Artificial Intelligence 245 (2017) 134–161

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Robust planning with incomplete domain models

Tuan Nguyen, Sarath Sreedharan, Subbarao Kambhampati *

School of Computing, Informatics and Decision System Engineering, Arizona State University, USA

ARTICLE INFO

Article history:
Received 8 November 2015
Received in revised form 20 December 2016
Accepted 23 December 2016
Available online 19 January 2017

Keywords:
Robust planning
Incomplete domain models

ABSTRACT

Most current planners assume complete domain models and focus on generating correct plans. Unfortunately, domain modeling is a laborious and error-prone task, thus real world agents have to plan with incomplete domain models. While domain experts cannot guarantee completeness, often they are able to circumscribe the incompleteness of the model by providing annotations as to which parts of the domain model may be incomplete. In this paper, we study planning problems with incomplete domain models where the annotations specify *possible* preconditions and effects of actions. We show that the problem of assessing the quality of a plan, or its plan robustness, is $\#P$-complete, establishing its equivalence with the weighted model counting problems. We present two approaches to synthesizing robust plans. While the method based on the compilation to conformant probabilistic planning is much intuitive, its performance appears to be limited to only small problem instances. Our second approach based on stochastic heuristic search works well for much larger problems. It aims to use the robustness measure directly for estimating heuristic distance, which is then used to guide the search. Our planning system, *PISA*, outperforms a state-of-the-art planner handling incomplete domain models in most of the tested domains, both in terms of plan quality and planning time. Finally, we also present an extension of *PISA* called *CPISA* that is able to exploit the available of past successful plan traces to both improve the robustness of the synthesized plans and reduce the domain modeling burden.

1. Introduction

In the past several years, significant strides have been made in scaling up plan synthesis techniques. We now have technology to routinely generate plans with hundreds of actions. All this work, however, makes a crucial assumption—that the action models of an agent are completely known in advance. While there are domains where knowledge-engineering such detailed models is necessary and feasible (e.g., mission planning domains in NASA and factory-floor planning), it is increasingly recognized (cf. Kambhampati [18]) that there are also many scenarios where insistence on correct and complete models renders the current planning technology unusable. The incompleteness in such cases arises because domain writers do not have the full knowledge of the domain physics, or when the planner is embedded into an integrated architecture where the domain model is being learned incrementally.

One tempting idea is to wait until the models become complete, either by manual revision or by machine learning. Alas, the users often do not have the luxury of delaying their decision making. For example, while existing domain modeling

* Corresponding author.
E-mail addresses: atnhan@gmail.com (T. Nguyen), ssreedh3@gmail.com (S. Sreedharan), rao@asu.edu (S. Kambhampati).

http://dx.doi.org/10.1016/j.artint.2016.12.003
0004-3702/

tools [27,19] can make significant efforts to correct any detectable omissions in the specified models by alerting the domain writers, elicitation of complete models is impossible when the writers themselves do not know the full dynamics. Similarly, although there exist efforts [1,34,37,4] that attempt to either learn models from scratch or revise existing ones, their operation is contingent on the availability of successful plan traces or access to execution experience. There is thus a critical need for planning technology that can get by with partially specified domain models, and yet generate plans that are "robust" in the sense that they are likely to execute successfully in the real world.

Although the domain modelers cannot provide complete models, often they are able to provide "annotations" on the partial model circumscribing the places where it is incomplete. In automated planning, Garland and Lesh [11] was the first, to the best of our knowledge, to allow annotations on the specification of incomplete actions; these annotations specify parts of the domains not affecting or being affected by the corresponding actions. The notion of plan quality in their work is defined in terms of four different types of "risks", which however has tenuous heuristic connections with the likelihood of successful execution of plans.

In this research, annotations on the incompleteness of the domains specify *possible* preconditions and effects of actions. (It should be straightforward to extend the existing model acquisition tools to support acquisition of incompleteness annotations.) These annotations facilitate, though only conceptually, the enumeration of all "candidate" complete models, and thus the counting of models under which a plan succeeds. The corresponding *robustness* measure for plans, therefore, captures exactly the probability of success for plans given such an incompleteness language. Given an incompletely specified domain model, an action of a plan might fail to apply during the execution of the plan. Depending on the planning scenario, a user might want to terminate the current plan (and triggering replanning process) or continue its execution after an action failure. This work considers two different semantics for plan execution. In the Generous Execution semantics, the failure to execute of an action does not automatically cause plan failure. Thus additional actions could be inserted into an existing plan for robustifying against potential action failures. The STRIPS Execution semantics, on the other hand, follows STRIPS-style planning [9] preventing a plan from continuing its execution when one of its actions fails to apply. The problem of assessing robustness of a given plan under the two execution semantics is studied using model counting techniques, and its complexity is also established.

Two approaches are proposed for synthesizing robust plans. The first one translates this problem into a conformant probabilistic planning problem [8]. While perhaps the most intuitive approach, this compilation method appears to work only for small planning instances given the state-of-the-art conformant probabilistic planner, Probabilistic-FF [8]. We present the details of the compilation under the Generous Execution semantics and briefly discuss the approach for the STRIPS Execution semantics.

The second approach is a heuristic search method that works well in much larger problem instances. It aims to use the robustness measure directly for estimating heuristic distance, which is then used to guide the search. In the current work, we fully investigate the heuristic approach under the STRIPS Execution semantics. The novel idea is to overcome the complexity of computing the exact robustness measure by exploiting the structures of the correctness constraints for plans. This results in the lower and upper bounds for the robustness measure that can be incorporated in the extraction of robust relaxed plans and guiding the search for robust plans. The experiments show that the resulting planner, *PISA* (**P**lanning with **I**ncomplete **S**TRIPS **A**ctions), outperforms DeFault, a planner that can handle incomplete STRIPS models [31], in most of the tested domain both in terms of plan robustness and in planning time.

We also present a novel extension of *PISA* called $\mathcal{C}PISA$ that is able to exploit past successful plan traces to both improve the robustness of the synthesized plans and reduce the domain modeling burden. As we shall see, $\mathcal{C}PISA$ can be viewed as a bayesian learning extension to *PISA* that starts with a distribution over domain models (that are consistent with the annotated STRIPS model), and learning from past successful plan cases to compute a new posterior distribution over the domain models, which it then uses to compute robust plans for new problems. We will present empirical results demonstrating the effectiveness of $\mathcal{C}PISA$. We will also compare $\mathcal{C}PISA$ to RIM [36], an existing system that starts with a single (incorrect) domain model and modifies it in light of available successful plan traces.

This journal article is a unification of two prior conference publications, [21] and [20], with several significant extensions. The most important extension is the introduction of $\mathcal{C}PISA$, and its evaluation comparing to RIM. In addition to this, an approach to assessing plan robustness with the Generous Execution semantics is presented, and an approximate, fast to compute, transition function is introduced for the STRIPS Execution semantics and used during the forward heuristic search.

The rest of the paper is organized as follows. We start with a discussion of related work in Section 2. Section 3 formulates the planning problems with incomplete domain models. Section 4 formalizes the robustness measure for plans under incomplete domain models. Section 5 presents a spectrum of problems given an incomplete domain model. Section 6 shows a method to assess the plan robustness measure for the two execution semantics using weighted model counting, and then establishes the complexity of the plan robustness assessment problem. Section 7 presents a compilation approach to synthesizing robust plans under the Generous Execution semantics. Section 8 presents a heuristic approach to synthesizing robust plans given incomplete STRIPS domain models, which includes a method to approximate plan robustness and a procedure for extracting robust relaxed plans. Section 9 shows how the approach can exploit successful plan traces to improve the robustness of the plans it generates. The contributions of this work are summarized in Section 10.

2. Related work

As mentioned earlier, Garland and Lesh [11] share the same objective with us on generating robust plans under incomplete domain models. However, their notion of robustness, which is defined in terms of four different types of risks, only has tenuous heuristic connections with the likelihood of successful execution of plans. Robertson and Bryce [24] focus on the plan generation in Garland and Lesh model, but their approach still relies on the same unsatisfactory formulation of robustness. Weber and Bryce [31] use the same formulation with Nguyen et al. [22] and propose a heuristic approach to search for plans minimizing their *risks*, which is essentially one minus plan robustness. Their planner, DeFault, employs a systematic search guided by an FF-like heuristic, breaking ties on a so called "prime implicant" heuristic. It however does not directly estimate the risk or robustness measures that are supposed to be optimized, but rather uses them indirectly to break ties over the standard FF heuristic. Using this tie breaking heuristic as the main guidance for the search, as observed by Weber and Bryce, does not result in an informative heuristic for generating low risk plans. Zhuo et al. [35,36] consider planning under incomplete domain models, where instead of possible precondition/effect annotations, the incomplete model is augmented with a set of successful plan traces. Unlike *PISA* these approaches cannot provide *a priori* guarantees on the robustness of the plan they generate. In Section 9, we discuss an extension of *PISA* (called *CPISA*), that is able to leverage the availability of successful plan traces, while still keeping robustness guarantees.

The work by Fox et al. [10] also explores robustness of plans, but their focus is on temporal plans under unforeseen execution-time variations rather than on incompletely specified domain models. Although there has been some work on reducing the "faults" in plan execution, e.g., the work on *k-fault* plans for non-deterministic planning [17], it is based in the context of stochastic/non-deterministic actions rather than incompletely specified ones. The semantics of the possible preconditions/effects in the incomplete domain models of this work differs fundamentally from non-deterministic and stochastic effects. Executing different instances of the same pick-up action in the *Gripper* example above would either all fail or all succeed, since there is no uncertainty but the information is unknown at the time the model is built. In contrast, if the pick-up action's effects are stochastic, then trying the same picking action multiple times increases the chances of success.

Our approaches to assessing plan robustness and synthesizing robust plans share the same spirit with the Probabilistic-FF planner [8] in utilizing the correctness constraints of a plan or a plan prefix, but are different in how they reflect the incompleteness semantics in deterministic domains. The constraints in our methods involve a unique set of boolean variables representing the realization of possible preconditions and effects of deterministic albeit incomplete actions, which can then be re-used across multiple steps of an action sequence. On the other hand, unknown effects in non-deterministic or stochastic domains are naturally much more costly to model; among other sets of boolean variables, the Probabilistic-FF planner introduces different variables for a same effect in different time steps. The encodings we construct, especially under the SE semantics where action failure implies plan failure, are therefore significantly more compact. Our heuristic search, described in Section 8, exploits certain structures of these compact encoding to approximate plan robustness, resulting in more efficient approach for plan synthesis under the SE semantics.

The possible precondition/effects to formalize domain model incompleteness has interesting applications in adapting planning technology to penetration testing in cybersecurity. In particular, as pointed out in Hoffmann [14], from a planning point of view, a realistic model involves assuming deterministic exploits whose exact preconditions and effects are nonetheless not completely known. Such a situation can be naturally modeled in terms of possible preconditions and effects. Interestingly, Hoffmann [14] proposes handling this situation in terms of stochastic actions. As discussed in the context of *k-fault* planning above, this will lead to the problematic semantics allowing for the action outcomes to change when it is executed repeatedly. Not surprisingly, the approach in Hoffmann [14] involves adding constraints to prohibit multiple executions of any single action. The robust planning framework based on possible precondition/effect annotations are, in our view, a more natural formulation.

In the context of decision-theoretic planning, a domain model is defined with a transition function mapping each pair of current state and action to a probability distribution of successor states. Much work has also been done for "uncertainty incompleteness", in particular with imprecise parameters representing incomplete transition probabilities. In Satia and Lave Jr [26], incomplete transition probabilities are modeled for each vector of probabilities representing a transition function from each pair of current state and action to a next state. They can be modeled either by a set (described with lower and upper bounds on component probabilities of the vector), or a prior distribution of probability values. For each of which, different optimal criteria were considered for quality of policies. Similar set-based representation and more efficient algorithm for computing max-min policy can also be found in White III and Eldeib [33]. With similar incompleteness representation, Givan et al. [12] considered bounded parameter Markov Decision Process (MDP) and propose a new value function representing values of states using closed real value intervals. The authors also devise algorithms for evaluating policies and computing optimal policies in this setting. Delgado et al. [6] introduce factored MDPs with similar incompleteness representation and propose efficient dynamic programming exploiting their structures.

While the focus of this paper is on planning with incomplete domain models, a closely related issue is learning to improve the completeness of the domain models. Reinforcement learning, and in particular the variant called bayesian reinforcement learning [5,28] aims to learn a posterior over the domain models, and plan (determine behavior) using this distribution over the models. This second phase bears several relations to planning with incomplete STRIPS models that we address in this paper (since, as we shall see, an incomplete annotated STRIPS model is a distribution over complete domain

models). Although we do not address integration of learning and acting that traditional reinforcement learning does, we do address some aspects of learning. In particular, *CPISA*, described in Section 9 can be viewed as starting with a distribution over domain models, and learning from past successful plan cases to compute a new posterior distribution over the domain models, which it then uses to compute robust plans for new problems. In contrast, RIM [36] is a competing approach that starts with a single domain model, and given successful plan cases, refines its initial model into a single new model. The comparison between RIM and *CPISA*, presented in Section 9.4, bears some relation to the advantages of normal vs. bayesian reinforcement learning.

This work can also be categorized as one particular instance of the general model-lite planning problem, as defined in Kambhampati [18], in which the author points out a large class of applications where handling incomplete models is unavoidable due to the difficulty in getting a complete model.

3. Problem formulation

We define an incomplete action model $\widetilde{\mathcal{D}}$ as $\widetilde{\mathcal{D}} = \langle \mathcal{R}, \mathcal{O} \rangle$, where $\mathcal{R}$ is a set of predicates with typed variables, $\mathcal{O}$ is a set of operators, each might be incompletely specified. In particular, in addition to the sets of known preconditions $Pre(o) \subseteq \mathcal{R}$, add effects $Add(o) \subseteq \mathcal{R}$ and delete effects $Del(o) \subseteq \mathcal{R}$, each operator $o \in \mathcal{O}$ also contains:

- possible precondition set $\widetilde{Pre}(o) \subseteq \mathcal{R}$ that operator o *might* need as its preconditions;
- possible add (delete) effect set $\widetilde{Add}(o) \subseteq \mathcal{R}$ ($\widetilde{Del}(o) \subseteq \mathcal{R}$) that o *might* add (delete, respectively) during execution.

In addition, each possible precondition, add and delete effect $r \in \mathcal{R}$ of an operator o is (optionally) associated with a "weight", or subjective probability, $w_o^{pre}(r)$, $w_o^{add}(r)$ and $w_o^{del}(r)$ ($0 < w_o^{pre}(r)$, $w_o^{add}(r)$, $w_o^{del}(r) < 1$) representing the domain modeler's assessment of the likelihood that r will actually be realized as a precondition, add and delete effect of o, respectively.[1] We assume that the "annotations" on possible preconditions and effects are uncorrelated, thus can be realized independently (both within each operator and across different ones).[2]

Given an incomplete domain model $\widetilde{\mathcal{D}}$, we define its *completion set* $\langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ as the set of complete domain models whose operators have all the known and "realized" preconditions and effects. Since any subset of $\widetilde{Pre}(o)$, $\widetilde{Add}(o)$ and $\widetilde{Del}(o)$ can be realized as preconditions and effects of o, there are 2^K possible complete domain models in $\langle\langle \widetilde{\mathcal{D}} \rangle\rangle$, where $K = \sum_{o \in \mathcal{O}} (|\widetilde{Pre}(o)| + |\widetilde{Add}(o)| + |\widetilde{Del}(o)|)$. There is exactly one (unknown) complete model, denoted by $\mathcal{D}^*$, that is the ground truth. For each complete model $\mathcal{D}$, we denote the complete sets of preconditions and effects for each operator o as $Pre^{\mathcal{D}}(o)$, $Add^{\mathcal{D}}(o)$ and $Del^{\mathcal{D}}(o)$.

A planning problem $\widetilde{\mathcal{P}}$ with respect to an incomplete domain model $\widetilde{\mathcal{D}}$ and a set of typed objects O is defined as $\widetilde{\mathcal{P}} = \langle F, A, I, G \rangle$ where F is the set of propositions instantiated from predicates $\mathcal{R}$ and objects O, A is the set of actions instantiated from $\mathcal{O}$ and O, $I \subseteq F$ is the initial state, and $G \subseteq F$ is the set of goal propositions. In a specific complete model $\mathcal{D}$, actions instantiated from one operator have the same realized preconditions and effects. The complete sets of preconditions and effects for action $a \in A$ in complete model $\mathcal{D}$ are denoted by $Pre^{\mathcal{D}}(a)$, $Add^{\mathcal{D}}(a)$ and $Del^{\mathcal{D}}(a)$.

A state is defined as either a set of propositions $s \subseteq F$ that are *true* (**T**), and all the remaining are *false* (**F**), or a special state $s_\perp = \{\perp\}$ where $\perp \notin F$ is a proposition used exclusively to define this "dead end" state. Since $s_\perp \not\subseteq F$, $Pre^{\mathcal{D}}(a) \not\subseteq s_\perp$, $(\forall a \in A, \mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle)$, and $s_\perp \not\models G$.

The resulting state after executing a sequence of actions π in a state s under a complete model $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ is denoted by $\gamma^{\mathcal{D}}(\pi, s)$. The projection of π in s with respect to the incomplete model $\widetilde{\mathcal{D}}$, $\gamma(\pi, s)$, is defined as the union of all projection of π from s with respect to each and every complete models in $\langle\langle \widetilde{\mathcal{D}} \rangle\rangle$:

$$\gamma(\pi, s) = \bigcup_{\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle} \gamma^{\mathcal{D}}(\pi, s). \tag{1}$$

The transition function $\gamma^{\mathcal{D}}(\pi, s)$ with respect to a complete model $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ is defined recursively as follows:

$$\gamma^{\mathcal{D}}(\pi, s) = \begin{cases} s & \text{if } \pi = \langle\rangle; \\ \gamma^{\mathcal{D}}(\langle a \rangle, \gamma^{\mathcal{D}}(\pi', s)) & \text{if } \pi = \pi' \circ \langle a \rangle. \end{cases} \tag{2}$$

In order to complete the definition of $\gamma(s, \pi)$, it is necessary to define $\gamma^{\mathcal{D}}(\langle a \rangle, s)$, the resulting state after applying action a in state s with respect to a complete model $\mathcal{D}$. Given that the domain model $\widetilde{\mathcal{D}}$ used during planning is incomplete, it is quite expected that some action of a synthesized plan might fail to be applied during execution. We consider two execution semantics with different ways in defining resulting states after executing an "inapplicable" action. In the first one, called *Generous execution* (GE) semantics, executing an action with unsatisfied preconditions does not change the world state. The transition function following this semantics is denoted with γ_{GE}, and the resulting state $\gamma_{GE}^{\mathcal{D}}(\langle a \rangle, s)$ is defined as follows:

[1] Possible preconditions and effects whose likelihood of realization is not given are assumed to have weights of $\frac{1}{2}$.

[2] While we cannot completely rule out a domain modeler capable of making annotations about correlated sources of incompleteness, we assume that this is less likely.

```
pick-up
:parameters (?b - ball ?r - room)
:precondition
    (and (at ?b ?r) (at-robot ?r) (free-gripper))
:possible_precondition
    (and (light ?b))
:effect
    (and (carry ?b)(not (at ?b ?r))(not (free-gripper)))
:possible_effect
    (and (dirty ?b))
```

Fig. 1. Description of the incomplete operator *pick-up(?b – ball,?r – room)* in the *Gripper* domain.

$$\gamma^{\mathcal{D}}_{GE}(\langle a\rangle, s) = \begin{cases} (s \setminus Del^{\mathcal{D}}(a)) \cup Add^{\mathcal{D}}(a) & \text{if } Pre^{\mathcal{D}}(a) \subseteq s; \\ s & \text{otherwise.} \end{cases} \tag{3}$$

The *STRIPS execution* (SE) semantics follows the definition in STRIPS-style planning [9], making the resulting state "undefined" if action a is not executable in state s—the plan is considered failed with respect to the complete model $\mathcal{D}$. The transition function $\gamma^{\mathcal{D}}_{SE}$ for a single action sequence with respect to a complete model $\mathcal{D}$ is defined for action $a \in A$ and state $s \in 2^F \cup \{s_\perp\}$ as follows:

$$\gamma^{\mathcal{D}}_{SE}(\langle a\rangle, s) = \begin{cases} (s \setminus Del^{\mathcal{D}}(a)) \cup Add^{\mathcal{D}}(a) & \text{if } Pre^{\mathcal{D}}(a) \subseteq s; \\ s_\perp & \text{otherwise.} \end{cases} \tag{4}$$

The definition of $\gamma(\pi, s)$ in Eq. (1) is now complete with the definitions of $\gamma^{\mathcal{D}}(\langle a\rangle, s)$ for the two semantics, $\gamma^{\mathcal{D}}_{GE}(\langle a\rangle, s)$ and $\gamma^{\mathcal{D}}_{SE}(\langle a\rangle, s)$. In the following, we use $\gamma(\pi, s)$, $\gamma^{\mathcal{D}}(\pi, s)$ in the discussion that applies for both semantics, and the notation with subscripts $\gamma_{GE}(\pi, s)$, $\gamma^{\mathcal{D}}_{GE}(\pi, s)$, $\gamma_{SE}(\pi, s)$, $\gamma^{\mathcal{D}}_{SE}(\pi, s)$ when the discussion limits to a particular execution semantics.

Given the transition function $\gamma(\pi, s)$, a sequence of actions π is a *valid plan* for $\widetilde{\mathcal{P}}$ if, after executing in the state I, it achieves the goals G with respect to at least one complete model: $\exists_{\mathcal{D}\in\langle\langle\widetilde{\mathcal{D}}\rangle\rangle} \gamma^{\mathcal{D}}(\pi, I) \models G$. Given that $\langle\langle\widetilde{\mathcal{D}}\rangle\rangle$ can be exponentially large in terms of possible preconditions and effects, validity is too weak to guarantee on the quality of plans. The quality of a plan, therefore, will be measured with its *robustness* value, which will be presented in the next section.

Example. Fig. 1 shows the description of incomplete action *pick-up(?b – ball,?r – room)* as described above. In addition to the possible precondition *(light ?b)* on the weight of the ball *?b*, we also assume that since the modeler is unsure if the gripper has been cleaned or not, she models it with a possible add effect *(dirty ?b)* indicating that the action might make the ball dirty. Those two possible preconditions and effects can be realized independently, resulting in four possible candidate complete domain models (assuming all other action schemes in the domain model are completely described).

4. A robustness measure for plans

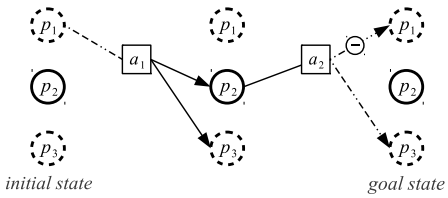
The robustness of a plan π for the problem $\widetilde{\mathcal{P}}$ is defined as the probability mass of the completions of $\widetilde{\mathcal{D}}$ under which π succeeds (in achieving the goals). More formally, let $Pr(\mathcal{D})$ be the modeler's estimate of the probability that a given model $\mathcal{D}$ in $\langle\langle\widetilde{\mathcal{D}}\rangle\rangle$ is the real model of the world ($0 < Pr(\mathcal{D}) < 1, \forall \mathcal{D} \in \langle\langle\widetilde{\mathcal{D}}\rangle\rangle$; $\sum_{\mathcal{D}\in\langle\langle\widetilde{\mathcal{D}}\rangle\rangle} Pr(\mathcal{D}) = 1$). The robustness of π is defined as follows:

$$R(\pi) \stackrel{def}{\equiv} \sum_{\mathcal{D}\in\langle\langle\widetilde{\mathcal{D}}\rangle\rangle, \gamma^{\mathcal{D}}(\pi, I)\models G} Pr(\mathcal{D}) \tag{5}$$

Note that given the uncorrelated incompleteness assumption, the probability $Pr(\mathcal{D})$ for a model $\mathcal{D} \in \langle\langle\widetilde{\mathcal{D}}\rangle\rangle$ can be computed as the product of the weights $w^{pre}_o(r)$, $w^{add}_o(r)$, and $w^{del}_o(r)$ for all $o \in \mathcal{O}$ and its possible preconditions/effects r if r *is* realized as a precondition, add and delete effect of the operator in $\mathcal{D}$ (or the product of their "complement" $1 - w^{pre}_o(r)$, $1 - w^{add}_o(r)$, and $1 - w^{del}_o(r)$ if r is *not* realized).

It is easy to see that if $R(\pi) > 0$, then π is a valid plan for $\widetilde{\mathcal{P}}$. The definition of plan robustness introduced here applies for both two execution semantics. The robustness of a given plan under the SE semantics is no greater than it is under the GE semantics—any plan that succeeds under SE semantics does so under the GE semantics.

Example. Fig. 2 shows an example with an incomplete domain model $\widetilde{\mathcal{D}} = \langle \mathcal{R} \equiv F, \mathcal{O} \equiv A\rangle$ with $F = \{p_1, p_2, p_3\}$ and $A = \{a_1, a_2\}$ and a solution plan $\pi = \langle a_1, a_2\rangle$ for the problem $\widetilde{\mathcal{P}} = \langle F, \underset{\sim}{A}, I = \{p_2\}, G = \{p_3\}\rangle$. The incomplete model is: $Pre(\underset{\sim}{a_1}) = \emptyset$, $\widetilde{Pre}(a_1) = \{p_1\}$, $Add(\underset{\sim}{a_1}) = \{p_2, p_3\}$, $\widetilde{Add}(a_1) = \emptyset$, $Del(a_1) = \emptyset$, $\widetilde{Del}(a_1) = \emptyset$; $Pre(a_2) = \{p_2\}$, $\widetilde{Pre}(a_2) = \emptyset$, $Add(a_2) = \emptyset$, $\widetilde{Add}(a_2) = \{p_3\}$, $Del(a_2) = \emptyset$, $\widetilde{Del}(a_2) = \{p_1\}$. Given that the total number of possible preconditions and effects is 3, the total number of completions ($|\langle\langle\widetilde{\mathcal{D}}\rangle\rangle|$) is $2^3 = 8$, for each of which the plan π may succeed or fail to achieve G, as shown in the



Candidate models	1	2	3	4	5	6	7	8
a_1 relies on p_1	yes	yes	yes	yes	no	no	no	no
a_2 adds p_3	yes	yes	no	no	yes	yes	no	no
a_2 deletes p_1	yes	no	yes	no	yes	no	yes	no
Plan status – GE semantics	succeed	succeed	fail	fail	succeed	succeed	succeed	succeed
Plan status – SE semantics	fail	fail	fail	fail	succeed	succeed	succeed	succeed

Fig. 2. Example for a set of complete candidate domain models, and the corresponding plan status under two semantics. Circles with solid and dash boundary respectively are propositions that are known to be **T** and might be **F** when the plan executes (see more in text).

table. In the fifth candidate model, for instance, p_1 and p_3 are realized as precondition and add effect of a_1 and a_2, whereas p_1 is not a delete effect of action a_2. Even though a_1 could not execute (and thus p_3 remains *false* in the second state), the goal eventually is achieved according to the GE semantics by action a_2 with respects to this candidate model. Overall, with the GE semantics there are two of eight candidate models where π fails and six for which it succeeds. The robustness value of the plan is $R(\pi) = \frac{3}{4}$ if $Pr(\mathcal{D}_i)$ is uniform. However, if the domain writer thinks that p_1 is very likely to be a precondition of a_1 and provides $w^{pre}_{a_1}(p_1) = 0.9$, the robustness of π decreases to $R(\pi) = 2 \times (0.9 \times 0.5 \times 0.5) + 4 \times (0.1 \times 0.5 \times 0.5) = 0.55$ (as intuitively, the last four models with which π succeeds are very unlikely to be the real one). Note that according to the SE semantics, the plan π would be considered failing to achieve G in the first four complete models since a_1 fails to apply (and thus a_2 is prevented from execution).

5. A spectrum of robust planning problems

Given this set up, we can now talk about a spectrum of problems related to planning under incomplete domain models:

Robustness Assessment (RA): Given a plan π for the problem $\widetilde{\mathcal{P}}$, assess the robustness of π.
Maximally Robust Plan Generation (RG):* Given a problem $\widetilde{\mathcal{P}}$, generate the maximally robust plan π^*.
Generating Plan with Desired Level of Robustness (RG$^\rho$): Given a problem $\widetilde{\mathcal{P}}$ and a robustness threshold ρ ($0 < \rho \leq 1$), generate a plan π with robustness greater than or equal to ρ.
Cost-sensitive Robust Plan Generation (RG$^*_c$): Given a problem $\widetilde{\mathcal{P}}$ and a cost bound c, generate a plan π of maximal robustness subject to cost bound c (where the cost of a plan π is defined as the cumulative costs of the actions in π).
Incremental Robustification (RI$_c$): Given a plan π for the problem $\widetilde{\mathcal{P}}$, improve the robustness of π, subject to a cost budget c.

The problem of assessing robustness of plans, RA, can be tackled by compiling it into a weighted model-counting problem. We will also show in Section 6 that RA with uniform distribution of candidate complete models is complete for #P complexity class [29], and thus the robustness assessment problem is at least as hard as NP-complete.

For plan synthesis problems, we can talk about either generating a maximally robust plan, RG*, or finding a plan with a robustness value above the given threshold, RG$^\rho$. A related issue is that of the interaction between plan cost and robustness. Increasing robustness might involve using additional or costlier actions to support the desired goals, and thus comes at the expense of increased plan cost. We can also talk about cost-constrained robust plan generation, RG$^*_c$. Finally, in practice, we are often interested in increasing the robustness of a given plan (either during iterative search, or during mixed-initiative planning). We thus also have the incremental variant RI$_c$. In the next sections, we will focus on the problems of assessing plan robustness and synthesizing robust plans, ignoring plan cost.

6. Assessing plan robustness

Given an incomplete domain model $\widetilde{\mathcal{D}}$, problem $\widetilde{\mathcal{P}}$ and plan $\pi = \langle a_1, ..., a_n \rangle$, a naive approach to compute the robustness of π is to enumerate all domain models $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ and check for the success of π with respect to $\mathcal{D}$. This is prohibitively expensive when the number of possible preconditions and effects is large. In this section, we show that the problem of assessing plan robustness can be reduced to the weighted model counting problem (cf. Sang et al. [25]) in which a set of logical constraints Σ_π is constructed such that there is a one-to-one mapping between each of its models and a candidate domain model under which the plan succeeds. In the rest of this section, we assume $a_0 \equiv a_I$ and $a_{n+1} \equiv a_G$ for any sequence π of length n, where a_I and a_G are two dummy *complete* actions representing the initial and goal state: $Pre(a_I) = \emptyset$, $Add(a_I) = I$, $Pre(a_G) = G$, $Add(a_G) = \{\top\}$, where $\top \notin F$ denotes a dummy proposition representing goal achievement (those

precondition and effect sets not specified are empty). We also denote $\langle s_0 \equiv I, s_1, ..., s_n, s_{n+1} \equiv \{\top\}\rangle$ as the sequence of states generated from the execution of π.

6.1. GE semantics

Boolean variables: First, we create variables representing whether a possible precondition or effect of an operator $o \in \mathcal{O}$ is realized: r_o^{pre}, r_o^{add} and r_o^{del} with the associated weights $w_o^{pre}(r)$, $w_o^{add}(r)$ and $w_o^{del}(r)$ for each predicate r respectively in $\widetilde{Pre}(o)$, $\widetilde{Add}(o)$ and $\widetilde{Del}(o)$. Abusing notation, we often write p_a^{pre} and $w_a^{pre}(p)$ for action $a \in A$, $p \in \widetilde{Pre}(a)$, and similarly for $p \in \widetilde{Add}(a)$ or $\widetilde{Del}(a)$, to refer to the boolean variables and weights defined for the unique predicate and operator from which they are instantiated. We denote $\widetilde{Z}$ as the set of those boolean variables.

Second, for each action instance a_i in the solution plan π we create one variable with weight $\frac{1}{2}$ representing whether the action can execute or not. Abusing notation, we denote this boolean variable a_i.

Finally, for each proposition $p \in F$ and state index $i \in \{0, 1, ..., n\}$, we create a boolean variable p_i with weight $\frac{1}{2}$.

We denote Z as the set of all boolean variables defined; thus $Z \setminus \widetilde{Z}$ is the set of those representing action instances a_i's and propositions p_i's over time steps.

Constraints:

Initial and goal states constraints: for each proposition p that presents in the initial state, we set $p_0 = \mathbf{T}$; otherwise, $p_0 = \mathbf{F}$. Similarly, if $p \in G$ then $p_n = \mathbf{T}$.

Action execution constraints: for each action a_i $(1 \le i \le n+1)$, we create a constraint:

$$a_i \Leftrightarrow exe(a_i), \tag{6}$$

specifying the necessary and sufficient conditions under which the action can and will execute:

$$exe(a_i) \equiv \bigwedge_{p \in Pre(a_i)} p_{i-1} \wedge \bigwedge_{p \in \widetilde{Pre}(a_i)} (p_{a_i}^{pre} \Rightarrow p_{i-1}). \tag{7}$$

We note that, slightly different from a state-based encoding for plan synthesis (cf., Rintanen et al. [23]), we need $a_i \Leftarrow exe(a_i)$ to correctly capture our semantics of action execution: when an action is executable (i.e., when all of its preconditions, including realized ones, are satisfied), its effects will take place in the successor state.

Effect constraints: we create constraints specifying that if action a_i executes, then its effects are true in the next state:

$$a_i \Rightarrow \bigwedge_{p \in Add(a_i)} p_i \wedge \bigwedge_{p \in Del(a_i)} \neg p_i \wedge \bigwedge_{p \in \widetilde{Add}(a_i)} (p_{a_i}^{add} \Rightarrow p_i) \wedge \bigwedge_{p \in \widetilde{Del}(a_i)} (p_{a_i}^{del} \Rightarrow \neg p_i). \tag{8}$$

State change constraints: for every proposition $p \in F$, we add constraints specifying conditions under which the value of p being changed in two consecutive time steps $(i-1)$-th and i-th $(1 \le i \le n)$. For the changes from **F** to **T**:

$$\neg p^{i-1} \wedge p^i \Rightarrow a_i \wedge \phi_i, \tag{9}$$

in which ϕ_i is defined as follows:

$$\phi_i = \begin{cases} \mathbf{T} & \text{if } p \in Add(a_i); \\ p_{a_i}^{add} & \text{if } p \in \widetilde{Add}(a_i); \\ \mathbf{F} & \text{otherwise.} \end{cases} \tag{10}$$

The constraints for changing from **T** to **F** are defined similarly.

Let Σ_π be the set of all constraints above. For each assignment σ for variables Z, we denote $\mathcal{D}_\sigma \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ as the candidate domain model in which the realization of possible preconditions and effects is determined by the truth assignment for variables $\widetilde{Z}$ in σ. Let $\mathcal{M}(\Sigma_\pi)$ be the set of all models satisfying Σ_π.

Theorem 1. *For each* $\sigma \in \mathcal{M}(\Sigma_\pi)$, $\gamma_{GE}^{\mathcal{D}_\sigma}(\pi, I) \models G$.

Proof. Each model $\sigma \in \mathcal{M}(\Sigma_\pi)$ assigns truth values to variables $\widetilde{Z}$. With those truth values, all constraints in which variables in $\widetilde{Z}$ are involved are similar to those for synthesizing plans with SAT-based approach (for instance, in Rintanen et al. [23]), except that there is only one action instance of π at each time step. Following the correctness of encoding for those SAT-based approaches, π must achieve G with respect to the model σ. □

The following theorem establishes the connection between the robustness of π with the weighted model count of Σ_π.

Theorem 2. *The robustness of π in the GE semantics is*

$$R(\pi) = 2^{|Z \setminus \widetilde{Z}|} \times WMC(\Sigma_\pi),$$

where $WMC(\Sigma_\pi)$ is the weighted model count of Σ_π.

Proof. Let $w(\sigma)$ be the weight of an assignment σ. We have:

$$w(\sigma) = (\frac{1}{2})^{|Z \setminus \widetilde{Z}|} \times Pr(\mathcal{D}_\sigma).$$

By definition:

$$\begin{aligned} \text{WMC}(\Sigma_\pi) &= \sum_{\sigma \in \mathcal{M}(\Sigma_\pi)} w(\sigma) \\ &= (\frac{1}{2})^{|Z \setminus \widetilde{Z}|} \times \sum_{\sigma \in \mathcal{M}(\Sigma_\pi)} Pr(\mathcal{D}_\sigma) \\ &= (\frac{1}{2})^{|Z \setminus \widetilde{Z}|} \times R(\pi). \end{aligned}$$

The last equality is due to both Theorem 1 and the definition of plan robustness. □

6.2. SE semantics

Variables: The variables are the same as those in the set of variables $\widetilde{Z}$ used in the correctness constraints under the GE semantics.

Constraints: Given that all actions must be executable, and that the initial state at the first step is complete, the truth value of any proposition p at level i ($1 < i \leq n+1$) can only be affected by actions at steps $k \in \{C_p^i, ..., i-1\}$. Here, $C_p^i \in \{1, ..., i-1\}$ is the latest level before i at which the truth value of p at C_p^i is completely "confirmed" by the success of either action $a_{C_p^i}$ or $a_{C_p^i - 1}$. Specifically, it is confirmed **T** if $p \in Pre(a_{C_p^i})$ or $p \in Add(a_{C_p^i - 1})$; and confirmed **F** if $p \in Del(a_{C_p^i - 1})$.

Precondition establishment and protection: for each $p \in Pre(a_i)$ ($1 \leq i \leq n+1$), we create constraints establishing and protecting the **T** value of this precondition. If $p = \mathbf{F}$ at level C_p^i, we add the following constraints to ensure that it is supported before level i:

$$\bigvee_{C_p^i \leq k \leq i-1, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}. \tag{11}$$

Note that there exists at least one such action a_k for a_i to be executable. If there exists actions a_m ($C_p^i \leq m \leq i-1$) that possibly deletes p, we protect its value with the constraints:

$$p_{a_m}^{del} \Rightarrow \bigvee_{m < k < i, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}, \tag{12}$$

or $\neg p_{a_m}^{del}$ if there is no such action a_k possibly support p. Note that the constraints for known preconditions of a_{n+1} ensure that goals G are achieved after the plan execution.

Possible precondition establishment and protection: when a possible precondition p of action a_i ($1 \leq i \leq n$) is realized, its value also needs to be established to **T** and protected. Specifically, for $p = \mathbf{F}$ at level C_p^i, we add the constraints:

$$p_{a_i}^{pre} \Rightarrow \bigvee_{C_p^i \leq k \leq i-1, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}. \tag{13}$$

Finally, with actions a_m ($C_p^i \leq m \leq i-1$) having p as a possible delete effect, we must ensure that:

$$p_{a_i}^{pre} \Rightarrow \left(p_{a_m}^{del} \Rightarrow \bigvee_{m < k < i, p \in \widetilde{Add}(a_k)} p_{a_k}^{add} \right). \tag{14}$$

We again denote the set of constraints (11)–(14) established for π as Σ_π. It can be shown that any assignment to Boolean variables p_a^{pre}, p_a^{add} and p_a^{del} satisfying all constraints above corresponds to a complete domain model $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ under which all actions a_1, ..., a_n and a_{n+1} succeeds to execute. The weighted model count of Σ_π is thus the robustness of the plan. We will therefore use $R(\pi)$ and $\text{WMC}(\Sigma_\pi)$ interchangeably under the SE semantics.

6.3. Computational complexity

The fact that weighted model counting can be used to compute plan robustness does not immediately mean that assessing plan robustness is hard. We now show that it is indeed #P-complete, the same complexity of model counting problems.

Theorem 3. *Given an incomplete model $\widetilde{\mathcal{D}} = \langle \mathcal{R}, \mathcal{O} \rangle$ with annotations of weights $\frac{1}{2}$, a planning problem $\widetilde{\mathcal{P}} = \langle F, A, I, G \rangle$ and a plan π. The problem of computing $R(\pi)$ is #P-complete, and this holds for both GE and SE semantics.*

Proof. The following proof applies for both two semantics. The membership can be seen by having a Counting Turing Machine [29] nondeterministically guess a complete model, and check the correctness of the plan. The number of accepting branches output is the number of complete models under which the plan succeeds.

To prove the hardness, we show that there is a polynomial-time reduction from an instance $\langle X, \Sigma \rangle$ of MONOTONE 2-SAT, a #P-complete problem [30], to an instance $\langle \widetilde{\mathcal{D}}, \widetilde{\mathcal{P}}, \pi \rangle$ such that $R(\pi) = \frac{|\mathcal{M}(\Sigma)|}{2^n}$. The input of the MONOTONE 2-SAT counting problem is a set of n Boolean variables $X = \{x_1, ..., x_n\}$, a monotone CNF formulae $\Sigma = \{c_1, c_2, ..., c_m\}$ with m clauses $c_j = x_{j1} \vee x_{j2}$, where $x_{j1}, x_{j2} \in X$. The required output is $|\mathcal{M}(\Sigma)|$, the number of assignments of X satisfying Σ.

Let $\mathcal{G} = \langle V, E \rangle$ be the constraint graph of the clause set Σ, where $V = X$ and $(x_i, x_j) \in E$ if $x_i \vee x_j \in \Sigma$. We partition this graph into connected components (or subgraph) $\mathcal{G}_1, ..., \mathcal{G}_l$. Our set of propositions F then includes propositions p_k for each subgraph $\mathcal{G}_k$ $(1 \leq k \leq l)$ and propositions g_j for each clause c_j $(1 \leq j \leq m)$: $F = \{p_1, ..., p_l, g_1, ..., g_m\}$. We denote $k(x_i)$ and $k(c_j)$ as the indices of the subgraphs containing x_i and the edge (x_{j1}, x_{j2}). The set of actions A consists of $n + m$ actions a_{x_i} and b_{g_j}, respectively defined for each variable x_i and proposition g_j. Action a_{x_i} has $\widetilde{Add}(a_{x_i}) = \{p_{k(x_i)}\}$ and the other components are empty. Action b_{g_j} is complete and defined with $Pre(b_{g_j}) = Del(b_{g_j}) = \{p_{k(c_j)}\}$, $Add(b_{g_j}) = \{g_j\}$. Given $\widetilde{\mathcal{D}} = \langle F, A \rangle$,[3] we define a planning problem $\widetilde{\mathcal{P}}$ with $I = \emptyset$, $G = \{g_1, ..., g_m\}$ and a plan π that is the concatenation of m "subplans": $\pi = \pi_1 \circ ... \circ \pi_m$. Each subplan is $\pi_j = \langle a_{x_{j1}}, a_{x_{j2}}, b_{g_j} \rangle$ $(1 \leq j \leq m)$ such that $c_j = x_{j1} \vee x_{j2}$ is the clause corresponding to g_j.

From the reduction, there is a one-to-one mapping between the assignments of X to $\langle\langle \widetilde{\mathcal{D}} \rangle\rangle$: $x_i = \mathbf{T}$ if and only if a_{x_i} has $p_{k(x_i)}$ as its add effect. The relation $R(\pi) = \frac{|\mathcal{M}(\Sigma)|}{2^n}$ can be established by verifying that an assignment σ satisfies Σ if and only if π succeeds under the corresponding complete model $\mathcal{D}_\sigma$.

Consider an assignment σ satisfying Σ, and the corresponding complete model $\mathcal{D}_\sigma \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$. For each clause $c_j = x_{j1} \vee x_{j2}$, at least one of the variables x_{j1} and x_{j2} must be $\mathbf{T}$, implying that one of the two actions $a_{x_{j1}}$ and $a_{x_{j2}}$ will support $p_{k(c_j)}$ under $\mathcal{D}_\sigma$. Action b_{g_j} is therefore executable in π, achieving g_j. As a result, π succeeds in achieving G under $\mathcal{D}_\sigma$. Vice versa, assuming that π succeeds under $\mathcal{D}_\sigma$. Since each g_j can only be supported by the corresponding b_{g_j}, these actions must be executable. Therefore, $p_{k(c_j)}$ must be realized as an add effect of either $a_{x_{j1}}$ or $a_{x_{j2}}$, implying that $c_j = x_{j1} \vee x_{j2}$ is satisfiable. The set of clauses Σ must then be satisfiable with respect to σ. The relation $R(\pi) = \frac{|\mathcal{M}(\Sigma)|}{2^n}$ can now be established. □

7. Synthesizing robust plans with a compilation approach

In this section we will show that the problem of generating plans with at least ρ robustness value can be compiled into a conformant probabilistic planning problem [8]. We focus the details only on the GE semantics in this section and then present our experimental results using Probabilistic-FF, a state-of-the-art planner created by Domshlak and Hoffmann [8]. We briefly discuss how to adapt the compilation presented for the SE semantics.

7.1. Conformant probabilistic planning

Following the formalism used by Domshlak and Hoffmann [8], a domain model in conformant probabilistic planning (CPP) is a tuple $\mathcal{D}' = \langle F', A' \rangle$, where F' and A' are the sets of propositions and probabilistic actions, respectively. A belief state $b : 2^{F'} \rightarrow [0, 1]$ is a distribution of states $s \subseteq F'$ (we denote $s \in b$ if $b(s) > 0$). Each action $a' \in A'$ is specified by a set of preconditions $Pre(a') \subseteq F'$ and conditional effects $E(a')$. For each $e = (cons(e), \mathcal{O}(e)) \in E(a')$, $cons(e) \subseteq F'$ is the condition set and $\mathcal{O}(e)$ determines the set of outcomes $\varepsilon = (Pr(\varepsilon), add(\varepsilon), del(\varepsilon))$ that will add and delete proposition sets $add(\varepsilon)$, $del(\varepsilon)$ into and from the resulting state with the probability $Pr(\varepsilon)$ $(0 \leq Pr(\varepsilon) \leq 1$, $\sum_{\varepsilon \in \mathcal{O}(e)} Pr(\varepsilon) = 1)$. All condition sets of the effects in $E(a')$ are assumed to be mutually exclusive and exhaustive. The action a' is applicable in a belief state b if $Pre(a') \subseteq s$ for all $s \in b$, and the probability of a state s' in the resulting belief state is $b_{a'}(s') = \sum_{s \supseteq Pre(a')} b(s) \sum_{\varepsilon \in \mathcal{O}'(e)} Pr(\varepsilon)$, where $e \in E(a')$ is the conditional effect such that $cons(e) \subseteq s$, and $\mathcal{O}'(e) \subseteq \mathcal{O}(e)$ is the set of outcomes ε such that $s' = s \cup add(\varepsilon) \setminus del(\varepsilon)$.

Given the domain model $\mathcal{D}'$, a problem $\mathcal{P}'$ is a quadruple $\mathcal{P}' = \langle \mathcal{D}', b_I, G', \rho' \rangle$, where b_I is an initial belief state, G' is a set of goal propositions and ρ' is the acceptable goal satisfaction probability. A sequence of actions $\pi' = \langle a'_1, ..., a'_n \rangle$ is a

[3] The resulting robustness assessment problem does not have objects, thus $\mathcal{R} \equiv F$ and $\mathcal{O} \equiv A$.

solution plan for $\mathcal{P}'$ if a'_i is applicable in the belief state b_i (assuming $b_1 \equiv b_I$), which results in b_{i+1} $(1 \leq i \leq n)$, and it achieves all goal propositions with at least ρ' probability.

7.2. Compilation

Given an incomplete domain model $\widetilde{\mathcal{D}} = \langle \mathcal{R}, \mathcal{O} \rangle$ and a planning problem $\widetilde{\mathcal{P}} = \langle F, A, I, G \rangle$, we now describe a compilation that translates the problem of synthesizing a solution plan π for $\widetilde{\mathcal{P}}$ such that $R(\pi) \geq \rho$ to a CPP problem $\mathcal{P}'$. At a high level, the realization of possible preconditions $p \in \widetilde{Pre}(a)$ and effects $q \in \widetilde{Add}(a)$, $r \in \widetilde{Del}(a)$ of an action $a \in A$ can be understood as being determined by the truth values of *hidden* propositions p_a^{pre}, q_a^{add} and r_a^{del} that are certain (i.e. unchanged in any world state) but unknown. (These propositions play the same role with variables $\widetilde{Z}$ in Section 6.1.) Specifically, the applicability of the action in a state $s \subseteq F$ depends on possible preconditions p that are realized (i.e. $p_a^{pre} = \mathbf{T}$), and their truth values in s. Similarly, the values of q and r are affected by a in the resulting state only if they are realized as add and delete effects of the action (i.e., $q_a^{add} = \mathbf{T}$, $r_a^{del} = \mathbf{T}$). There are totally $2^{|\widetilde{Pre}(a)|+|\widetilde{Add}(a)|+|\widetilde{Del}(a)|}$ realizations of the action a, and all of them should be considered simultaneously in checking the applicability of the action and in defining corresponding resulting states.

With those observations, we use multiple conditional effects to compile away incomplete knowledge on preconditions and effects of the action a. Each conditional effect corresponds to one realization of the action, and can be fired only if $p = \mathbf{T}$ whenever $p_a^{pre} = \mathbf{T}$, and adding (removing) an effect q (r) into (from) the resulting state depending on the values of q_a^{add} (r_a^{del}, respectively) in the realization.

While the partial knowledge can be removed, the hidden propositions introduce uncertainty into the initial state, and therefore making it a *belief* state. Since the action a may be applicable in some but rarely all states of a belief state, *certain* preconditions $Pre(a)$ should be modeled as conditions of all conditional effects. We are now ready to formally specify the resulting domain model $\mathcal{D}'$ and problem $\mathcal{P}'$.

For each action $a \in A$, we introduce new propositions p_a^{pre}, q_a^{add}, r_a^{del} and their negations np_a^{pre}, nq_a^{add}, nr_a^{del} for each $p \in \widetilde{Pre}(a)$, $q \in \widetilde{Add}(a)$ and $r \in \widetilde{Del}(a)$ to determine whether they are realized as preconditions and effects of a in the real domain model.[4] Let F_{new} be the set of those new propositions, then $F' = F \cup F_{new}$ is the proposition set of $\mathcal{D}'$.

Each action $a' \in A'$ is made from one action $a \in A$ such that $Pre(a') = \emptyset$, and $E(a')$ consists of $2^{|\widetilde{Pre}(a)|+|\widetilde{Add}(a)|+|\widetilde{Del}(a)|}$ conditional effects e. For each conditional effect e:

- $cons(e)$ is the union of the following sets:
  - the certain preconditions $Pre(a)$,
  - the set of possible preconditions of a that are realized, and hidden propositions representing their realization: $\overline{Pre}(a) \cup \{p_a^{pre} | p \in \overline{Pre}(a)\} \cup \{np_a^{pre} | p \in \widetilde{Pre}(a) \setminus \overline{Pre}(a)\}$,
  - the set of hidden propositions corresponding to the realization of possible add (delete) effects of a: $\{q_a^{add} | q \in \overline{Add}(a)\} \cup \{nq_a^{add} | q \in \widetilde{Add}(a) \setminus \overline{Add}(a)\}$ ($\{r_a^{del} | r \in \overline{Del}(a)\} \cup \{nr_a^{del} | r \in \widetilde{Del}(a) \setminus \overline{Del}(a)\}$, respectively);
- the *single* outcome ε of e is defined as $add(\varepsilon) = Add(a) \cup \overline{Add}(a)$, $del(\varepsilon) = Del(a) \cup \overline{Del}(a)$, and $Pr(\varepsilon) = 1$,

where $\overline{Pre}(a) \subseteq \widetilde{Pre}(a)$, $\overline{Add}(a) \subseteq \widetilde{Add}(a)$ and $\overline{Del}(a) \subseteq \widetilde{Del}(a)$ represent the sets of realized preconditions and effects of the action. In other words, we create a conditional effect for each subset of the union of the possible precondition and effect sets of the action a. Note that the inclusion of new propositions derived from $\overline{Pre}(a)$, $\overline{Add}(a)$, $\overline{Del}(a)$ and their "complement" sets $\widetilde{Pre}(a) \setminus \overline{Pre}(a)$, $\widetilde{Add}(a) \setminus \overline{Add}(a)$, $\widetilde{Del}(a) \setminus \overline{Del}(a)$ makes all condition sets of the action a' mutually exclusive. As for other cases (including those in which some precondition in $Pre(a)$ is excluded), the action has no effect on the resulting state, they can be ignored. The condition sets, therefore, are also exhaustive.

The initial belief state b_I consists of $2^{|F_{new}|}$ states $s' \subseteq F'$ such that $p \in s'$ iff $p \in I$ $(\forall p \in F)$, each represents a complete domain model $\mathcal{D}_i \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ and with the probability $Pr(\mathcal{D}_i)$. The goal is $G' = G$, and the acceptable goal satisfaction probability is $\rho' = \rho$.

Theorem 4. *Given a plan $\pi = \langle a_1, ..., a_n \rangle$ for the problem $\widetilde{\mathcal{P}}$, and $\pi' = (a'_1, ..., a'_n)$ where a'_k is the compiled version of a_k $(1 \leq k \leq n)$ in $\mathcal{P}'$. Then $R(\pi) \geq \rho$ iff π' achieves all goals with at least ρ probability in $\mathcal{P}'$.*

Proof. According to the compilation, there is one-to-one mapping between each complete model $\mathcal{D}_i \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ in $\widetilde{\mathcal{P}}$ and a (complete) state $s'_{i0} \in b_I$ in $\mathcal{P}'$. Moreover, if $\mathcal{D}_i$ has a probability of $Pr(\mathcal{D}_i)$ to be the real model, then s'_{i0} also has a probability of $Pr(\mathcal{D}_i)$ in the belief state b_I of $\mathcal{P}'$.

Given the projection over complete model $\mathcal{D}_i$, executing π from the state I with respect to $\mathcal{D}_i$ results in a sequence of complete states $\langle s_{i1}, ..., s_{i(n+1)} \rangle$. On the other hand, executing π' from $\{s'_{i0}\}$ in $\mathcal{P}'$ results in a sequence of belief states $(\{s'_{i1}\}, ..., \{s'_{i(n+1)}\})$. Since $p \in s'_{i0}$ iff $p \in I$ $(\forall p \in F)$, by induction it can be shown that $p \in s'_{ij}$ iff $p \in s_{ij}$ $(\forall j \in \{1, ..., n+1\}$, $p \in F)$. Therefore, $s_{i(n+1)} \models G$ iff $s'_{i(n+1)} \models G = G'$.

[4] These propositions are introduced once, and re-used for all actions instantiated from the same operator with a.

```
pick-up
:parameters (?b - ball ?r - room)
:precondition
    (and (at ?b ?r) (at-robot ?r) (free-gripper))
:possible_precondition
    (and (light ?b))
:effect
    (and (carry ?b)(not (at ?b ?r))(not (free-gripper)))
:possible_effect
    (and (dirty ?b))
```

```
pick-up
:parameters (?b - ball ?r - room)
:precondition (and )
:effect (and
  (when (and (at ?b ?r) (at-robot ?r) (free-gripper)
             (light ?b) ( p^pre_pick-up ) ( q^add_pick-up ))
        (and (carry ?b)(not (at ?b ?r))(not (free-gripper))
             (dirty ?b)))
  (when (and (at ?b ?r) (at-robot ?r) (free-gripper)
             (light ?b) ( p^pre_pick-up ) ( nq^add_pick-up ))
        (and (carry ?b)(not (at ?b ?r))(not (free-gripper))))
  (when (and (at ?b ?r) (at-robot ?r) (free-gripper)
             ( np^pre_pick-up ) ( q^add_pick-up ))
        (and (carry ?b)(not (at ?b ?r))(not (free-gripper))
             (dirty ?b)))
  (when (and (at ?b ?r) (at-robot ?r) (free-gripper)
             ( np^pre_pick-up ) ( nq^add_pick-up ))
        (and (carry ?b)(not (at ?b ?r))(not (free-gripper)))))
```

Fig. 3. An example of compiling the action *pick-up* in an incomplete domain model (top) into CPP domain (bottom). The hidden propositions $p^{pre}_{pick\text{-}up}$, $q^{add}_{pick\text{-}up}$ and their negations can be interpreted as whether the action requires light balls and makes balls dirty. Newly introduced and relevant propositions are marked in bold.

Since all actions a'_i are deterministic and s'_{i0} has a probability of $Pr(\mathcal{D}_i)$ in the belief state b_I of $\mathcal{P}'$, the probability that π' achieves G' is $\sum_{s'_{i(n+1)} \models G} Pr(\mathcal{D}_i)$, which is equal to $R(\pi)$ as defined in Equation (5). This proves the theorem. □

Example. Consider the action *pick-up(?b – ball,?r – room)* in the Gripper domain as described above. In addition to the possible precondition *(light ?b)* on the weight of the ball *?b*, we also assume that since the modeler is unsure if the gripper has been cleaned or not, she models it with a possible add effect *(dirty ?b)* indicating that the action might make the ball dirty. Fig. 3 shows both the original and the compiled specification of the action.

7.3. Experimental results

We tested the compilation with Probabilistic-FF (PFF), a state-of-the-art planner, on a range of domains in the International Planning Competition.We first discuss the results on the variants of the Logistics and Satellite domains, where domain incompleteness is deliberately modeled on the preconditions and effects of actions (respectively). Our purpose here is to observe how generated plans are robustified to satisfy a given robustness threshold, and how the amount of incompleteness in the domains affects the plan generation phase. We then describe the second experimental setting in which we randomly introduce incompleteness into IPC domains, and discuss the feasibility of our approach in this setting.[5]

Domains with deliberate incompleteness

Logistics: In this domain, each of the two cities C_1 and C_2 has an airport and a downtown area. The transportation between the two distant cities can only be done by two airplanes A_1 and A_2. In the downtown area of C_i ($i \in \{1, 2\}$), there are three *heavy* containers $P_{i1}, ..., P_{i3}$ that can be moved to the airport by a truck T_i. Loading those containers onto the truck in the city C_i, however, requires moving a team of m robots $R_{i1}, ..., R_{im}$ ($m \geq 1$), initially located in the airport, to the downtown

[5] The experiments were conducted using an Intel Core2 Duo 3.16 GHz machine with 4 Gb of RAM, and the time limit is 15 minutes.

Table 1
The results of generating robust plans in Logistics domain (see text).

ρ	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
0.1	32/10.9	36/26.2	40/57.8	44/121.8	48/245.6
0.2	32/10.9	36/25.9	40/57.8	44/121.8	48/245.6
0.3	32/10.9	36/26.2	40/57.7	44/122.2	48/245.6
0.4	$\perp$	42/42.1	50/107.9	58/252.8	66/551.4
0.5	$\perp$	42/42.0	50/107.9	58/253.1	66/551.1
0.6	$\perp$	$\perp$	50/108.2	58/252.8	66/551.1
0.7	$\perp$	$\perp$	$\perp$	58/253.1	66/551.6
0.8	$\perp$	$\perp$	$\perp$	$\perp$	66/550.9
0.9	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Table 2
The results of generating robust plans in Satellite domain (see text).

ρ	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
0.1	10/0.1	10/0.1	10/0.2	10/0.2	10/0.2
0.2	10/0.1	10/0.1	10/0.1	10/0.2	10/0.2
0.3	$\perp$	10/0.1	10/0.1	10/0.2	10/0.2
0.4	$\perp$	37/17.7	37/25.1	10/0.2	10/0.3
0.5	$\perp$	$\perp$	37/25.5	37/79.2	37/199.2
0.6	$\perp$	$\perp$	53/216.7	37/94.1	37/216.7
0.7	$\perp$	$\perp$	$\perp$	53/462.0	–
0.8	$\perp$	$\perp$	$\perp$	$\perp$	–
0.9	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

area. The source of incompleteness in this domain model comes from the assumption that each pair of robots R_{1j} and R_{2j} ($1 \leq j \leq m$) are made by the same manufacturer M_j, both therefore might fail to load a *heavy* container.[6] The actions loading containers onto trucks using robots made by a particular manufacturer (e.g., the action schema *load-truck-with-robots-of-M1* using robots of manufacturer M_1), therefore, have a *possible precondition* requiring that containers should not be heavy. To simplify discussion (see below), we assume that robots of different manufacturers may fail to load heavy containers, though independently, with the same probability of 0.7. The goal is to transport all three containers in the city C_1 to C_2, and vice versa. For this domain model, a plan to ship a container to another city involves a step of loading it onto the truck, which can be done by a robot (after moving it from the airport to the downtown). Plans can be made more robust by using additional robots of *different* manufacturer after moving them into the downtown areas, with the cost of increasing plan length.

Satellite: In this domain, there are two satellites S_1 and S_2 orbiting the planet Earth, on each of which there are m instruments $L_{i1}, ..., L_{im}$ ($i \in \{1, 2\}$, $m \geq 1$) used to take images of interested modes at some direction in the space. For each $j \in \{1, ..., m\}$, the lenses of instruments L_{ij}'s were made from a type of material M_j, which might have an error affecting the quality of images that they take. If the material M_j actually has error, all instruments L_{ij}'s produce mangled images. The knowledge of this incompleteness is modeled as a *possible add effect* of the action taking images using instruments made from M_j (for instance, the action schema *take-image-with-instruments-M1* using instruments of type M_1) with a probability of p_j, asserting that images taken might be in a bad condition. A typical plan to take an image using an instrument, e.g. L_{14} of type M_4 on the satellite S_1, is first to switch on L_{14}, turning the satellite S_1 to a ground direction from which L_{14} can be calibrated, and then taking image. Plans can be made more robust by using additional instruments, which might be on a different satellite, but should be of *different* type of materials and can also take an image of the interested mode at the same direction.

Tables 1 and 2 show respectively the results in the Logistics and Satellite domains with $m = \{1, 2, ..., 5\}$ and $\rho \in \{0.1, 0.2, ..., 0.9\}$. The number of complete domain models in the two domains is 2^m. For Satellite domain, the probabilities p_j's range from 0.25, 0.3,... to 0.45 when m increases from 1, 2, ... to 5. For each specific value of ρ and m, we report l/t where l is the length of plan and t is the running time (in seconds). Cases in which no plan is found within the time limit are denoted by "–", and those where it is provable that no plan with the desired robustness exists are denoted by "$\perp$".

Observations on fixed value of m: In both domains, for a fixed value of m we observe that the solution plans tend to be longer with higher robustness threshold ρ, and the time to synthesize plans is also larger. For instance, in Logistics with $m = 5$, the plan returned has 48 actions if $\rho = 0.3$, whereas 66-length plan is needed if ρ increases to 0.4. Since loading containers using the same robot multiple times does not increase the chance of success, more robots of different manufacturers need to move into the downtown area for loading containers, which causes an increase in plan length. In the Satellite domain with $m = 3$, similarly, the returned plan has 37 actions when $\rho = 0.5$, but requires 53 actions if

[6] The *uncorrelated incompleteness* assumption applies for possible preconditions of action schemes specified for different manufacturers. It should not be confused here that robots R_{1j} and R_{2j} of the same manufacturer M_j can independently have fault.

$\rho = 0.6$—more actions need to calibrate an instrument of different material types in order to increase the chance of having a good image of interested mode at the same direction.

Since the cost of actions is currently ignored in the compilation approach, we also observe that more than the needed number of actions have been used in many solution plans. In the Logistics domain, specifically, it is easy to see that the probability of successfully loading a container onto a truck using robots of k ($1 \leq k \leq m$) different manufacturers is $(1-0.7^k)$. As an example, however, robots of all five manufacturers are used in a plan when $\rho = 0.4$, whereas using those of three manufacturers is enough.

Observations on fixed value of ρ: In both domains, we observe that the maximal robustness value of plans that can be returned increases with higher number of manufacturers (though the higher the value of m is, the higher number of complete models is). For instance, when $m = 2$ there is not any plan returned with at least $\rho = 0.6$ in the Logistics domain, and with $\rho = 0.4$ in the Satellite domain. Intuitively, more robots of different manufacturers offer higher probability of successfully loading a container in the Logistics domain (and similarly for instruments of different materials in the Satellite domain).

Finally, it may take longer time to synthesize plans with the same length when m is higher—in other words, the increasing amount of incompleteness of the domain makes the plan generation phase harder. As an example, in the Satellite domain, with $\rho = 0.6$ it takes 216.7 seconds to synthesize a 37-length plan when there are $m = 5$ possible add effects at the schema level of the domain, whereas the search time is only 94.1 seconds when $m = 4$. With $\rho = 0.7$, no plan is found within the time limit when $m = 5$, although a plan with robustness of 0.7075 exists in the solution space. It is the increase of the branching factors and the time spent on satisfiability test and weighted model-counting used inside the planner that affect the search efficiency.

Domains with random incompleteness: We built a program to generate an incomplete domain model from a deterministic one by introducing M new propositions into each domain model (all are initially **T**). Some of those new propositions were randomly added into the sets of *possible* preconditions/effects of actions. Some of them were also randomly made *certain* add/delete effects of actions. With this strategy, each solution plan in an original deterministic domain model is also a *valid plan*, as defined earlier, in the corresponding incomplete domain model. Our experiments with the Depots, Driverlog, Satellite and Zenotravel domains indicate that because the annotations are random, there are often fewer opportunities for the PFF planner to increase the robustness of a plan prefix during the search. This makes it hard to generate plans with a desired level of robustness under given time constraint.

Discussion: The techniques presented in this section for the GE semantics can be adapted for the compilation approach under the SE semantics. The compiled actions must have additional conditional effects to capture execution scenarios where either a known precondition or a possible precondition being realized is not satisfied in the current state. The effects in those cases must result in $\perp$, thus leading the system to the dead end state $s_\perp$. We however note that the requirement for mutually exclusive and exhaustive conditions in each conditional effect of the CPP formulation can significantly increase the number of conditional effects in the compiled actions.

8. Synthesizing robust plans with heuristic search

We now present an anytime forward search approach to synthesizing robust plans under the STRIPS execution semantics. Although the solution space is more limited than that of the GE semantics, the proposed approach is much more scalable than the compilation approach in the previous section. At a high level, in each iteration it searches for a plan π with the robustness $R(\pi)$ greater than a threshold δ, which is set to zero initially. The threshold is then updated with $R(\pi)$, preparing for the next iteration. The last plan produced is the most robust plan. We will briefly discuss a similar idea for the Generous Execution semantics at the end of this section.

The main technical part of our approach is a procedure to extract a relaxed plan $\widetilde{\pi}$, given the current plan prefix π_k of k actions and threshold δ, such that $\text{WMC}(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}}) > \delta$. (The sets of constraints Σ_π, Σ_{π_k} and $\Sigma_{\widetilde{\pi}}$ used in this approach for the SE semantics are presented in Section 6.2.) The length of $\widetilde{\pi}$ estimates the additional search cost $h(\pi_k, \delta)$ to reach goals G, starting from π_k, with more than δ probability of success. Since $\text{WMC}(\cdot)$ is costly to compute, we approximate it with a lower bound $l(\cdot)$ and upper bound $u(\cdot)$, which will then be used during the relaxed plan extraction. In particular, we look for the relaxed plan $\widetilde{\pi}$ satisfying $l(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}}) > \delta$, and compute the exact robustness of a candidate plan π only if $u(\Sigma_\pi) > \delta$.

In this section, we first introduce an approximate transition function $\widetilde{\gamma}_{SE}(\pi, s)$ that will be used during the forward search. It defines a set of propositions that might be **T** during the execution phase, thus helps the search in quickly defining applicable actions and checking potential goal satisfaction at each search step. The set will also be used as the first propositional layer of the relaxed planning graphs for extracting relaxed plans. We then describe our lower and upper bound for $\text{WMC}(\Sigma)$ of a clause set Σ, presents our procedure for extracting relaxed plan, and then discuss our choice for the underlying search algorithm.

8.1. An approximate transition function

In our approximate transition function, the resulting state from applying action a in state s is defined as follows:

$$\widetilde{\gamma}_{SE}(\langle a\rangle, s) = \begin{cases} (s \setminus Del(a)) \cup Add(a) \cup \widetilde{Add}(a) & \text{if } Pre(a) \subseteq s; \\ s_\perp & \text{otherwise.} \end{cases} \quad (15)$$

The projection of an action sequence π from a state s is defined as $\widetilde{\gamma}_{SE}(\pi, s) = s$ if $\pi = \langle\rangle$, and $\widetilde{\gamma}_{SE}(\pi, s) = \widetilde{\gamma}_{SE}(\langle a\rangle, \widetilde{\gamma}_{SE}(\pi', s))$ if $\pi = \pi' \circ \langle a\rangle$. The sequence π is a *valid plan* for a planning problem $\widetilde{\mathcal{P}}$ under the approximate transition function if $\widetilde{\gamma}_{SE}(\pi, I) \supseteq G$.

Proposition 1. *For any complete model $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$, action sequence π and state s such that $\gamma_{SE}^{\mathcal{D}}(\pi, s) \neq s_\perp$, $\gamma_{SE}^{\mathcal{D}}(\pi, s) \subseteq \widetilde{\gamma}_{SE}(\pi, s)$.*

Proof. We prove the theorem by induction on the length of π. For the base case when $\pi = \langle\rangle$, $\gamma_{SE}^{\mathcal{D}}(\langle\rangle, s) = \widetilde{\gamma}_{SE}(\langle\rangle, s) = s$ for any complete model $\mathcal{D}$ and state s. Assuming that the theorem holds for any action sequences π of length k ($k \geq 0$), complete model $\mathcal{D}$ and state s such that $\gamma_{SE}^{\mathcal{D}}(\pi, s) \neq s_\perp$. Consider a complete model $\mathcal{D}$, state $s \neq s_\perp$ and a sequence $\pi = \pi' \circ \langle a\rangle$ of $k+1$ actions such that $\gamma_{SE}^{\mathcal{D}}(\pi, s) \neq s_\perp$. Let $s_{\pi'}^{\mathcal{D}} = \gamma_{SE}^{\mathcal{D}}(\pi', s)$, $\tilde{s}_{\pi'}^{\mathcal{D}} = \widetilde{\gamma}_{SE}(\pi', s)$. It must hold that $s_{\pi'}^{\mathcal{D}} \neq s_\perp$ and $Pre(a) \subseteq \gamma_{SE}^{\mathcal{D}}(\pi', s)$, because otherwise $\gamma_{SE}^{\mathcal{D}}(\pi, s) = s_\perp$. Thus,

$$\gamma_{SE}^{\mathcal{D}}(\pi, s) = (s_{\pi'}^{\mathcal{D}} \setminus Del^{\mathcal{D}}(a)) \cup Add^{\mathcal{D}}(a). \quad (16)$$

From the inductive hypothesis: $s_{\pi'}^{\mathcal{D}} \subseteq \tilde{s}_{\pi'}^{\mathcal{D}}$, and since $Pre(a) \subseteq \gamma_{SE}^{\mathcal{D}}(\pi', s)$ we have $Pre(a) \subseteq \widetilde{\gamma}_{SE}(\pi', s)$. Therefore,

$$\widetilde{\gamma}_{SE}(\pi, s) = (\tilde{s}_{\pi'}^{\mathcal{D}} \setminus Del(a)) \cup Add(a) \cup \widetilde{Add}(a). \quad (17)$$

Since $s_{\pi'}^{\mathcal{D}} \subseteq \tilde{s}_{\pi'}^{\mathcal{D}}$, $Del^{\mathcal{D}}(a) \supseteq Del(a)$, $Add^{\mathcal{D}}(a) \subseteq Add(a) \cup \widetilde{Add}(a)$, from Eq. (16) and (17) we have $\gamma_{SE}^{\mathcal{D}}(\pi, s) \subseteq \widetilde{\gamma}_{SE}(\pi, s)$. The theorem thus holds with sequences of $k+1$ actions. □

Theorem 5. *A sequence of actions π is a valid plan under γ_{SE} if and only if it is a valid plan under $\widetilde{\gamma}_{SE}$.*

Proof. *If:* Given that $\widetilde{\gamma}_{SE}(\pi, I) \models G$, π is a plan under the complete model $\mathcal{D}_0$ in which $Pre^{\mathcal{D}_0}(a) = Pre(a)$, $Add^{\mathcal{D}_0}(a) = Add(a) \cup \widetilde{Add}(a)$ and $Del^{\mathcal{D}_0}(a) = Del(a)$ for all actions $a \in A$. Thus π is a valid plan under γ_{SE}.
Only if: Assuming that π is a valid plan under γ_{SE}. Let $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ such that $\gamma_{SE}^{\mathcal{D}}(\pi, I) \models G$. From Proposition 1, $\gamma_{SE}^{\mathcal{D}}(\pi, I) \subseteq \widetilde{\gamma}_{SE}(\pi, I)$. Thus, $\widetilde{\gamma}_{SE}(\pi, I) \models G$, or π is a valid plan under $\widetilde{\gamma}_{SE}$. □

Theorem 5 above implies that using the approximate transition function ensures the *completeness* property (that is, any plan achieving goals G in the ground truth model $\mathcal{D}^*$ is not excluded from the search space), and the *soundness* property (in the sense that any valid plan for $\widetilde{\mathcal{P}}$ is a plan achieving G in at least one complete model $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$). The approximate transition function $\widetilde{\gamma}_{SE}$ will therefore be used together with other components presented below for synthesizing robust plans.

8.2. Approximating weighted model count

Our approach for generating robust plans requires the computation for weighted model counts of clause sets during the search and robust relaxed plan extraction. Given that such a computation has been proved to be hard, we introduce the lower bound and upper bound for the exact weighted model count WMC(Σ) of clause set Σ, which can be computed in polynomial time.

Lower bound: We observe that the set of constraints Σ, constructed as in (11)–(14), can be converted into a set of clauses containing only positive literals (or *monotone clauses*). In particular, the variables p_a^{add} only appear in positive form in the resulting clauses. Variables p_a^{pre} and p_a^{del}, however, are all in negation form, thus can be replaced with np_a^{pre} and np_a^{del} having the corresponding weights $1 - w_a^{pre}(p)$ and $1 - w_a^{del}(p)$. As a result, the following theorem shows that the quantity $l_\Sigma = \prod_{c_i} Pr(c_i)$ can be used as a lower bound for WMC(Σ), where $Pr(c_i)$ is the probability of $c_i = \mathbf{T}$. (Recall that for a monotone clause $c = \bigvee_i x_i$, $Pr(c) = 1 - \prod_i (1 - w_i)$, where w_i is the probability of $x_i = \mathbf{T}$.)

Theorem 6. *Given a set of monotone clauses $\Sigma = \{c_1, ..., c_k\}$, $l_\Sigma = \prod_{c_i \in \Sigma} Pr(c_i) \leq WMC(\Sigma)$.*

Proof (sketch). For any two monotone clauses c and c', we can show that $Pr(c|c') \geq Pr(c)$ holds. (As an intuition, since c and c' have "positive interaction" only, observing one of the literals in c' cannot reduce belief that c is $\mathbf{T}$.) More generally, $Pr(c|c'_1 \wedge ... \wedge c'_t) \geq Pr(c)$ holds for monotone clauses c, c'_1, ..., c'_t. Therefore, WMC(Σ) $= Pr(\Sigma) = Pr(c_1)Pr(c_2|c_1)...Pr(c_k|c_1 \wedge ... \wedge c_{k-1}) \geq \prod_{c_i} Pr(c_i)$. □

Upper bound: One trivial upper bound for $Pr(\Sigma)$ is $\min_{c_i} Pr(c_i)$. We can however derive a much tighter bound for WMC(Σ) by observing that literals representing the realization of preconditions and effects on different predicates would

not be present together in one clause. This suggests that the set of clauses Σ is essentially decomposable into independent sets of clauses, each contains literals on one specific predicate. Clauses related to the same predicate, furthermore, can also be partitioned into smaller sets. Thus, to derive a better upper bound for $Pr(\Sigma)$, we first divide it into independent clause sets $\Sigma^1, ..., \Sigma^m$, and compute an upper bound u_Σ as follows: $u_\Sigma = \prod_{\Sigma^i} \min_{c \in \Sigma^i} Pr(c)$.

8.3. Extracting robust relaxed plan

We now introduce our procedure to extract a relaxed plan $\widetilde{\pi}$ such that $l(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}}) > \delta$, where Σ_{π_k} and $\Sigma_{\widetilde{\pi}}$ are the sets of constraints for the executability of actions in π_k and $\widetilde{\pi}$; note that $\Sigma_{\widetilde{\pi}}$ also includes constraints for a_G, thus for the achievement of G. Our procedure employs an extension of the common relaxation technique (cf., Bonet and Geffner [2]) by ignoring both the known and possible delete effects of actions in constructing $\widetilde{\pi}$.[7]

Relaxed planning graph construction
We construct the relaxed planning graph $\mathcal{G} = \langle L_1, A_1, ..., L_{T-1}, A_{T-1}, L_T \rangle$ for the plan prefix π_k, in which each proposition p and action a at layer t is associated with clause sets $\Sigma_p(t)$ and $\Sigma_a(t)$.

- $L_1 \equiv s_{k+1}$, where $s_{k+1} = \widetilde{\gamma}_{SE}(\pi_k, I)$. The clause set $\Sigma_p(1)$ for each $p \in L_1$, constructed with Constraints (11) and (12), represents the constraints on actions of π_k under which $p = \mathbf{T}$ at the first layer.
- Given proposition layer L_t, A_t contains all actions a whose known preconditions appear in L_t (i.e., $Pre(a) \subseteq L_t$), and a complete action, $noop_p$, for each $p \in L_t$: $Pre(noop_p) = Add(noop_p) = \{p\}$, $Del(noop_p) = \emptyset$. The constraints for the non-*noop* actions:

$$\Sigma_a(t) = \bigwedge_{p \in Pre(a)} \Sigma_p(t) \wedge \bigwedge_{q \in \widetilde{Pre}(a)} (q_a^{pre} \Rightarrow \Sigma_q(t)). \tag{18}$$

  All actions *noop* have the same constraints with their corresponding propositions.
- Given action layer A_t, the resulting proposition layer L_{t+1} contains all known and possible add effects of actions in A_t. The clause set of p at layer $t+1$ will be constructed by considering clause sets of all actions supporting and possibly supporting it at the previous layer, taking into account correctness constraints for the current plan prefix, i.e., Σ_{π_k}. In particular:

$$\Sigma_p(t+1) = \operatorname*{argmax}_{\Sigma \in S_1 \cup S_2} l(\Sigma \wedge \Sigma_{\pi_k}), \tag{19}$$

  where $S_1 = \{\Sigma_a(t) \mid p \in Add(a)\}$ and $S_2 = \{p_{a'}^{add} \wedge \Sigma_{a'}(t) \mid p \in \widetilde{Add}(a')\}$.[8]

We stop expanding the relaxed planning graph at layer L_T satisfying: (1) $G \subseteq L_T$, (2) the two layers L_{T-1} and L_T, which include the set of propositions and their associated clause sets, are exactly the same. If the second condition is met, but not the first, then the relaxed plan extraction fails. We also recognize an early stopping condition at which (1) $G \subseteq L_T$ and (2) $l(\Sigma_G) > \delta$, where Σ_G is the conjunction of clauses attached to goals $g \in G$ at layer T.

Relaxed plan extraction
We now extract actions from $\mathcal{G}$, forming a relaxed plan $\widetilde{\pi}$ such that $l(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}}) > \delta$. At a high level, our procedure at each step will pop a subgoal for being potentially supported. Each subgoal g is either a known or a possible precondition of an action a that has been inserted into the relaxed plan. The decision as to whether a subgoal should be supported depends on many factors (see below), all of which reflect our strategy that *a new action is inserted only if the insertion increases the robustness of the current relaxed plan.* If a new action is inserted, all of its known and possible preconditions will be pushed into the subgoal queue. Our procedure will stop as soon as it reaches a *complete* relaxed plan (see below) and its approximated robustness, specifically the value $l(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}})$, exceeds the current threshold δ (stop with success), or the subgoal queue is empty (stop with failure).

The relaxed plan while being constructed might contain actions a with unsupported *known* preconditions p, for which the supporting and protecting constraints cannot be defined. The constraints defining the (potential) executability of actions in such an *incomplete* relaxed plan, therefore, are not well-defined.[9] This is when the clause sets propagated in the relaxed planning graph play their role. In particular, we reuse the clause sets $\Sigma_p(t)$ associated with unsupported known preconditions p at the same layer t with a in the relaxed planning graph. We combine them with the constraints generated from Constraints (11)–(14) for (possibly) supported known and possible preconditions, resulting in constraints $\Sigma_{\widetilde{\pi}}$. Together with Σ_{π_k}, it is used to define the robustness of the relaxed plan being constructed.

[7] Thus, there are no protecting constraints in $\Sigma_{\widetilde{\pi}}$ caused by possible delete effects of actions in the relaxed plan.

[8] Note that the observation about converting constraints into monotone clauses, which facilitates our lower bound computation, still holds for constraints in (18) and (19).

[9] Unsupported possible preconditions do not result in incomplete relaxed plan, since they do not have to be supported.

Algorithm 1: Extracting robust relaxed plan.

1 **Input:** A threshold $\delta \in [0, 1]$, a plan prefix π_k, its correctness constraints Σ_{π_k}, the relaxed planning graph $\mathcal{G}$ of T layers.
2 **Output**: $\langle h(\pi_k, \delta), r\rangle$ if *success*, or nothing if *failure*.
3 **begin**
4 $\quad \widetilde{\pi} \leftarrow \langle a_G\rangle$, $s_{\rightarrow a_G} \leftarrow L_1$, $s^+_{\rightarrow a_G} \leftarrow s^+_{k+1}$.
5 $\quad$ **if** $r \leftarrow$ *CheckPlan*(π_k, δ, G) *succeeds* **then**
6 $\quad\quad$ **return** $\langle 0, r\rangle$, *success*.
7 $\quad$ **end**
8 $\quad \langle \Sigma_{\widetilde{\pi}}, Q\rangle \leftarrow$ InitializeConstraintsAndQueue$(G, \mathcal{G})$.
9 $\quad r \leftarrow l(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}})$.
10 $\quad$ **while** Q *not empty* **do**
11 $\quad\quad$ Pop $\langle p, a, t\rangle$ from Q s.t. $p \in L_t$.
12 $\quad\quad \langle a_{\text{best}}, l_{\text{best}}\rangle \leftarrow$ best supporting action and its layer in $\mathcal{G}$.
13 $\quad\quad$ **if** $l_{best} = 0$ **or** $\widetilde{\pi}$ *contains* $\langle a_{best}, l_{best}\rangle$ **then**
14 $\quad\quad\quad$ **continue**.
15 $\quad\quad$ **end**
16 $\quad\quad$ **if** $p \in Pre(a)$ **and** $p \notin s_{\rightarrow a}$ **then**
17 $\quad\quad\quad$ Insert $\langle a_{\text{best}}, l_{\text{best}}\rangle$ into $\widetilde{\pi}$.
18 $\quad\quad$ **end**
19 $\quad\quad$ **else**
20 $\quad\quad\quad r' \leftarrow$ evaluate$(a_{\text{best}}, l_{\text{best}}, \widetilde{\pi})$.
21 $\quad\quad\quad$ **if** $r' > r$ **then**
22 $\quad\quad\quad\quad$ Insert $\langle a_{\text{best}}, l_{\text{best}}\rangle$ into $\widetilde{\pi}$.
23 $\quad\quad\quad$ **end**
24 $\quad\quad$ **end**
25 $\quad\quad$ **if** $\langle a_{best}, l_{best}\rangle$ *inserted into* $\widetilde{\pi}$ **then**
26 $\quad\quad\quad$ Update $s_{\rightarrow a_{\text{best}}}$, $s^+_{\rightarrow a_{\text{best}}}$, $s_{\rightarrow a'}$ and $s^+_{\rightarrow a'}$ for all a' succeeding a_{best} in $\widetilde{\pi}$.
27 $\quad\quad\quad$ Update $\Sigma_{\widetilde{\pi}}$.
28 $\quad\quad\quad r \leftarrow l(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}})$.
29 $\quad\quad\quad count \leftarrow$ number of unsupported known preconditions in $\widetilde{\pi}$.
30 $\quad\quad\quad$ **if** $count = 0$ **and** $r > \delta$ **then**
31 $\quad\quad\quad\quad$ **return** $\langle |\widetilde{\pi}| - 1, r\rangle$, *success*.
32 $\quad\quad\quad$ **end**
33 $\quad\quad\quad$ **for** $q \in Pre(a_{best}) \cup \widetilde{Pre}(a_{best}) \setminus s^+_{\rightarrow a_{best}}$ **do**
34 $\quad\quad\quad\quad$ Push $\langle q, a_{\text{best}}, l_{\text{best}}\rangle$ into Q.
35 $\quad\quad\quad$ **end**
36 $\quad\quad$ **end**
37 $\quad$ **end**
38 $\quad$ **if** $r > \delta$ **then return** $\langle |\widetilde{\pi}| - 1, r\rangle$, *success*.
39 $\quad$ **return** *failure*.
40 **end**

The use of propagated clause sets in the relaxed planning graph reflects our intuition as to what the resulting relaxed plan should be. Given that each proposition p at layer t has a corresponding "best supporting action" at layer $t-1$, defined from Equation (19), whose known and possible preconditions in turn have their best supporting actions, there exists a set of actions, denoted by $Sup(p, t)$, that can be selected at all layers $t' \in \{1, ..., t-1\}$ to support p at layer t. Taking $\Sigma_p(t)$ into defining the robustness of an incomplete relaxed plan therefore implies that, in the worst case, we will have to include all actions in $Sup(p, t)$ into the resulting relaxed plan. Our procedure however will stop as soon as the relaxed plan is complete (i.e., all known preconditions and goals G are supported) and its lower bound for the robustness exceeds the threshold.

Algorithm 1 shows the details of our relaxed plan extraction procedure. We initialize the relaxed plan $\widetilde{\pi}$ with $\langle a_G\rangle$, the *relaxed plan state* $s_{\rightarrow a_G}$ before a_G and the set of propositions $s^+_{\rightarrow a_G}$ known to be **T** (Line 4). The set s^+_{k+1} contains propositions p known to be **T** after executing all actions in π_k—they are confirmed to be **T** at some step $C_p^{k+1} < k+1$, and not being possibly deleted by any other actions at steps after C_p^{k+1}. Line 5–7 checks if π_k is actually a plan with the robustness exceeding δ. This procedure, presented in Algorithm 2, uses the bound $u(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}})$ to prevent the exact weighted model counting from being called unnecessarily.

Line 8 initializes the constraints $\Sigma_{\widetilde{\pi}}$ and the queue Q of subgoals.[10] This procedure, presented in Algorithm 3, uses the constraints caused by actions of π_k for goals g that are (possibly) supported by π_k (Line 6-9), and the clause sets in $\mathcal{G}$ for those that are not (Line 11).

[10] In our implementation, subgoals in Q are ordered based on their layers (lower layers are preferred), breaking ties on the types of preconditions (known preconditions are preferred to possible preconditions), and then on the number of supporting actions.

Algorithm 2: CheckPlan.

Input: Plan prefix π_k, threshold $\delta \in [0, 1]$, goals G.
Output: $r \equiv R(\pi_k)$ if *success*, or nothing if *failure*.
begin
  if $G \subseteq s_{k+1}$ **then**
    Construct $\Sigma_{\widetilde{\pi}}$ using Constraints (11)–(14).
    if $u(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}}) > \delta$ **then**
      Compute $R(\pi_k) = \text{WMC}(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}})$.
      if $R(\pi_k) > \delta$ **then**
        $r \leftarrow R(\pi_k)$; **return** *success*.
      end
    end
  end
  return *failure*
end

Algorithm 3: InitializeConstraintsAndQueue.

Input: Goals G, relaxed planning graph $\mathcal{G}$.
Output: $\langle \Sigma_{\widetilde{\pi}}, Q \rangle$
begin
  $\Sigma_{\widetilde{\pi}} \leftarrow \mathbf{T}$.
  for $g \in G \setminus s^{+}_{\rightarrow a_G}$ **do**
    if $g \in s_{\rightarrow a_G}$ **then**
      $c \leftarrow$ Constraints (11)–(14) for g.
      $\Sigma_{\widetilde{\pi}} \leftarrow \Sigma_{\widetilde{\pi}} \wedge c$.
    end
    else
      $\Sigma_{\widetilde{\pi}} \leftarrow \Sigma_{\widetilde{\pi}} \wedge \Sigma_g(T)$.
    end
    Push $\langle g, a_G, T \rangle$ into Q.
  end
  return $\langle \Sigma_{\widetilde{\pi}}, Q \rangle$.
end

Line 10–37 of Algorithm 1 is our greedy process for building the relaxed plan. It first pops a triple $\langle p, a, t \rangle$ from Q to consider for supporting, then checks some conditions under which this subgoal can simply be skipped. Specifically, if $p \notin L_t$ (Line 11), which also implies that it is a possible precondition of a (otherwise, a cannot be included into A_t), then there are not any actions in the layer A_{t-1} that can (possibly) support p. Line 13–15 includes the other two conditions: a_{best} is in fact the last action in π_k, or it has already been inserted into $\widetilde{\pi}$. Here, a_{best} is the best non-*noop* supporting action for p at layer l_{best} of $\mathcal{G}$ (retrieved from Equation (19) and possibly a backward traversal over *noop* actions).

Line 16–18 handles a situation where the current relaxed plan $\widetilde{\pi}$ is actually incomplete—it includes actions having a known precondition p not (possibly) supported; thus $\pi_k \circ \widetilde{\pi}$ would have (exact) zero robustness. It is therefore reasonable to immediately insert a_{best} into the current relaxed plan to support p. Note that this insertion means that a_{best} is put right after all actions at layers before l_{best} that have been inserted into $\widetilde{\pi}$, maintaining the total-order of actions in $\widetilde{\pi}$.

In other scenarios (Line 19–24), i.e., $p \in Pre(a) \cap s_{\rightarrow a}$ or $p \in \widetilde{Pre}(a)$, from the plan validity perspective we do not need to add new action (possibly) supporting this subgoal. However, since valid relaxed plan might not be robust enough (w.r.t. the current threshold δ), a new supporting action might be needed. Here, we employ a greedy approach: Line 20 evaluates the approximate robustness r' of the relaxed plan if a_{best} at layer l_{best} is inserted. If inserting this action increases the current approximate robustness of the relaxed plan, i.e., $r' > r$, then the insertion will take place.

Line 25–36 are works to be done after a new action is inserted into the relaxed plan. They include the updating of relaxed plan states and set of propositions known to be $\mathbf{T}$ (Line 26),[11] the constraints $\Sigma_{\widetilde{\pi}}$ (Line 27) and the lower bound on the robustness of the relaxed plan (Line 28). The new approximate robustness will then be checked against the threshold, and returns the relaxed plan with success if the conditions meet (Line 29–32). We note that in addition to the condition that the new approximate robustness exceeds δ, the relaxed plan must also satisfy the validity condition: all known preconditions of actions must be (possibly) supported. This ensures that all constraints in $\Sigma_{\widetilde{\pi}}$ come from actions in $\widetilde{\pi}$, not from the clause set propagated in $\mathcal{G}$. In case the new relaxed plan is not robust enough, we push known and possible preconditions of the

[11] They are updated as follows: if a_j and a_{j+1} are two actions at steps j and $j+1$ of $\widetilde{\pi}$, then $s_{\rightarrow a_{j+1}} \leftarrow s_{\rightarrow a_j} \cup Add(a_j) \cup \widetilde{Add}(a_j)$ and $s^{+}_{\rightarrow a_{j+1}} \leftarrow s^{+}_{\rightarrow a_j} \cup Add(a_j)$.

Algorithm 4: Local Search with Restarts [3].

```
1  Data: S_min – a local minimum, failcount, failbound
2  Result: S' – a state with a strictly better heuristic value, failcount – updated fail count
3  begin
4      depth_bound ← initial_depth_bound
5      for i ← 1 to iterations do
6          for probes ← 1 to probes_at_depths do
7              S' ← S_min; for depth ← 1 to depth_bound do
8                  S' ← a neighbor of S'; if h(S') < h(S_min) then
9                      return S', failcount
10                 end
11             end
12             increment failcount if failcount = failbound then
13                 return abort search
14             end
15         end
16         double depthbound;
17     end
18     return abort search
19 end
```

newly inserted action into the subgoal queue Q, ignoring those known to be **T** (Line 33–35), and repeat the process. Finally, we again check if the approximate robustness exceeds δ (Line 38) to return the relaxed plan length with its approximate robustness; otherwise, our procedure fails (Line 39).

8.4. Search

We now discuss our choice for the underlying search algorithm. We note that the search for our problem, in essence, is performed over a space of belief states—in fact, our usage of the plan prefix π_k, the state $s_{k+1} = \widetilde{\gamma}_{SE}(\pi_k, I)$ and the set of known propositions s^+_{k+1} in the relaxed plan extraction makes the representation of belief state in our approach implicit, as in Conformant-FF [15] and Probabilistic-FF [7]. Checking duplicate belief states, if needed during the search, is expensive; to do this, for instance, Probabilistic-FF invokes satisfiability tests for certain propositions at a state just to check for a sufficient condition.

To avoid such an expensive cost, in searching for a plan π with $R(\pi) > \delta$ we incorporate our relaxed plan extraction into an extension of the *stochastic* local search with failed-bounded restarts (Algorithm 4) proposed by Coles et al. [3]. Given a plan prefix π_k (initially empty) and its heuristic estimation $h(\pi_k, \delta)$, we look for a sequence of actions π' such that the new sequence $\pi_k \circ \pi'$ has a better heuristic estimation: $h(\pi_k \circ \pi', \delta) < h(\pi_k, \delta)$. This is done by performing multiple probes [3] starting from $s_{k+1} = \widetilde{\gamma}_{SE}(\pi_k, I)$; the resulting sequence π is a plan with $R(\pi) > \delta$ if $h(\pi, \delta) = 0$. Since actions are stochastically sampled during the search, there is no need to perform belief state duplication detection.

8.5. Experimental results

We test our planner, *PISA*, with incomplete domain models generated from IPC domains: Depots, Driverlog, Freecell, Rover, Satellite and Zenotravel. For each of these IPC domains, we make them incomplete with n_p possible preconditions, n_a possible add and n_d possible delete effects. We make possible lists using the following ways: (1) randomly "moving" some known preconditions and effects into the possible lists, (2) delete effects that are not preconditions are randomly made to be possible preconditions of the corresponding operators, (3) predicates whose parameters fit into the operator signatures, which however are not parts of the operator, are randomly added into possible lists. For these experiments, we vary n_p, n_a and n_d in $\{0, 1, ..., 5\}$, resulting in $6^3 - 1 = 215$ incomplete domain models for each IPC domain mentioned above. We test our planner with the first 10 planning problems in each domain model, thus 2150 *instances* (i.e., a pair of incomplete domain model and planning problem). In addition, we also test with the Parcprinter incomplete domain available to us from the distribution of DeFault planner, which contains 300 instances. We restrict our experiments to incomplete domain models with only annotations of weights $\frac{1}{2}$; this is also the only setting that DeFault can accept. We run them on a cluster of computing nodes, each possesses multiple Intel(R) Xeon(R) CPU E5440 @ 2.83 GHz. For exact model counting, we use Cachet model counting software ([25]). For the search in *PISA* we use the similar configuration in Coles et al. [3], except for the size of the neighborhood being 5—our experiments on small set of instances suggest this is probably the best. All experiments were limited to the 15 minutes time bound.

Comparing to DeFault: We present our comparison between *PISA* and DeFault on five domains: Freecell, Parcprinter, Rover, Satellite and Zenotravel; DeFault cannot parse the other domains. We note that, although DeFault reads domain files describing operators with annotations, it assumes all annotations are specified at the grounded (or instantiated) level. Thus,

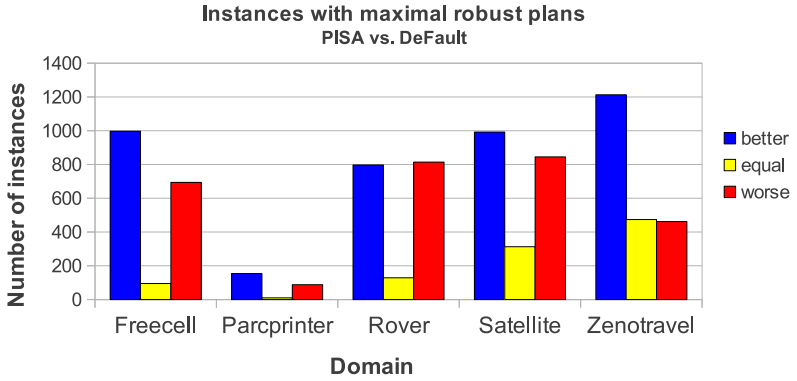


Fig. 4. Number of instances for which *PISA* produces better, equal and worse robust plans than DeFault.

we follow the same treatment by assuming possible preconditions and effects of grounded actions are all independent; the incompleteness amount can go up to, for instance in the Freecell domain, $K = 73034$ annotations (many of them however might be irrelevant to the actions in a resulting plan). We use the best configuration of DeFault: prime implicant heuristics with size $k = 2$ and 2 GB RAM as suggested, running it in the anytime mode.

Fig. 4 shows the number of instances for which *PISA* generates plans having better, equal and worse **robustness** values compared to DeFault.[12] Since the planners run in their anytime mode, returning plans with increasing plan robustness, we use the last plans produced by them in this comparison. Out of five domains, *PISA* generates better plans than DeFault in more instances of the four domains Freecell, Parcprinter, Satellite, Zenotravel, and a bit less in the Rover domain. More specifically, the percentage of instances for which *PISA* returns plans with *equal or better* robustness are always more than 50% in all domains: 61% in Freecell, 65% in Parcprinter, 53% in Rover, 60% in Satellite and 78.5% in Zenotravel. The percentages of instances with *strictly better* robustness in these domains respectively are 55.8%, 61.1%, 45.8%, 46.1% and 56.4%.

Robustness ratio: Regarding the robustness value, we calculate the ratio of best robustness values of plans returned by *PISA* to those by DeFault. Note that in this comparison, we consider only instances for which both planners return valid plans. These ratios are: 8069.65 in Freecell, 166.36 in Parcprinter, 1.78 in Rover, 135.97 in Satellite and 898.66 in Zenotravel. Thus, on average, *PISA* produces plans with significantly higher robustness than DeFault.

Planning time: Not only does *PISA* produce higher quality for plans, it also takes much less planning time.[13] To demonstrate this, we first consider instances for which the two approaches return best plans with the *same* robustness values. Fig. 5 shows the total time taken by *PISA* and DeFault in these instances. We observe that in 95 instances of Freecell domain that the two approaches produce equally robust plans, *PISA* is faster in 72 instances, and slower in the remaining 23 instances—thus, it wins in 75.8% of instances. Comparing the ratio of planning times, *PISA* is actually 654.7× faster, on average, than DeFault in these 95 instances. These faster vs. slower instances and the planning time ratio are 8 vs. 2 (80.0%) and 29.6× for Parcprinter, 103 vs. 10 (91.1%) and 1665× for Rover, 281 vs. 28 (90.9%) and 562.7× for Satellite, 329 vs. 86 (79.3%) and 482.9× for Zenotravel.

What is more interesting, in many cases, *PISA* is faster than Default, even when it produces significantly more robust plans. To show this, we again considered the last plan returned by each planner within the time bound, and the time when it was returned. Even for instances where the plan returned by *PISA* has *strictly better* robustness than that returned by DeFault, *PISA* often managed to return its plan significantly earlier. For example, in 65.4% of such instances in Freecell domain (315 out of 482), the planning time of *PISA* is also faster. These percentages of instances in Parcprinter, Rover, Satellite and Zenotravel are 52.8% (28 out of 53), 87.8% (144 out of 164), 55.9% (555 out of 992) and 46.5% (491 out of 1057) respectively. We also notice that *PISA* is faster than DeFault in synthesizing the *first valid plans* (that is, plans π such that $\widetilde{\gamma}_{SE}(\pi, I) \supseteq G$) for most of the instances in all domains: 84.9% (960 out of 1130) instances in Freecell domain, 94% (141 out of 150) in Parcprinter, 98.1% (789 vs. 804) in Rover, 93.4% (1999 out of 2140) in Satellite and 92.4% (1821 out of 1971) in Zenotravel. We think that both the search and the fact that our heuristic is sensitive to the robustness threshold, so that it can perform pruning during search, contribute to the performance of *PISA* in planning time.

Comparing with baseline approaches: We also compare *PISA* with an approach in which the relaxed plans are extracted in the similar way used in the FF planner [16]. In this approach, all annotations on possible preconditions and effects are ignored during the relaxed plan extraction. Note that all the other designs in *PISA* (such as checking $l(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}})$ against δ and the search algorithm) still remain the same. Unlike the earlier comparison with DeFault, incompleteness annotations are now applied at the operator level. *PISA* outperforms this FF-like heuristic approach in five domains: in particular, it produces *better* plans in 72.9% and *worse* in 8.3% instances of the Depots domain; similarly, 75.3% and 4.2% instances of Driverlog,

[12] We ignore instances for which both planners fail to return any valid plan; for comparing instances with equal robustness, we only consider those for which both planner return valid plans.

[13] In running time comparison, we consider only instances in which both planners produce valid plans within the time bound.

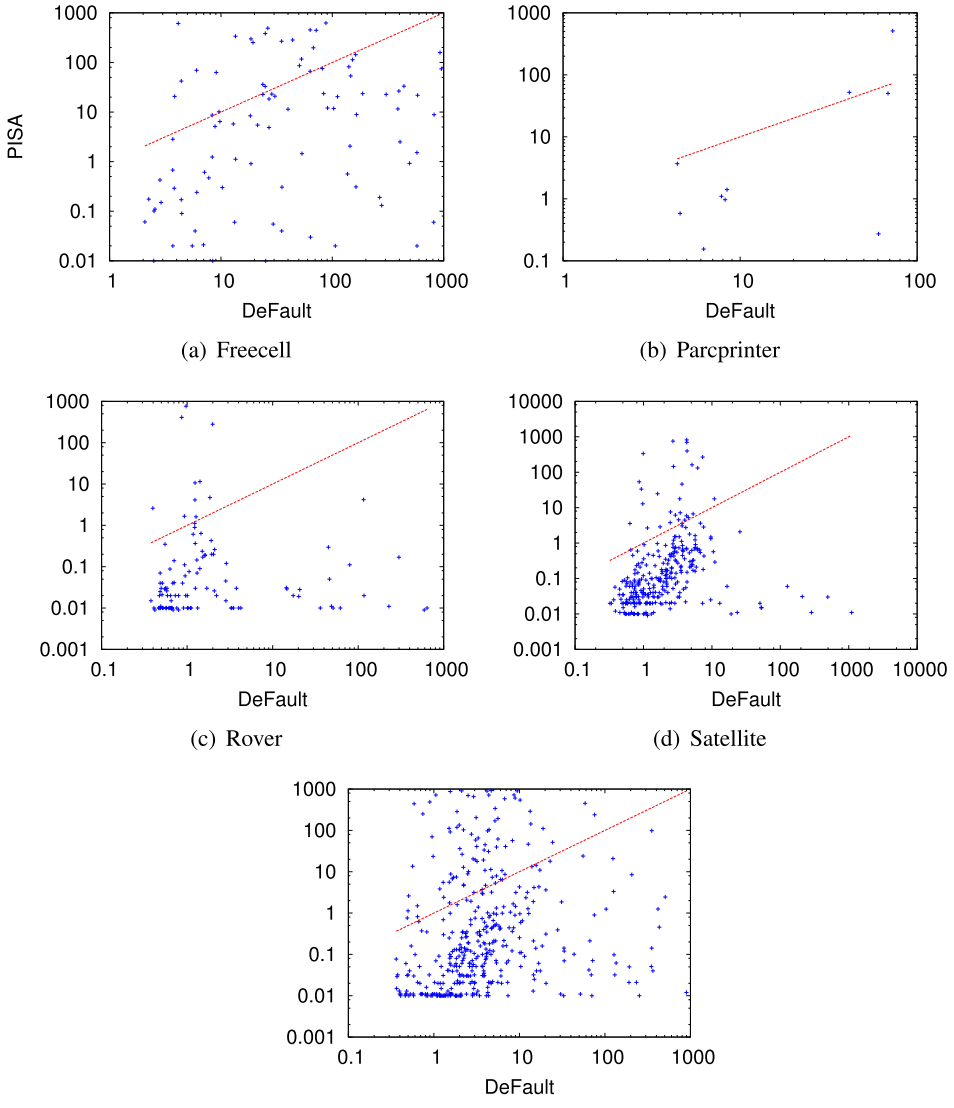


Fig. 5. Total time in seconds (log scale) to generate plans with the same robustness by *PISA* and DeFault. Instances below the red line are those in which *PISA* is faster. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

70.2% and 2.9% for Rover, 84.1% and 1.8% for Satellite, 61% and 8.3% in Zenotravel. *PISA* however is not as good as this baseline approach in the other two domains: it returns worse plans in 53.1% and only better in 12.4% instances of Freecell; the corresponding percentages in Parcprinter are 50.5% and 13.1%.

In the second baseline approach, we use exact model counting WMC(Σ) during the extraction of relaxed plans, replacing the approximation $l(\Sigma)$. This approach, as anticipated, spends most of the running time for the exact model counting. The results are discouraging, thus we will not go into the details.

Discussion: The high level ideas of the heuristic approach presented in this section might also be useful for a similar approach under the GE semantics, however with additional challenges. In particular, the special property used for approximating plan robustness under the SE semantic no longer holds for GE semantics, thus the proposed lower and upper bounds are inapplicable. One might attempt to investigate the application of the work on approximate model counting, cf. [32,13], for the encoding for plan correctness under the GE semantics in Section 6.1.

9. Using successful plan traces to improve plan robustness

PISA shows that in planning scenarios with incomplete domain models, the knowledge about the incompleteness such as the annotations considered in our setting greatly helps with defining the quality of plans and in synthesizing robust plans.

In this section we extend our *PISA* approach to exploit a different source of information, the observation of past successful plans. These plan traces reveal information about the true domain model in an indirect way. In particular, any candidate domain model where the traces are not "correct" (i.e., do not successfully execute) cannot be true domain models, and thus can be eliminated. In other words, successful plan traces prune the candidate set of feasible domain models, and *PISA* needs only to synthesize a plan that is robust with respect to the remaining candidate domain models.

In our extended approach, called *CPISA*, this is accomplished by modifying the robustness assessment phase such that in addition to the constraints ensuring the correctness of the plan under development, we also add constraints ensuring that the provided plan traces are indeed correct. These latter constraints effectively put additional restrictions on which possible preconditions and possible effects are realized in the true domain model. In particular, the plan traces can either eliminate, force or correlate the realization of possible preconditions, possible adds and possible deletes of any of the actions in the domain model.

As an example, consider a successful plan trace $\langle a_i, a_j \rangle$ for a problem with initial state I and goals G. Suppose a_j has a possible precondition p, which doesn't occur in I or anywhere in the definite or possible effects of a_i. In such a case, we can infer that in the true model, p cannot be a precondition of a_j (because otherwise, the plan couldn't be a successful trace). Thus, p is effectively eliminated as a "possible" precondition of a_j. This in turn halves the number of candidate domain models.

For the case above, the possible precondition establishment and protection constraints (see Equation (13)) will add the constraint

$$p^{pre}_{a_j} \Rightarrow \mathbf{F}$$

(since there are no a_k that can possibly support p, the right hand side disjunction in Equation (13) becomes empty). On the other hand if a_i, in fact, had a possible add effect that could have turned p true, *CPISA* would have included the constraint

$$p^{pre}_{a_j} \implies p^{add}_{a_i}.$$

It is important to note that the trace does not force the possible add for the action a_i to be true, rather it correlates it to the possible precondition of a_j. Specifically, if a_j does wind up having p as a precondition in the real model, then a_i must have p as an effect in the real model.

9.1. Posterior robustness measure

It is worth noting that since robustness measures the fraction of the candidate domain models in which the plan under development correctly executes, the estimate naturally changes when the planner has access to more successful traces. This is because the successful traces eliminate some of the candidate models, changing the base set of models over which we evaluate the correctness of the plan. Since *PISA* can be seen as a special case of *CPISA* with no knowledge of successful traces, a given plan π can be seen to have different robustness according to *PISA* and *CPISA*. The robustness estimated by *PISA* can be seen as a 'prior' estimate (with no additional knowledge of plan traces), while that estimated by *CPISA* is a 'posterior' estimate (that takes into account the knowledge of the plan traces).

Consider a set of plan traces $\mathcal{T}$ that succeeded in achieving their goals under the (unknown) ground truth model $\mathcal{D}^*$. Recall that the robustness of a plan π for the problem $\widetilde{\mathcal{P}}$, as defined in Eq. (5), is the cumulative probability mass of the completions of $\widetilde{\mathcal{D}}$ under which π succeeds (in achieving the goals). In the presence of plan traces $\mathcal{T}$, the extended robustness measure, denoted by $R(\pi|\mathcal{T})$, naturally is revised to be the probability that π succeeds given that all plans in $\mathcal{T}$ succeed:

$$R(\pi|\mathcal{T}) \stackrel{def}{\equiv} Pr(\pi|\mathcal{T}). \tag{20}$$

We denote $Pr(\mathcal{T})$ as the probability that all the plans in $\mathcal{T}$ succeed, and similarly $Pr(\pi \wedge \mathcal{T})$ the probability that π and all plans in $\mathcal{T}$ succeed. The robustness $R(\pi|\mathcal{T})$ above is equivalent to

$$Pr(\pi|\mathcal{T}) = \frac{Pr(\pi \wedge \mathcal{T})}{Pr(\mathcal{T})} = \frac{\text{WMC}(\Sigma_\pi \wedge \Sigma_\mathcal{T})}{\text{WMC}(\Sigma_\mathcal{T})}, \tag{21}$$

where $\Sigma_\mathcal{T} = \bigcup_{\pi_i \in \mathcal{T}} \Sigma_{\pi_i}$, i.e., the set of correctness constraints, as defined in Equations (11)–(14), for all plan traces.

In extending *PISA* to take plan traces $\mathcal{T}$ into account, we create the set of constraints $\Sigma_\mathcal{T}$ up front and then use them at every step of the search. In particular, the quality of a relaxed plan $\widetilde{\pi}$ constructed during the search for the plan prefix π_k is evaluated with the quantity

$$Pr(\pi_k \circ \widetilde{\pi}|\mathcal{T}) = \frac{Pr(\pi_k \circ \widetilde{\pi} \wedge \mathcal{T})}{Pr(\mathcal{T})} = \frac{\text{WMC}(\Sigma_{\pi_k} \wedge \Sigma_{\widetilde{\pi}} \wedge \Sigma_\mathcal{T})}{\text{WMC}(\Sigma_\mathcal{T})}. \tag{22}$$

While the denominator is computed once, the numerator of the above quantity is approximated during the search. We use the same approximations presented earlier for the weighted model counting.

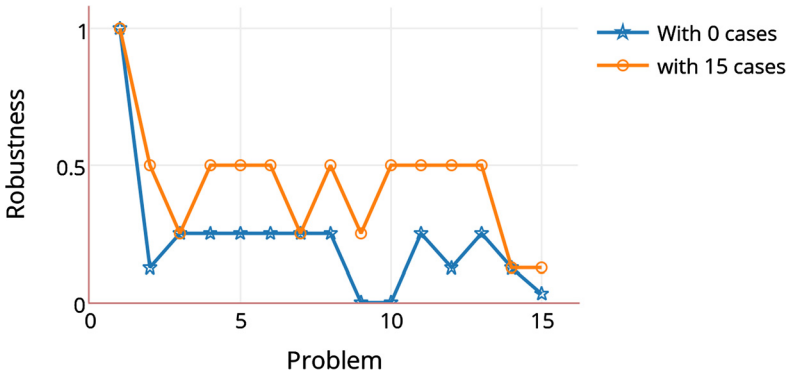


Fig. 6. Comparing the plans produced with 0 cases (*PISA*) against 15 cases (*CPISA*) in Zenotravel domain.

9.2. Experiments comparing CPISA and PISA

We ran *CPISA* against two domains Zenotravel and Rover to demonstrate its utility. Both of these domains were used in evaluating the original approach. As before, we start from the correct domain models, and annotating a random set of preconditions and effects as *possible* preconditions and effects. In addition to making definite preconditions/effects into possible preconditions/effects, we also add additional domain conditions as possible preconditions and effects. This is to account for the fact that the domain writers may not necessarily know, at the time of annotation, which conditions are actually present in the true model. While adding these new predicates, we make sure that the parameter list of each new predicate was compatible with the signature of the operator. Next we created a set of random problems of varying sizes for each original complete domain model (using standard problem generators), which we then solved with respect to the actual domain model using an off-the-shelf planner. The solutions to these problems formed our case library.

Some of these cases might implicitly eliminate some domain models from consideration. For example, in the case of Zenotravel, let us assume we have an operator '`board`' with the following definition

```
(:action board
     (?p - person ?a - aircraft ?c - city)
      :precondition (and (at ?a ?c) (at ?p ?c))
      :poss-precondition ()
      :effect (and (in ?p ?a))
      :poss-effect (and (not (at ?a ?c))).
```

As shown in its definition, the operator `board` has a possible effect `(not (at ?a ?c))`, which means that after boarding the plane will no longer be in that city. Next let us consider a plan trace containing the following actions

```
(board person1 plane3 city4)
(fly plane3 city4 city5 fl4 fl3)
```

This plan trace tells us that after `person1` boarded `plane3` in city `city4`, the plane flew from `city4` to `city5` thereby reducing its fuel level from `fl4` to `fl3`. If we knew that the operator `fly` had a known precondition that the plane should be in the source city, this plan trace would end up eliminating all domain models where the operator `board` has an effect

```
(not (at ?a ?c))
```

In the case of Zenotravel domain, we created an incomplete domain model by introducing five possible preconditions, five possible add effects and nine possible delete effects. Out of these possible predicates, nine of them are part of the actual base domain model, and ten of them were incorrect predicates added randomly. We created 15 case files by solving random problems using the base domain model. The 19 possible predicates represent 524,288 possible number of domains, of which only 1024 domains can support the execution of all the provided cases.

We chose to test this domain on 15 problems of varying sizes and we ran the same problems using the original *PISA* system and with differing number of cases. Each system was run with a 15 minutes time limit per problem. Once we had the resulting plans for all the problems, we collected the best plan for each problem as produced by the respective approaches. Since both approaches rely on stochastic search, each planning instance was run multiple times to ensure that the search were not getting stuck on undesirable search paths. The result of the comparison is given in Figs. 6 and 7.

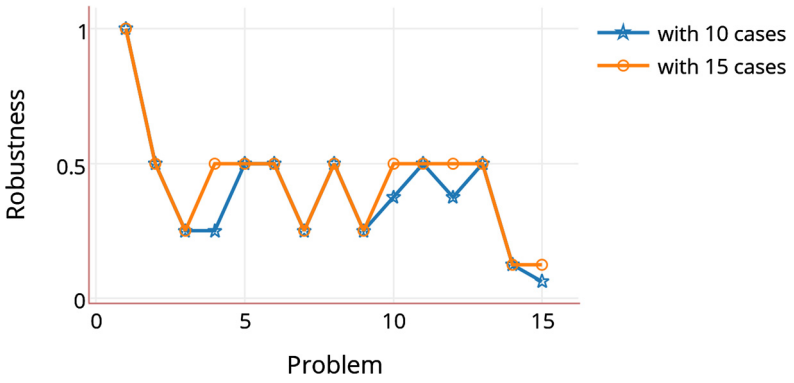


Fig. 7. Comparing the plans produced with 10 cases against 15 cases (Zenotravel domain).

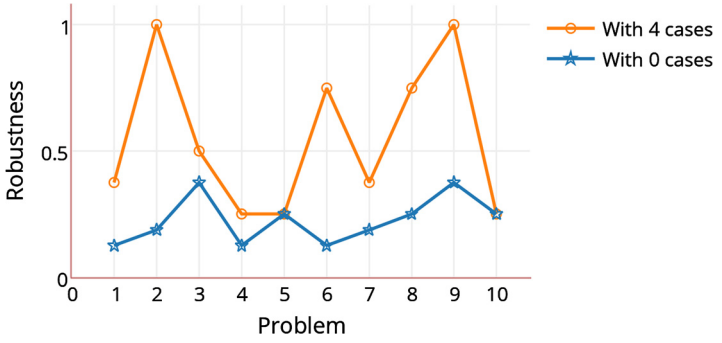


Fig. 8. Comparing the plans produced with 0 cases against 4 cases (Rover domain).

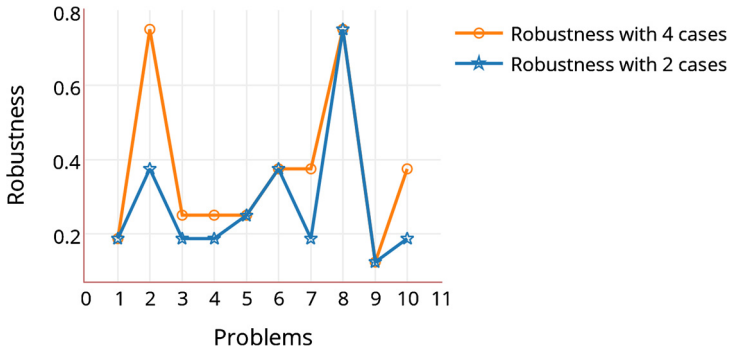


Fig. 9. Comparing the plans produced with 2 cases against 4 cases (Rover domain).

The results in Figs. 6 and 7 illustrate that, the plans produced in the presence of full case library are consistently more robust. In Fig. 6 we see that *CPISA* running with full case library was able to outperform the original approach (which is same as running *CPISA* in the absence of any case files) in 9/15 problems. It is interesting to note that *PISA* in fact twice chose plans which ended up having zero robustness in the reduced domain space. In Fig. 7, we compare the outputs of *CPISA* running with full case library against *CPISA* running on a subset of the total cases. Again we see that even though the full set only had five additional cases, it was still able to produce better plans in many problems.

Next for the Rover domain, we created an incomplete domain similar to Zenotravel. Our incomplete domain had 21 possible effect predicates and four possible precondition predicates. This includes a combination of existing predicates moved to possible section and new predicates. In total this represents 33,554,432 possible domains. Next we created four cases for the base domain, such that the four cases reduced the possible number of domains down to 65,535.

We tested the domain using ten problems of different sizes. Similar to Zenotravel, each problem in Rover domain was tested with the original approach and then tested with our extension for varying sizes of case library. The best plan produced by each variation of the approach is evaluated in the reduced domain space to obtain the best possible approximation of the robustness. Similar to Zenotravel each problem was run multiple times. The result of the experiment is shown in Fig. 8 and Fig. 9.

As is evident from the above results, *CPISA* was able to produce plans with better robustness for the majority of the problems.[14] One interesting point we noticed was that, the plans with better estimates of prior robustness (as measured by *PISA*) may actually have significantly worse posterior robustness estimate. This makes sense as the plan might have been working in models that are eliminated by the successful cases.

9.3. CPISA as a way to further reduce domain-modeling burden

Although our study focused on how *CPISA* can exploit knowledge about past successful cases, the same mechanism also allows a way out of a more fundamental dilemma about the operation of *PISA*. Specifically, one might question whether a domain-writer not willing to write down the complete domain theory can be expected to mark possible preconditions and effects. After all, it would be more natural to expect that the domain writer provides an incomplete model, with no special indications on possible preconditions/effects. *CPISA* suggests a satisfactory solution to this dilemma: we can start by assuming that for each action, any condition that is not already mentioned in its preconditions or effects list is a possible precondition or effect (respectively). While this exponentially increases the number of possible complete models that are consistent with the resultant incomplete model, the availability of the successful cases can help *CPISA* filter out many of these possible complete models. In particular, any complete model that does't ensure successful execution of every one of the cases will in effect be pruned from consideration.

9.4. Comparing CPISA to RIM

CPISA can be viewed as starting with a distribution over domain models (that are consistent with the annotated STRIPS model), and learning from past successful plan cases to compute a new posterior distribution over the domain models, which it then uses to compute robust plans for new problems. In contrast, RIM [36] is a competing approach that starts with a single domain model, and given successful plan cases, modifies its initial model into a single new model. This modified model not only captures additional precondition/effect knowledge about the given actions, but also "macro actions". RIM uses a MAX-SAT framework for learning, where the constraints are derived from the executability of the given plan traces, as well as the preconditions/effects of the given incomplete model. Unlike traditional macro-action learners which use macros to increase the efficiency of planning (in the context of a complete model), RIM's motivation for learning macros is to increase the accuracy (robustness) of the plans generated with the modified model. It thus provides an interesting counter-point to the *CPISA* approach.

We compare the performance of *CPISA* and RIM in three domains: Zenotravel, Blocksworld and Driverlog. For each domain, as before, a random set of preconditions and effects are removed from the domain file. This truncated domain model is given as the initial model for RIM. For *CPISA*, while we could simply annotate the removed preconditions/effects as possible preconditions/effects (as was done in our earlier evaluations), we decided not to do that as RIM cannot use the knowledge about possible preconditions/effects. Instead, we made *CPISA* make its own annotations in the manner described in Section 9.3. Specifically, *CPISA* creates its own annotated model by adding all possible predicates that are not already part of the precondition and effects of each operator (provided the operator parameters matches the predicate parameter types). Note that this approach considerably increases the complexity of the problem faced by *CPISA*, and can be seen as putting it at an unnecessary disadvantage in cases where annotated models are readily available.

Once the initial models are fixed, *CPISA* and RIM are both given the same set of successful cases, as well as a set of test problems. The plans produced by each of the systems for the test problems are evaluated in terms of the estimate of their posterior robustness. Note that while *CPISA* is starting with an incomplete domain model (which specifies a distribution over the complete candidate models, one of which is guaranteed to be the real model), RIM's initial model is a single model that is incorrect. While *CPISA* uses the successful cases to modify the "distribution" of the domain models (while retaining the correct model in the distribution), RIM uses the cases to *modify* the initial model into another (single) new model. The modified model may still be incorrect: it may produce incorrect plan or even no plan for a given problem.

In Zenotravel, we created an incomplete domain with 10 missing predicates by randomly removing 6 preconditions and 4 effects from the ground truth. This is converted to an annotated PDDL model with 56 annotations (20 possible preconditions and 36 possible effects), which is equivalent to $2^{46} = 7.03 \times 10^{13}$ possible domains. We start with eight cases that were randomly generated, and then we make use of various subsets of this case library. In particular, we use a set of two cases which halves the total number of possible domains, a set of four cases which further cuts it down to 1.28×10^{10}, and the complete set which represents a set of 3.2×10^9 domains. We test with 18 planning problems, with each plan generated by *CPISA* and RIM compared in terms of their posterior robustness estimate.

As shown in Figs. 10 and 11, *CPISA* clearly does better than RIM for smaller number of cases. This is because in the presence of a small number of cases (which means large number of candidate domain models), the probability that the single domain produced by RIM is the ground truth is quite unlikely. In contrast, *CPISA*, which reasons with the full distribution of candidate domain models, is able to produce plans with high robustness. As the number cases increase however, the case

[14] Note that because *PISA* and *CPISA* both use local search in finding robust plans, it is occasionally possible for *CPISA* to find a plan with lower robustness than *PISA*.

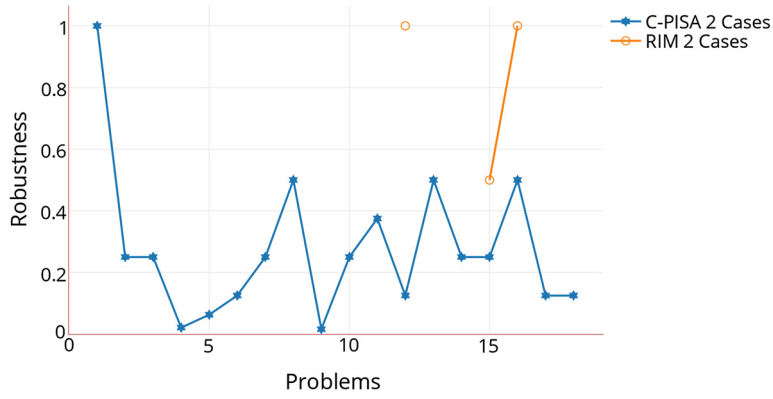


Fig. 10. Comparing plans produced by *CPISA* vs RIM provided two single cases in Zenotravel domain.

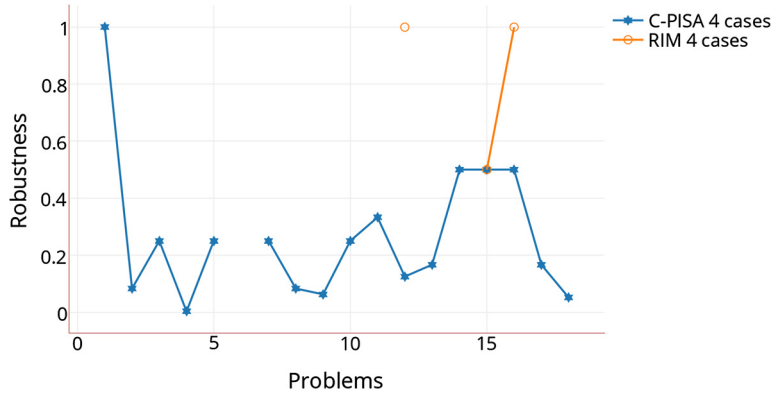


Fig. 11. Comparing plans produced by *CPISA* vs RIM provided with four cases in Zenotravel domain.

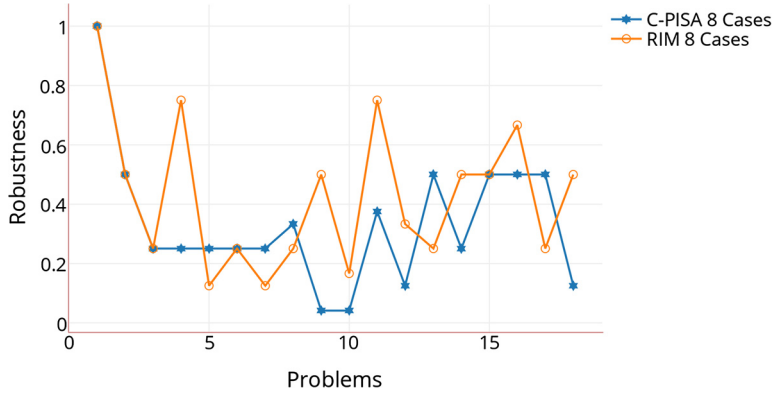


Fig. 12. Comparing plans produced by *CPISA* vs RIM provided with eight cases in Zenotravel domain.

constraints that *CPISA* has to reason with increases, thereby increasing the computational cost of the robustness estimation it uses during the plan generation. Because of this, in Fig. 12 we see that RIM starts doing better as it is able to access more cases.

For Blocksworld, we chose to compare the two approaches using a domain model with 5 missing preconditions and 10 missing effects. The incomplete model was converted to an annotated model with 19 possible preconditions and 32 possible effects (a total of 2.2×10^{15} possible domains) We evaluated both approaches by solving 15 random problems using a set of 5 and 15 randomly produced cases (representing approximately 5.1×10^{8} and 4.0×10^{7} domains respectively). While *CPISA* was able to solve all 15 problems (Figs. 13 and 14), we found that the domain produced by RIM was not able to solve any

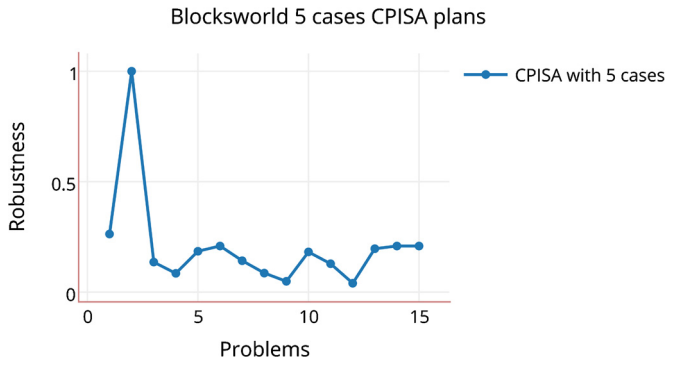


Fig. 13. Plans produced by *CPISA* on Blocksworld domain with five cases.

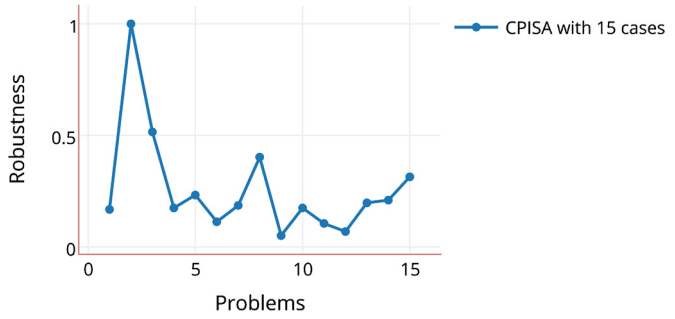


Fig. 14. Plans produced by *CPISA* on Blocksworld domain with 15 cases.

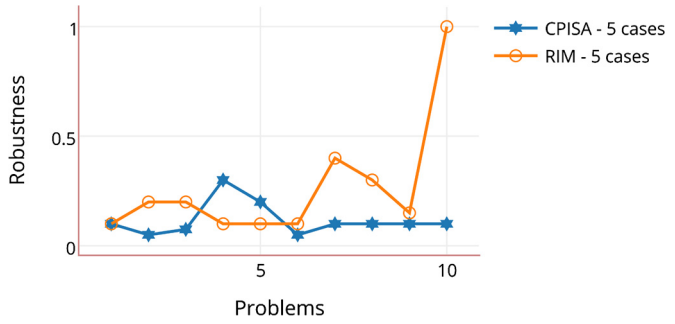


Fig. 15. Comparing plans produced by *CPISA* vs RIM provided five cases in Driverlog domain.

of the given problems. In particular, the modified model learned by RIM is still incorrect, and happened to be incorrect in such a way that all test case problems were unsolvable in that domain.[15]

For the Driverlog domain, we created an incomplete model with 8 missing preconditions and two missing effects. This maps into an annotated domain with 21 possible preconditions and 28 possible effects (5.6×10^{14} possible complete domains). We tested our approach using 10 cases against 10 random problems, with both approaches trying to solve each problem using the first 5 cases (that are consistent with approximately 1.6×10^8 domains) and then using the entire set (approximately 4.9×10^7 possible domains). As seen from Figs. 15 and 16, RIM generally outperforms *CPISA* for this particular domain.

The experiments reported above suggest that *CPISA* is able to perform better for domains with higher incompleteness and with a lower number of cases (at times even producing plans with robustness value of 1, while RIM failed to produce plans). But we see that RIM starts to perform much better as we are provided more cases. We believe that in general, the RIM approach will start performing better as more cases are provided to it, as RIM is able to learn more and more macro operators. As macro operators are action sequences that were already seen in the plan traces, using these sequences will not add any new constraints and will not reduce the robustness of a plan.

[15] The fact that none of the test cases were solvable by the model learned by RIM, while fortuitous, is by no means impossible.

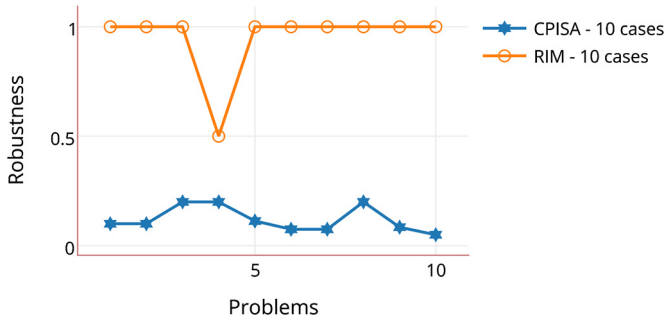


Fig. 16. Comparing plans produced by *CPISA* vs RIM provided ten cases in Driverlog domain.

Given that the relation between *CPISA* and RIM is similar to some extent to that between bayesian vs. traditional (maximum likelihood) reinforcement learning, this performance is expected to a certain extent.

10. Conclusion

This paper addresses planning problems in which the domain models are incompletely specified. We study planning situations when the domain model is annotated with possible preconditions and effects (in addition to the known preconditions and effects). The solution concept for planning problems with incomplete domain models is introduced with the notion of plan robustness. It is computed with the set of candidate complete models under which the plan succeeds, thus correctly captures the probability of plan success. This work considers two execution semantics for plans: the Generous Execution and STRIPS Execution semantics. The difference between the two semantics is how an action's failure affects plan execution. The problem of assessing plan robustness is also considered and shown to be $\#P$-complete. Two approaches are considered for synthesizing robust plans—the compilation approach and the heuristic search approach. While the first one is more intuitive, its performance appears to be limited to small planning instances only. The heuristic approach is much more scalable, and in this work we fully develop it under the STRIPS semantics, exploiting the structures of plan correctness constraints in order to approximate robustness measures. These approximations are then used during the extraction of robust relaxed plans to estimate the heuristic distance to goals. The resulting planner, *PISA*, outperforms a state-of-the-art planner, DeFault [31], in both plan quality and planning time. Finally, we also present an extension of *PISA* called *CPISA* that is able to leverage successful plan traces to further improve robustness. *CPISA* also shows how such traces can be used to reduce the dependence on the domain expert for specifying possible precondition/effect annotations.

Acknowledgements

This research is supported in part by the ARO grant W911NF-13-1-0023, and the ONR grants N00014-13-1-0176, N0014-13-1-0519 and N00014-15-1-2027. We gratefully acknowledge the helpful discussions with Minh Do, David Smith (NASA), Hankz Hankui Zhuo (Sun Yat-Sen University) and Will Cushing.

References

[1] E. Amir, A. Chang, Learning partially observable deterministic action models, J. Artif. Intell. Res. 33 (1) (2008) 349–402.
[2] B. Bonet, H. Geffner, Planning as heuristic search, Artif. Intell. 129 (1) (2001) 5–33.
[3] A. Coles, M. Fox, A. Smith, A new local-search algorithm for forward-chaining planning, in: ICAPS, 2007, pp. 89–96.
[4] S. Cresswell, T.L. McCluskey, M.M. West, Acquiring planning domain models using LOCM, Knowl. Eng. Rev. 28 (2) (2013) 195–213, http://dx.doi.org/10.1017/S0269888912000422.
[5] R. Dearden, N. Friedman, D. Andre, Model based bayesian exploration, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1999, pp. 150–159.
[6] K. Delgado, S. Sanner, L. De Barros, Efficient solutions to factored MDPS with imprecise transition probabilities, Artif. Intell. 175 (9–10) (2011) 1498–1527.
[7] C. Domshlak, J. Hoffmann, Fast probabilistic planning through weighted model counting, in: ICAPS, 2006, pp. 243–252.
[8] C. Domshlak, J. Hoffmann, Probabilistic planning via heuristic forward search and weighted model counting, J. Artif. Intell. Res. 30 (1) (2007) 565–620.
[9] R.E. Fikes, N.J. Nilsson, Strips: a new approach to the application of theorem proving to problem solving, Artif. Intell. 2 (3) (1972) 189–208.
[10] M. Fox, R. Howey, D. Long, Exploration of the robustness of plans, in: Proceedings of the National Conference on Artificial Intelligence, vol. 21, 2006, p. 834.
[11] A. Garland, N. Lesh, Plan evaluation with incomplete action descriptions, in: Proceedings of the National Conference on Artificial Intelligence, 2002, pp. 461–467.
[12] R. Givan, S. Leach, T. Dean, Bounded-parameter Markov decision processes, Artif. Intell. 122 (1–2) (2000) 71–109.
[13] C.P. Gomes, A. Sabharwal, B. Selman, Model counting: a new strategy for obtaining good bounds, in: Proceedings of the National Conference on Artificial Intelligence, vol. 21, Menlo Park, CA, AAAI Press/MIT Press, Cambridge, MA/London, 2006, 1999, p. 54.
[14] J. Hoffmann, Simulated penetration testing: from "Dijkstra" to "Turing Test++", in: Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7–11, 2015, 2015, pp. 364–372.
[15] J. Hoffmann, R.I. Brafman, Conformant planning via heuristic forward search: a new approach, Artif. Intell. 170 (6) (2006) 507–541.

[16] J. Hoffmann, B. Nebel, The FF planning system: fast plan generation through heuristic search, J. Artif. Intell. Res. 14 (2001) 253–302.
[17] R. Jensen, M. Veloso, R. Bryant, Fault tolerant planning: toward probabilistic uncertainty models in symbolic non-deterministic planning, in: Proceedings of the 14th International Conference on Automated Planning and Scheduling, ICAPS, vol. 4, 2004, pp. 235–344.
[18] S. Kambhampati, Model-lite planning for the web age masses: the challenges of planning with incomplete and evolving domain models, in: Proceedings of the National Conference on Artificial Intelligence, vol. 22, 2007, p. 1601.
[19] T.L. McCluskey, S. Cresswell, N.E. Richardson, M.M. West, Action knowledge acquisition with Opmaker2, in: Agents and Artificial Intelligence – International Conference, ICAART 2009, Porto, Portugal, January 19–21, 2009, 2009, pp. 137–150, Revised Selected Papers.
[20] T. Nguyen, S. Kambhampati, A heuristic approach to planning with incomplete strips action models, in: ICAPS, 2014.
[21] T.A. Nguyen, S. Kambhampati, M. Do, Synthesizing robust plans under incomplete domain models, in: Advances in Neural Information Processing Systems, 2013, pp. 2472–2480.
[22] T.A. Nguyen, S. Kambhampati, M.B. Do, Assessing and generating robust plans with partial domain models, in: ICAPS-10 Workshop on Planning and Scheduling Under Uncertainty, 2010.
[23] J. Rintanen, K. Heljanko, I. Niemelä, Parallel encodings of classical planning as satisfiability, in: Logics in Artificial Intelligence, 2004, pp. 307–319.
[24] J. Robertson, D. Bryce, Reachability heuristics for planning in incomplete domains, in: ICAPS'09 Workshop on Heuristics for Domain Independent Planning, 2009.
[25] T. Sang, P. Beame, H. Kautz, Heuristics for fast exact model counting, in: Theory and Applications of Satisfiability Testing, Springer, 2005, pp. 226–240.
[26] J. Satia, R. Lave Jr, Markovian decision processes with uncertain transition probabilities, Oper. Res. (1973) 728–740.
[27] R.M. Simpson, D.E. Kitchin, T.L. McCluskey, Planning domain definition using GIPO, Knowl. Eng. Rev. 22 (2) (2007) 117–134.
[28] M. Strens, A bayesian framework for reinforcement learning, in: ICML, 2000, pp. 943–950.
[29] L. Valiant, The complexity of computing the permanent, Theor. Comput. Sci. 8 (2) (1979) 189–201.
[30] L.G. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (3) (1979) 410–421.
[31] C. Weber, D. Bryce, Planning and acting in incomplete domains, in: Proceedings of ICAPS'11, 2011.
[32] W. Wei, B. Selman, A new approach to model counting, in: Theory and Applications of Satisfiability Testing, Springer, 2005, pp. 96–97.
[33] C. White III, H. Eldeib, Markov decision processes with imprecise transition probabilities, Oper. Res. (1994) 739–749.
[34] Q. Yang, K. Wu, Y. Jiang, Learning action models from plan examples using weighted max-sat, Artif. Intell. 171 (2) (2007) 107–143.
[35] H.H. Zhuo, S. Kambhampati, T. Nguyen, Model-lite case-based planning, in: Proceedings of the 27th AAAI Conference on Artificial Intelligence, 2013.
[36] H.H. Zhuo, T. Nguyen, S. Kambhampati, Refining incomplete planning domain models through plan traces, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, AAAI Press, 2013, pp. 2451–2457.
[37] H.H. Zhuo, Q. Yang, Action-model acquisition for planning via transfer learning, Artif. Intell. 212 (2014) 80–103.