \theta$, and by definition of conformant plan $Pr(\{w \in b_I | \overline{a}(w) \supseteq G\}) \geq \theta$.$\square$

Lemma 1 tells us that the task of finding a plan to a CPP can be divided into the identification of a suitable initial belief state b_I, followed by finding a plan to the conformant planning problem (V, A, b_I, G). However, rather than take this approach directly, we use a compilation-based method in which we let the classical planner handle both parts. That is, in the classical planning problem we generate the planner decides which states to ignore, and also generates a conformant plan for all other states. We must ensure that the total probability of ignored states does not exceed $1 - \theta$. Technically, this is done by introducing special actions that essentially tell the planner to ignore a state (or set of states). The cost of each such action is equal to the probability of the state(s) it allows us to ignore.

Technically, the effect of ignoring a state is to make it easier for the planner to obtain knowledge. Typically, we say that the agent knows φ at a certain belief state, if φ holds in all world states in this belief state. In the compilation approach such knowledge is added by applying *merge* actions. Once a state has been "ignored" by an "ignore" action, the merge actions effectively ignore it, and deduce the information as if this state is not possible.

Formal description

Let $P = (V, A, b_I, G, \theta)$ be the input CPP. Let T_p be the set of tags for each $p \in V$. (We discuss the tag computation algorithm later – right now it is assumed to be given to the compiler.) We use T to denote the entire set of tags (i.e., $\cup T_p$). We will also assume a special distinguished tag, the empty set. We now present our compilation methods for P.

Given P, we generate the following classical planning with action costs problem $\widetilde{P} = (\widetilde{V}, \widetilde{A}, \widetilde{I}, \widetilde{G})$:

Variables: $\widetilde{V} = \{p/t \mid p \in V, t \in T_p\} \cup \{Drop_t | t \in T\}$. The first set of variables are as explained above. $p/\{\}$, which we abbreviate as simply p, denotes the fact that p holds unconditionally, i.e., in all possible worlds. (Previous work denotes this by Kp.) The second set of variables – $Drop_t$ – denotes the fact that we can ignore tag t.

Initial State: $\widetilde{I} = \{l/t \mid l$ is a literal, and $t, I \models l\}$. All valid assumptions on the initial worlds captured by the special variables. Note that all $Drop_t$ propositions are false.

Goal State : $\widetilde{G} = G$. The goal must hold for all initial states. Recall that what we call knowledge is not real knowledge, because we allow ourselves to overlook the ignored states.

Actions: $\widetilde{A} = \widetilde{A}_1 \cup \widetilde{A}_2 \cup \widetilde{A}_3 \cup \widetilde{A}_4$ where:

- $\widetilde{A}_1 = \{\widetilde{a} \mid a \in A\}$: Essentially, the original set of actions.
  - $pre(\widetilde{a}) = pre(a)$. That is, to apply an action, its preconditions must be known.
  - For every conditional effect $(c \rightarrow p) \in E(a)$ and for every $t \in T$, $\widetilde{a}$ contains: $\{c/t \mid c \in con\} \rightarrow \{p/t\}$. That is, for every possible tag t, if the condition holds given t, so does the effect.
  - $cost(\widetilde{a}) = 0$.
- $\widetilde{A}_2 = \{\{p/t \mid t \in T_p\} \rightarrow p$ |p is a precondition of some action $a \in A$ or $p \in G\}$. These are known as the *merge* actions. They allow us to infer from conditional knowledge about p, given certain sets of tags, absolute knowledge about p. That is, if p holds given t, for an appropriate set of tags, then p must hold everywhere, we set the cost of all the merge actions to 0 as well.
- $\widetilde{A}_3 = \{Drop_t \mid t \in T\}$ where: $pre(Drop_t) = \{\}, eff(Drop_t) = \{Drop_t\}, cost(Drop_t) = Pr_I(t)$. That is, the $Drop_t$ action let's us drop tag t, making $Drop_t$ true in the cost of t's initial probability.
- $\widetilde{A}_4 = \{Assume_{p/t}$ |p is a precondition of some action $a \in A$ or $p \in G, t \in T_p\}$ $pre(Assume_{p/t}) = \{Drop_t\}, eff(Assume_{p/t}) = \{p/t\}, cost(Assume_{p/t}) = 0$. That is, we can assume whatever we want about what is true given a dropped tag.

Cost bound : The cost bound for the classical planning problem is set to $1 - \theta$.

Thus, using the $Drop_t$ action, the planner decides to "pay" some probability for dropping all initial states that correspond to this tag. Once it drops a tag, it can conclude whatever it wants given this tag; in particular, that the goal holds. A solution to $\widetilde{P}$ will make assumptions whose cost does not exceed the bound. Hence, it will work on a (probabilistically) sufficiently large set of initial states.

Example

We illustrate the ideas behind our planner *PCBP* using an example adapted from (Palacios and Geffner 2009) and (Brafman and Taig 2011). In this problem we need to move an object from the origin to a destination on a linear grid of size 4. There are two actions: $pick(l)$ picks an object from location l if the hand is empty and the object is in l. If the hand is full, it drops the object being held in l. $put(l)$ drops the object at l if the object is being held. All effects are conditional, and there are no preconditions. Formally, the actions are as follows: $pick(l) = (\{\}, \{at(l) \wedge \neg hold \rightarrow hold \wedge \neg at(l), hold \rightarrow \neg hold \wedge at(l)\})$, and $put(l) = (\{\}, \{hold \rightarrow \neg hold \wedge at(l)\})$. Initially, the hand is empty, and the object is at l_1 with probability 0.2, or at l_2 or l_3 with probability 0.4, each. It needs to be moved to l_4 with a probability $\theta = 0.5$. Thus, we can attain the desired probability by considering two initial states only, whereas conformant planning requires that we succeed on all three locations.

The tag sets we require are: $T_L = \{at(l_1), at(l_2), at(l_3)\}$ for $L = at(l_4)$. These tags are disjoint, deterministic, and complete. (These notions are explained later). Given T_L, our algorithm generates the following bounded-cost planning task: $\hat{P} = \{\hat{V}, \hat{R}, \hat{A}, \hat{I}, \hat{G}\}$ where $\hat{V} = \{L/t \mid L \in \{at(l), hold\}, l \in \{l_1, l_2, l_3\}\} \cup \{DROP_t \mid t \in \{at(l_1), at(l_2), at(l_3)\}\}$. $\hat{I} = \{\neg hold, \{at(l)/at(l) \mid l \in \{l_1, l_2, l_3\}\}\}$. $\hat{A}$ consists of three parts: modified original actions, *merge* actions, and *drop* and *assume* actions, some of which we illustrate:

- Consider the conditional effect of $pick(l)$, $(\neg hold, at(l) \rightarrow hold \wedge \neg at(l))$, the output consists of the following conditional effects: $\{(\neg hold, at(l) \rightarrow hold), \{(\neg hold/at(l'), at(l)/at(l') \rightarrow hold \wedge \neg at(l) \wedge hold/at(l')) | \forall l' \in \{l_1, l_2, l_3\}\}\}$.
- Consider the $Merge_{at(l_4)}$ action: it has one conditional effect and no pre-conditions: $at(l_4)/at(l_1) \wedge at(l_4)/at(l_2) \wedge at(l_4)/at(l_3) \rightarrow at(l_4)$.
- $Drop_{at_{l_3}}$ has no pre-conditions and a single effect $Drop_{at_{l_3}}$. It cost equals the initial probability of at_{l_3}: 0.4. We need also to be able to transform $Drop$ into "knowledge". For example, $Assume_{at_{l_4}/at_{l_3}}$ with precondition $Drop_{at_{l_3}}$ and a single effect is at_{l_4}/at_{l_3}.

A possible plan for $\hat{P}$ is: $\pi = \langle pick(l_1), put(l_4), pick(l_2), put(l_4), Dropat_{l_3}, Assume_{at_{l_4}/at_{l_3}}, Merge_{at(l_4)} \rangle$. The cost of π is $0.4 < 1 - \theta = 0.5$, so it is within the cost bound. If we drop from π all non-original actions, we get a plan for the original CPP.

Theoretical Guarantees

We now state some formal properties of this translation scheme. We will focus on the (admittedly wasteful) complete set of tags. That is, below, we assume that $T_p = b_I$ for all propositions p, i.e., there is a tag for every initial state with positive probability. These results essentially follow from Lemma 1 and soundness and completeness claims of Palacios and Geffner (Palacios and Geffner 2009).

Theorem 2 *The original CPP has a solution iff the cost-bounded classical planning problem generated using the above translation scheme has a solution. Moreover, every solution to the CPP can be transformed into a solution to the corresponding cost-bounded problem, and vice versa.*

Proof: (sketch) The proof proceeds by explaining how a plan for the CPP can be transformed to a plan for the cost-bounded problem and vice versa. The transformation is simple. 1) For every initial world state w on which the CPP solution plan does not achieve the goal, add a $Drop_w$ action at the beginning of the plan. We call these states *dropped* states. 2) Replace each action in the CPP solution with the corresponding classical action generated by the translation scheme. We call these *regular* actions. 3) Following each regular action, insert all possible *merge* actions.

We claim that the resulting plan is a solution to the cost-bounded planning problem generated by the translation scheme. First, note that the probability of states on which the CPP solution does not achieve the goal is no greater than $1 - \theta$, by definition. Hence, the cost of the $Drop_w$ actions inserted is no greater than $1 - \theta$. Since all other actions have zero cost, the cost bound is met. Next, focusing only on tags p/w where w is not a dropped world state, we know from the soundness and completeness results of Palacios and Geffner that following each regular action a the value of p/w variables correctly reflects the current (post a) value of p if w was the initial world state, and that following the application of all merge actions inserted after a, p holds in the classical planner state iff p holds in the post-a belief state during the execution of the conformant plan. Taking into account our modified *merge* actions, this implies that p holds after the application of a iff p is true after the application of a in all world states that are the result of applying the current plan prefix on non-dropped initial world states. We know that G holds on these states at the end of the conformant plan. Consequently, G holds at the end of the classical plan.

For the other direction. Suppose we are given a plan for the cost bounded classical planning problem. Imagine it has the form of the classical plan discussed above (i.e., *Drop* actions at first, and all *merge* actions following every regular action). In that case, we would get the result thanks to the correspondence established above. However, observe that we can add all merge actions as above following every regular action, and push all *Drop* actions to the start, and the plan would remain a valid solution. First, the added actions have zero cost, so the plan cost remains unchanged. Second, *Drop* actions have no preconditions (so can appear anywhere we like), nor do they have negative effects (hence will not destroy a precondition), nor can they activate an inactive conditional effect. Since we add no new *Drop* actions, the plan cost does not change. Similarly, by adding *merge* actions, we cannot harm the plan, as they can only lead to the conclusion of more positive literals of the form Kp, which will not destroy any preconditions of existing actions, nor will they lead to the applicability of any new conditional effects.□

Safe plans

Here we note a subtle semantic point about CPP. Most previous authors consider a slightly different definition of a solution plan for a CPP: $\bar{a}$ is a *safe plan* if $Pr(\{w : \bar{a}(w) \models G\}) \geq \theta$ **and** $\bar{a}(w) \neq \perp$ for every $w \in b_I$. That is, a safe plan is not only required to succeed on a sufficiently likely set of initial states, but it must also be well defined on all other initial states. This latter requirement makes sense when applying an inapplicable action is a catastrophic event. But if all we truly care about is the success probability, then the former definition appears more reasonable. Our methods and our planner are able to generate both plans and safe plans. Since the theoretical ideas are slightly easier to describe using the former semantics we used this formulation (note that Lemma 1 is not true as stated if we seek *safe* plans). The plan generated by our translation is not necessarily *safe*, even if it appears that the preconditions of an action must be known prior to its execution. The reason is that due to the definition of the merge actions, we can conclude $p/\{\}$ even if p/t is not true for all tags. It is enough that p/t hold given tags that were not dropped. To generate a safe plan, we can, for instance, maintain two copies of each proposition of the form $p/\{\}$, e.g., $K_u p$ and $K_s p$, representing unsafe and safe knowledge, with appropriate merge actions for each precondition. Perhaps surprisingly, safe plans are, in some sense, computationally easier to generate using the translation method. In unsafe plans, we have to be more sensitive, and be able to realize that the preconditions were satisfied with the desired probability. This results in larger sets of relevant variables, and therefore, larger tag sets.

From theory to practice

The scheme presented above must address two main difficulties to become practical:

1. The size of the classical problem outputted crucially depends on tag sets sizes. Unfortunately, some techniques used in conformant planning to minimize tag sets do not work in CPP. We explain what can be done.
2. Cost-bounded planners' performance is quite sensitive to the properties of the planning problems. Thus, we must adjust the planners to handle well problems generated by the translation process.

Computing Tags Our tag generation process is motivated by the K_i variant of the $K_{T,M}$ translation introduced by Palacios and Geffner (Palacios and Geffner 2009), where i is the conformant width of the problem. It slightly modifies it to handle the special character of the probabilistic setting.

Recall that we equate tags with sets of initial states. We seek a *deterministic, complete* and *disjoint* set of tags. We say that T_p is *deterministic* if for every $t \in T_p$ and any sequence of actions $\bar{a}$, the value of p is uniquely determined by t, the initial belief state b_I and $\bar{a}$. We say that T_p is *complete* w.r.t. an initial belief state b_I if $b_I \subseteq \bigcup_{t \in T_p} t$. That is, it covers all possible relevant cases. We say that a set of tags is *disjoint* when for every $t \neq t' \in T_p$ we have that $t \cap t' = \emptyset$. We say that a set of tags is *DCD* if it is deterministic, complete, and disjoint.

When tags are *deterministic* then $p/t \vee \neg p/t$ is a tautology. In this case, and assuming the tag set is complete, we know in each step of the planning process exactly under what initial tags p is currently true. If, in addition, the set of tags is disjoint, then the probability that p holds now equals the sum of probabilities of all tags t for which p/t currently holds.

Theorem 3 *Theorem 2 holds when the set of tags b_I is replaced by any DCD set of tags.*

Proof: (Sketch) Following the results of Palacios and Geffner, determinism and completeness are sufficient to ensure accurate values to the p/t values. This is also easy to see directly, as all states within a tag lead to the same value of p at each point in time, and completeness just ensures that we do not "forget" any initial world state. Disjointness is required to maintain the equivalence between plan cost and probability of failure. If tags are not disjoint, then states in the intersection of tags t, t' are double counted if we apply $Drop_t$ and $Drop_{t'}$. If tags are disjoint, then we do not face this problem. However, it may appear that if we use general tags, rather than states, we will not be able to fine-tune the probability of dropped states. For example, imagine that there are ten equally probable initial states, but only two tags, each aggregating five states. Imagine that $\theta = 0.6$. It seems that by using two tags only, we are forced to consider only success on all 10 states or success on fives states only, and cannot, for example, seek success on six states, as required here. However, the fact that a set of states was aggregated into a tag, implies that all states in this tag behave identically (with respect to the relevant proposition). Thus, it is indeed the case that it is only possible to achieve this proposition with probability of 0,0.5, or 1. Hence, there is no loss in accuracy.□

A trivial example of a DCD set of tags is one in which every state $w \in b_I$ is a tag. But of course, this leads to an exponential blowup in the number of propositions. Instead, we seek a DCD set of tags that is as small as possible. Thus, if two states w, w' differ only on the value of propositions that do not impact the value of p, they should belong to the same tag for p. Consequently, the tags generation process for a predicate p is based on identifying which literals are relevant to its value using the following recursive definition:

1. $p \in rel(p)$.
2. If q appears (possibly negated) in an effect condition c for action A such that $c \rightarrow r$ and r contains p or $\neg p$ then $q \in rel(p)$
3. If $r \in rel(q)$ and $q \in rel(p)$ then $r \in rel(p)$.

We define the set of tags C_p to be: those truth assignments π to $rel(p)$ such that $b_I \wedge \pi$ is consistent. The set of initial states that correspond to tag $\pi \in C_p$ is $\{s \in b_I | s \models \pi\}$.

Lemma 4 *C_p is a DCD set of tags for p.*

Proof: *(sketch)* C_p is complete and disjoint by definition. To see that it is deterministic, let $s, s' \in b_I$ be two possible initial states. We claim that $\overline{a}(s)$ and $\overline{a}(s')$ agree on p for any action sequence $\overline{a}$. The proof is immediate by mutual induction on all variables. The induction is on the length of $\overline{a}$. The base case $n = 0$ follows from (1) above ($p \in rel(p)$). The inductive step follows from conditions (2) & (3).□

In practice, checking the consistency of a large set of assignments may be expensive, and instead one can use the tag computation scheme used by P&G to address this problem (Palacios and Geffner 2009) for the full definition of this algorithm) which requires only examining the clauses of the initial state, with the following important change: a non unit clause $\varphi \in b_I$ is considered relevant to a proposition p if $\exists q \in \varphi$ s.t $q \in rel(p)$. Note that P&G consider φ relevant to p only if all propositions of φ are in $rel(p)$. The latter, of course yields fewer relevant propositions, and hence smaller tags. Unfortunately, CPP is more sensitive to initial state values, and so the stricter definition is required.

Searching for a Classical Plan Our goal is to find a solution to the classical problem that satisfies the cost bound as quickly as possible. Our first observation – well known in the case of cost-optimal planning – is that cost-bounded and cost-optimal planners have trouble dealing with zero-cost actions. Consequently, our planner gives, in practice, each zero-cost action a very small weight – smaller than the least likely initial possible state. Our second observation is that "real" planning from a possible initial state requires longer plans, while shorter plans will contain the costly *Drop* actions. Ideally, we would like to direct the search process towards as short as possible plan, reducing search depth, and thereby reducing the number of nodes expanded. We can get shorter plans by preferring states that appear to be closer to the goal. Yet, such states will often be more expensive, and may actually be dead-ends. To prune such states, we need to consider real h values. We have found that what works for us is using greedy search on h, breaking ties based on distance estimate using inadmissible estimates. Ties among promising states are then broken in favor of shorter plans that suitably combine realistic solutions (i.e., ones that can meet the cost bound) with as many *Drop* actions as possible.

An alternative compilation we considered is into *resource constrained classical planning*. The resource is the amount of probability we can "waste" (i.e., $1 - \theta$). The $Drop_s$ action's cost is replaced by an additional effect which consumes $Pr_I(s)$. All other details remains the same. Bounding cost in this problem and bounding resource in the new problem are virtually the same, except the latter can be solved by resource-aware planners. We found that current resource constrained classical planners have difficulty handling complex domains, although on simpler domains they fare better than other methods (see empirical results).

Empirical Evaluation

We implemented the algorithms and experimented with them on a variety of domains taken from *PFF* repository and the 2006 IPPC, as well as slightly modified versions of these domains. Our implementation combines our own tools with a modified versions of the *cf2cs* program, which is a part of the *T-0* planner, used to generate the set of tags. We used *Metric-FF* (Hoffmann 2003) as a resource constrained planner where resources are modeled as numeric variables and suitable pre-conditions are added to actions which consume resources to ensure resource level remain non-negative. We refer to the resulting planner as *PRP*. For cost bounded planning (resulted planner referred as *PCBP*) we used the *FD* (Helmert 2006) based implementation of cost bounded

		$\theta = 0.25$				$\theta = 0.5$				$\theta = 0.75$				$\theta = 1.0$			
Instance	#actions/#facts/#states	t/l				t/l				t/l				t/l			
		PFF	PTP	PRP	PCBP	PFF	PTP	PRP	PCBP	PFF	PTP	PRP	PCBP	PFF	PTP	PRP	PCBP
Safe-uni-70	70/71/140	2.65 /18	0.87/18	**0.55/70**	6.4/70	5.81/35	0.85/35	**0.55/70**	6.4/70	10.1/53	0.9/53	**0.59/70**	6.4/70	5.1/70	0.88/70	**0.65/70**	6.4/70
Safe-cub-70	70/70/138	0.88/5	0.9/5	**0.55/70**	5.3/70	1.7/12	0.94/12	**0.55/70**	5.3/70	3.24/21	0.95/21	**0.55/70**	5.3/70	4.80/70	0.96/70	**0.55/70**	5.3/70
H-Safe-uni-70		1080/53	–	–	**19/53**	5min$<$	–	–	**18.6/105**	5min$<$	–	–	**17.1/159**	5min$<$	–	–	**15.8/211**
H-Safe-cub-70		250/15	–	–	**19.1/32**	544/36	–	–	**18.2/73**	10min$<$	–	–	**25.8/123**	5min$<$	–	–	–
NC-Safe-uni-70	70/71/140	11.21/18	NT	NT	**3.37/18**	11.41/35	NT	NT	**5.51/35**	15.25/53	NT	NT	**7.4/53**		NT	NT	NT
Cube-uni-corner-11	6/90/3375	4.25/26	2.4/33	**0.8/30**	16.5/30	6.35/34	2.49/45	**0.84/34**	14.3/34	9.20/38	2.65/50	**0.7/38**	12.6/38	31.2/42	2.65/50	**0.73/42**	12.7/42
Cube-cub-corner-11	6/90/3375	**0.3/5**	1.17/12	0.61/7	10.3/7	0.9/9	1.31/15	**0.7/14**	10.2/14	1.43/13	1.41/21	**0.7/15**	11.5/15	28.07/31	3.65 /36	**1.1/31**	8.9/31
Push-Cube-uni-15	–	150/50	–	–	**13.4/70**	348/66	–	–	**11.5/84**	647/74	–	–	**13.4/125**	43.9/65	–	–	**14/132**
Push-Cube-cub-15	–	24.5/14	–	–	**10.2/45**	15.7/17	–	–	**10.2/45**	16.7/18	–	–	**10.2/45**	59/45	–	–	**10.2/45**
Bomb-50-50	2550/200/$> 2^{100}$	0.01/0	0.01/0	0.1/0	0.01/0	**0.10/16**	3.51/50	36.1/50	90/50	**0.25/36**	3.51/50	36.1/50	90/50	**0.14/51**	3.51/50	36.1/50	90/50
Bomb-50-10	510/120/$> 2^{60}$	0.01/0	0.01/0	0.1/0	0.1/0	**0.89/22**	1.41/90	2.05/66	8.4/90	4.04/62	**1.41/90**	2.8/42	8.4/90	1.74/90	1.46/90	**1.45/90**	8.4/90
Bomb-50-5	255/110/$> 2^{55}$	0.01/0	0.01/0	0.1/0	0.1/0	1.70/27	**1.32/95**	1.35/47	8.5/95	4.80/67	1.32/95	**1.15/71**	8.5/95	2.17/95	1.32/95	**0.9/95**	8.5/95
Bomb-50-1	51/102/$> 2^{51}$	0.01/0	0.01/0	0.1/0	0.1/0	2.12/31	**0.64/99**	0.85/50	5.6/49	6.19/71	**0.64/99**	0.7/74	4.7/74	2.58/99	**0.64/99**	0.7/99	21.3/99
H-Bomb-50-50	2550/200/$> 2^{100}$	0.01/0	0.01/0	0.1/0	0.1/0	5min$<$	–	–	**75/114.2**	5min$<$	–	–	**89/124**	5min$<$	–	–	**200/487.6**
H-Bomb-50-10	510/120/$> 2^{60}$	0.01/0	0.01/0	0.1/0	0.1/0	5min$<$	–	–	**90/19.9**	5min$<$	–	–	**142/22.8**	5min$<$	–	–	**240/23.2**
H-Bomb-50-5	255/110/$> 2^{55}$	0.01/0	0.01/0	0.1/0	0.1/0	5min$<$	–	–	**95/14.7**	5min$<$	–	–	**147/14.2**	5min$<$	–	–	**245/13.8**
H-Bomb-50-1	51/102/$> 2^{51}$	0.01/0	0.01/0	0.1/0	0.1/0	5min$<$	–	–	**99/18.2**	5min$<$	–	–	**151/13.9**	5min$<$	–	–	**249/8.9**
10-Log-2	3440/1040 /$> 20^{10}$	**1.14/54**	–	–	51/70	**5.29/62**	–	–	111.1/60	**6.63/69**	–	–	110/73	**2.14/78**	–	–	290/116
10-Log-3	3690/1260 /$> 30^{10}$	**2.85/64**	–	–	84.2/83	**8.80/98**	–	–	112.15/114	**4.60/99**	–	–	–	**4.14/105**	–	–	–
10-Log-4	3960/1480/$> 40^{10}$	**2.46/75**	–	–	–	**8.77/81**	–	–	–	**6.20/95**	–	–	–	**8.26/107**	–	–	–
8-H-Log-3	3690/1260 /$> 30^{10}$	10min$<$	–	–	**101.4/83**	10min$<$	–	–	**174.3/86**	10min$<$	–	–	**331.5/96**	10min$<$	–	–	**383/125**
8-H-Log-4	3960/1480/$> 40^{10}$	18min$<$	–	–	**690/70**	18min$<$	–	–	**590.6/83**	18min$<$	–	–	**904.1/110**	18min$<$	–	–	**825.7/133**
H-Rovers	393/97/$> 6^3 * 3^8$	**4.4/62**	–	–	8.3/66	5.8/63	–	–	**2.9/53**	9.4/59	–	–	**7.1/71**	**3.2/68**	–	–	4.2/87

Table 1: Empirical results. t: time in seconds. l: plan length. Entries marked – means the search did not return after 30 minutes, 'k-min$<$' means that after k minutes PFF declared it has reached it's max plan length limit. 'NT'-not tested due to technical issues.

search algorithms by (Thayer et al. 2012), slightly modified to our search needs. Their algorithms require a cost estimate as well as a distance estimate. We used *FF* (Hoffmann 2001) as the search heuristic and a count of the number of actions from the current state to the goal in the relaxed problem as the distance to go function. Table 1 shows the results. On the *bomb, cube, and safe* domains, *PRP* works as well or better than *PFF*, with few exceptions, such as bomb-50-50, bomb-50-10 and cube-11 for lower values of θ. On these domains, *PCBP* is slower. *NC-safe* is a modification where no conformant plan exists, *PCBP* dominates *PFF* in this domain. The *H-safe* and *H-bomb* domains are slight extensions of the known benchmarks where additional actions are needed in order to try a code on the safe (e.g. *choose-code*,*type-code*) or for disarming a bomb (e.g. *choose-bomb*,*prepare-to-disarm*). Longer plans are now needed so 'real' planning from a possible initial state requires more actions than choosing to ignore it. On these problems, *PCBP* clearly dominates all other planners which fail to scale up and handle these problems. The *PUSH-CUBE* domain is a version of the known *15-CUBE* domain. This domain describes a game where the user doesn't know the initial location of a ball in the cube. The user can hit any of the 15^3 locations and by that, if the ball is in that location he'll be moving to a neighboring location according to the direction of the user chooses. The goal is to bring the ball into a specific location in the cube. The results for this domain, which requires many decisions by the solver, shows clear dominance of *PCBP*. The *m-logistics-n* problem is the known logistics domain with m packages and m cities of size n. Here the resource-based method fails completely; *PCBP* is also slow and often fails, as it seems the planner prefers to seek "real" plans rather than ignore states. However, on a natural extension of this domain (denoted *H-logistics*) where additional actions are required before loading a package to a truck *PFF* does not scale beyond 7 cities, whereas *PCBP* does much better. *H-ROVERS* domain is a modification of the *ROVERS* domain on two levels. First, taking an image requires additional set up actions, in the spirit of previous modifications. Second, the amount of uncertainty in the initial state is reduced so problem width will become 1 as *PCBP* cannot handle problems with width greater than 1 at present.

Conclusion

We described a new translation scheme for CPP into cost-bounded classical planning that builds upon the techniques of (Palacios and Geffner 2009) and their extension to CPP (Brafman and Taig 2011). This is done by allowing the planner to select certain states to ignore, as long as their combined cost (= probability) is not greater than $1 - \theta$.

In future work we intend to focus on two directions which we believe will help our techniques scale better: search technique and tag generation. Based on our current experience, we believe that fine-tuning the cost-bounded search parameters can have much impact on its ability to deal with compiled planning problems. In addition, our tag generation process is currently inefficient, and needs to be improved to deal with problems whose conformant width is greater than 1.

Acknowledgments The authors were partly supported the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.

References

Albore, A., and Geffner, H. 2009. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *ICAPS'09 Planning and Plan Execution for Real-World Systems Workshop*.

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

Brafman, R. I., and Taig, R. 2011. A translation based approach to probabilistic conformant planning. In *ADT*.

Davis-Mendelow, S.; Baier, J.; and McIlraith, S. 2012. Making reasonable assumptions to plan with incomplete information. *HSDIP 2012* 69.

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

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. *Artificial Intelligence* 129(1-2):5–33.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

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

Hoffmann, J. 2001. Ff: The fast-forward planning system. *AI Magazine* 22(3):57–62.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291–341.

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

Keyder, E., and Geffner, H. 2008. The hmdp planner for planning with probabilitiesr. In *Sixth International Planning Competition (IPC 2008)*.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2010. Improving local search for resource-constrained planning. In *SOCS*.

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

Stern, R.; Puzis, R.; and Felner, A. 2011. Potential search: a bounded-cost search algorithm. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*.

Thayer, J.; Stern, R.; Felner, A.; and Ruml, W. 2012. Faster bounded-cost search using inadmissible estimates. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*.

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