A Heuristic Approach to Planning with Incomplete STRIPS Action Models

Tuan A. Nguyen and **Subbarao Kambhampati**
School of Computing, Informatics and Decision System Engineering, Arizona State University
{natuan,rao}@asu.edu

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 STRIPS 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 introduce two approximations, lower and upper bound, for plan robustness, and then utilize them to derive heuristics for synthesizing robust plans. Our planning system, PISA, incorporating stochastic local search with these novel techniques outperforms a state-of-the-art planner handling incomplete domains in most of the tested domains, both in terms of plan quality and planning time.

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 (c.f. (Kambhampati 2007)) 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 don't have the luxury of delaying their decision making. For example, although there exist efforts (Amir and Chang 2008; Yang, Wu, and Jiang 2007) 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 & Lesh (2002) 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.

Nguyen *et al.* (2010) propose annotations specifying *possible* preconditions and effects of actions. These annotations facilitate, though only conceptually, the enumeration of all "candidate" complete models, thus the counting of models under which a plan succeeds. Their *robustness* measure for plans, therefore, captures exactly the probability of success for plans given such an incompleteness language. Weber and Bryce (2011) use the same formulation 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.

In this paper, we propose a principled heuristic approach to synthesizing robust plans for incomplete STRIPS action models with possible precondition and effect annotations. Our idea is to use the robustness measure more directly in guiding the search. We show however that there is an immediate technical hurdle: the problem of assessing the robustness of a given plan is equivalent to weighted model

counting, and is thus $\#P$-complete. We then exploit two interesting structures in the logical constraints on plan correctness proposed by Nguyen *et al.* (2010), introducing the lower and upper approximations for the robustness for a given plan. We incorporate these approximations in the extraction of robust relaxed plans and guiding the search, resulting in a comprehensive approach for synthesizing robust plans. The experiments show that our planner, PISA (**P**lanning with **I**ncomplete **S**TRIPS **A**ctions), outperforms DeFault in most of the tested domains, both in terms of plan robustness and in planning time.

The paper is organized as follows. Section 2 formulates the planning problems with incomplete STRIPS action models. In Section 3, we discuss the robustness measure for plans, and then a method to assess the plan robustness measure using weighted model counting. We also show the complexity of the plan robustness assessment problem in this section. Section 4 presents our approach to synthesizing robust plans, which includes our method to approximate plan robustness and our procedure for extracting robust relaxed plans. Section 5 presents the experiment results. We discuss the related work in Section 6 and conclude our work in Section 7.

2 Problem Formulation

We define an incomplete STRIPS 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 STRIPS 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 $w_o^{pre}(r)$, $w_o^{add}(r)$ 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 STRIPS domain $\widetilde{\mathcal{D}}$, 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.

A planning problem $\widetilde{\mathcal{P}}$ with respect to an incomplete domain $\widetilde{\mathcal{D}}$ 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 set of propositions that are *true*, or **T**, in the initial state (and all the remaining are *false*, or **F**), and $G \subseteq F$ is the set of goal propositions. Note that in the same complete model $\mathcal{D}$, actions instantiated from the same operator have the same realized preconditions and effects.

An action $a \in A$ is *applicable* in a state $s \subseteq F$ if $Pre(a) \subseteq s$, and the resulting state is defined as $\gamma(\langle a \rangle, s) = (s \setminus Del(a)) \cup Add(a) \cup \widetilde{Add}(a)$. The projection of an action sequence $\pi = \langle a_1, ..., a_n \rangle$ from s is defined as $\gamma(\pi, s) = \gamma(a_n, \gamma(a_{n-1}, ..., \gamma(a_1, s)))$. The sequence π is a *valid plan* for a planning problem $\widetilde{\mathcal{P}}$ if $\gamma(I, \pi) \supseteq G$. Note that we neglect $\widetilde{Pre}(a)$ in applicability checking condition and $\widetilde{Del}(a)$ in creating the resulting state to ensure the *completeness* (that is, any plan achieving goals G in the ground truth model $\mathcal{D}^*$ is not excluded), and the *soundness* (in the sense that any valid plan for $\widetilde{\mathcal{P}}$ is a plan achieving G in at least one complete model $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$). To recall, the transition function under a complete model $\mathcal{D}$ is defined with STRIPS semantics: for the complete specification of action $a \in A$ in $\mathcal{D}$, $a^{\mathcal{D}}$, and a state $s \subseteq F$: $\gamma^{\mathcal{D}}(\langle a^{\mathcal{D}} \rangle, s) = (s \setminus Del(a^{\mathcal{D}})) \cup Add(a^{\mathcal{D}})$ if $Pre(a^{\mathcal{D}}) \subseteq s$ (otherwise, undefined); for $\pi^{\mathcal{D}} = \langle a_1^{\mathcal{D}}, ..., a_n^{\mathcal{D}} \rangle$, $\gamma^{\mathcal{D}}(\pi^{\mathcal{D}}, s) = \gamma^{\mathcal{D}}(a_n^{\mathcal{D}}, \gamma^{\mathcal{D}}(a_{n-1}^{\mathcal{D}}, ..., \gamma^{\mathcal{D}}(a_1^{\mathcal{D}}, s)))$. Action sequence $\pi^{\mathcal{D}}$ is a plan achieving G under $\mathcal{D}$ if $G \subseteq \gamma^{\mathcal{D}}(\pi^{\mathcal{D}}, I)$.

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.

3 A Robustness Measure for Plans

The robustness of a plan π for the problem $\widetilde{\mathcal{P}} = \langle F, A, I, G \rangle$ is defined as the cumulative probability mass of the completions of $\widetilde{\mathcal{D}}$ under which π succeeds in achieving the goals (Nguyen *et al.*, 2010). More formally, let $Pr(\mathcal{D})$ be the probability distribution representing the modeler's estimate of the probability that a given model $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$ is the ground truth model $\mathcal{D}^*$; $\sum_{\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle} Pr(\mathcal{D}) = 1$. The robustness of π is defined as follows:

$$R(\pi, \widetilde{\mathcal{P}}) \stackrel{def}{=} \sum_{\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle, \gamma^{\mathcal{D}}(\pi^{\mathcal{D}}, I) \models G} Pr(\mathcal{D}). \quad (1)$$

When the context is clear, we write $R(\pi)$ for the robustness of plan π. 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 $w_o^{pre}(r)$, $w_o^{add}(r)$, and $w_o^{del}(r)$ for all $o \in \mathcal{O}$ and its possible preconditions/effects r if r *is* realized in $\mathcal{D}$ (or the prod-

[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.

uct of their "complement" $1 - w_o^{pre}(r)$, $1 - w_o^{add}(r)$, and $1 - w_o^{del}(r)$ if r is *not* realized).

3.1 Plan Robustness Assessment as Weighted Model Counting

The notion of plan robustness as the cummulative probability mass of the complete models under which plans succeed draw the connection to the computation of the weighted model count of a logical formula. For the completeness of our presentation,[3] we now present the set of constraints for the correctness of a plan $\pi = \langle a_1, ..., a_n \rangle$, introduced by Nguyen *et al.* (2010), whose weighted model count provides its plan robustness. We then show that plan robustness assessment is indeed as hard as model counting problems. In the following, $a_0 \equiv a_I$ and $a_{n+1} \equiv a_G$ are two special complete actions: $Add(a_I) = I$, $Del(a_I) = F \setminus I$, $Pre(a_G) = G$, $Add(a_G) = \{\mathbf{T}\}$, the other components are empty. Plan π succeeds in achieving G if and only if all actions in $\langle a_1, ..., a_n, a_{n+1} \rangle$ are executable.

Variables: The variables used in the constraints are those 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 $\widetilde{Del}(o)$. An assignment of these K variables corresponds to a complete domain $\mathcal{D} \in \langle\langle \widetilde{\mathcal{D}} \rangle\rangle$, and those assignments satisfying the following constraints determine the complete models under which the plan succeeds. 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.

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 $\mathbf{T}$ if $p \in Pre(a_{C_p^i})$ or $p \in Add(a_{C_p^i - 1})$; and confirmed $\mathbf{F}$ 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 $\mathbf{T}$ 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:

$$\bigvee_{C_p^i \leq k \leq i-1, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}. \tag{2}$$

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:

$$p_{a_m}^{del} \Rightarrow \bigvee_{m < k < i, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}, \tag{3}$$

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 $\mathbf{T}$ and protected. Specifically, for $p = \mathbf{F}$ at level C_p^i, we add the constraints:

$$p_{a_i}^{pre} \Rightarrow \bigvee_{C_p^i \leq k \leq i-1, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}. \tag{4}$$

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

$$p_{a_i}^{pre} \Rightarrow \left(p_{a_m}^{del} \Rightarrow \bigvee_{m < k < i, p \in \widetilde{Add}(a_k)} p_{a_k}^{add} \right). \tag{5}$$

We denote the set of constraints established for π as $\mathbf{\Sigma}_\pi$. 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 $\mathbf{\Sigma}_\pi$ is therefore the robustness of the plan. Throughout the paper, we use $R(\pi)$ and $\text{WMC}(\mathbf{\Sigma}_\pi)$ interchangeably.

3.2 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 1. *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.*

Proof (sketch). The membership can be seen by having a Counting TM 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 $\langle X, \mathbf{\Sigma} \rangle$ of MONOTONE 2-SAT, a $\#P$-complete problem (Valiant 1979), to an instance $\langle \widetilde{\mathcal{D}}, \widetilde{\mathcal{P}}, \pi \rangle$ such that $R(\pi) = \frac{|\mathcal{M}(\mathbf{\Sigma})|}{2^n}$. The input of this problem is a set of n Boolean variables $X = \{x_1, ..., x_n\}$, a monotone CNF formulae $\mathbf{\Sigma} = \{c_1, c_2, ..., c_m\}$ with m clauses $c_j = x_{j1} \vee x_{j2}$, where $x_{j1}, x_{j2} \in X$. The required output is $|\mathcal{M}(\mathbf{\Sigma})|$, the number of assignments of X satisfying $\mathbf{\Sigma}$.

Let $\mathcal{G} = \langle V, E \rangle$ be the constraint graph of the clause set $\mathbf{\Sigma}$, where $V = X$ and $(x_i, x_j) \in E$ if $x_i \vee x_j \in \mathbf{\Sigma}$. We partition this graph into connected components (or subgraph) $\mathcal{G}_1, ..., \mathcal{G}_l$. Our set of propositions F then includes

[3] In Section 4, we will use these constraints, together with our approximation measures, during the synthesis of robust plans.

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 $\widetilde{\mathcal{D}} = \langle F, A \rangle$,[4] 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 $R(\pi) = \frac{|\mathcal{M}(\mathbf{\Sigma})|}{2^n}$ can be established by verifying that an assignment σ satisfies $\mathbf{\Sigma}$ if and only if π succeeds under the corresponding complete model $\mathcal{D}_\sigma$. □

4 Synthesizing Robust Plans

In this section, we present an anytime forward search approach to synthesizing robust plans. 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.

The main technical part of our approach, therefore, 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 $\text{WMC}(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}}) > \delta$. 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 $l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}}) > \delta$, and compute the exact robustness of a candidate plan π only if $u(\mathbf{\Sigma}_\pi) > \delta$.

In this section, we first introduce our lower and upper bound for $\text{WMC}(\mathbf{\Sigma})$ of a clause set $\mathbf{\Sigma}$, presents our procedure for extracting relaxed plan, and then discuss our choice for the underlying search algorithm.

4.1 Approximating Weighted Model Count

Lower bound: We observe that the set of constraints $\mathbf{\Sigma}$, constructed as in (2)-(5), 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 $l_{\mathbf{\Sigma}} = \prod_{c_i} Pr(c_i)$ can be used as a lower bound for $\text{WMC}(\mathbf{\Sigma})$, 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 2. *Given a set of monotone clauses* $\mathbf{\Sigma} = \{c_1, ..., c_k\}$, $l_{\mathbf{\Sigma}} = \prod_{c_i \in \mathbf{\Sigma}} Pr(c_i) \leq \textit{WMC}(\mathbf{\Sigma})$.

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, $\text{WMC}(\mathbf{\Sigma}) = Pr(\mathbf{\Sigma}) = Pr(c_1)Pr(c_2|c_1)...Pr(c_k|c_1 \wedge ... \wedge c_{k-1}) \geq \prod_{c_i} Pr(c_i)$. □

Upper bound: One trivial upper bound for $Pr(\mathbf{\Sigma})$ is $\min_{c_i} Pr(c_i)$. We can however derive a much tighter bound for $\text{WMC}(\mathbf{\Sigma})$ by observing that literals representing the realization of preconditions and effects on different predicates would *not* be present in the same clauses. This suggests that the set of clauses $\mathbf{\Sigma}$ 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 $Pr(\mathbf{\Sigma})$, we first divide it into independent clause sets $\mathbf{\Sigma}^1, ..., \mathbf{\Sigma}^m$, and compute an upper bound $u_{\mathbf{\Sigma}}$ as follows: $u_{\mathbf{\Sigma}} = \prod_{\mathbf{\Sigma}^i} \min_{c \in \mathbf{\Sigma}^i} Pr(c)$.

4.2 Extracting Robust Relaxed Plan

We now introduce our procedure to extract a relaxed plan $\widetilde{\pi}$ such that $l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}}) > \delta$, where $\mathbf{\Sigma}_{\pi_k}$ and $\mathbf{\Sigma}_{\widetilde{\pi}}$ are the sets of constraints for the executability of actions in π_k and $\widetilde{\pi}$; note that $\mathbf{\Sigma}_{\widetilde{\pi}}$ also includes constraints for a_G, thus for the achievement of G. Our procedure employs an extension of the common relaxation technique by ignoring both the known and possible delete effects of actions in constructing $\widetilde{\pi}$.[5]

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 $\mathbf{\Sigma}_p(t)$ and $\mathbf{\Sigma}_a(t)$.

- $L_1 \equiv s_{k+1}$, where $s_{k+1} = \gamma(\pi_k, I)$. The clause set $\mathbf{\Sigma}_p(1)$ for each $p \in L_1$, constructed with Constraints (2) and (3), 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 $p \in L_t$: $Pre(\textit{noop}_p) = Add(\textit{noop}_p) = \{p\}$, $Del(\textit{noop}_p) = \emptyset$. The constraints for the non-*noop* actions:

$$\mathbf{\Sigma}_a(t) = \bigwedge_{p \in Pre(a)} \mathbf{\Sigma}_p(t) \wedge \bigwedge_{q \in \widetilde{Pre}(a)} (q_a^{pre} \Rightarrow \mathbf{\Sigma}_q(t)). \quad (6)$$

All actions *noop* have the same constraints with their corresponding propositions.

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

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

- 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., $\mathbf{\Sigma}_{\pi_k}$. In particular:

$$\mathbf{\Sigma}_p(t+1) = \arg\max_{\mathbf{\Sigma} \in S_1 \cup S_2} l(\mathbf{\Sigma} \wedge \mathbf{\Sigma}_{\pi_k}), \tag{7}$$

where $S_1 = \{\mathbf{\Sigma}_a(t) \mid p \in Add(a)\}$ and $S_2 = \{p^{add}_{a'} \wedge \mathbf{\Sigma}_{a'}(t) \mid p \in \widetilde{Add}(a')\}$.[6]

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) $l(\mathbf{\Sigma}_G) > \delta$, where $\mathbf{\Sigma}_G$ 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 $l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}}) > \delta$. 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 $l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}})$, 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.[7] This is when the clause sets propagated in the relaxed planning graph play their role. In particular, we reuse the clause sets $\mathbf{\Sigma}_p(t)$ 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 2-5 for (possibly) supported known and possible preconditions, resulting in constraints $\mathbf{\Sigma}_{\widetilde{\pi}}$. Together with $\mathbf{\Sigma}_{\pi_k}$, it is used to define the robustness of the relaxed plan being constructed.

Algorithm 1 shows the details of our relaxed plan extraction procedure. We initialize the relaxed plan $\widetilde{\pi}$ with $\langle a_G \rangle$,

Algorithm 2: CheckPlan

```
1  Input: Plan prefix π_k, threshold δ ∈ [0,1], goals G.
2  Output: r ≡ R(π_k) if success, or nothing if failure.
3  begin
4      if G ⊆ s_{k+1} then
5          Construct Σ_π̃ using Constraints (2)-(5).
6          if u(Σ_{π_k} ∧ Σ_π̃) > δ then
7              Compute R(π_k) = WMC(Σ_{π_k} ∧ Σ_π̃).
8              if R(π_k) > δ then
9                  r ← R(π_k); return success.
10             end
11         end
12     end
13     return failure
14 end
```

Algorithm 3: InitializeConstraintsAndQueue

```
1  Input: Goals G, relaxed planning graph G.
2  Output: ⟨Σ_π̃, Q⟩
3  begin
4      Σ_π̃ ← T.
5      for g ∈ G \ s^+_{→a_G} do
6          if g ∈ s_{→a_G} then
7              c ← Constraints (2)-(5) for g.
8              Σ_π̃ ← Σ_π̃ ∧ c.
9          end
10         else
11             Σ_π̃ ← Σ_π̃ ∧ Σ_g(T).
12         end
13         Push ⟨g, a_G, T⟩ into Q.
14     end
15     return ⟨Σ_π̃, Q⟩.
16 end
```

the *relaxed plan state* $s_{\rightarrow a_G}$ before a_G and the set of propositions $s^+_{\rightarrow a_G}$ known to be $\mathbf{T}$ (Line 4). The set s^+_{k+1} contains propositions p known to be $\mathbf{T}$ after executing all actions in π_k—they are confirmed to be $\mathbf{T}$ at some step $C^{k+1}_p < k+1$, and not being possibly deleted by any other actions at steps after C^{k+1}_p. 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 $u(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}})$ to prevent the exact weighted model counting from being called unnecessarily.

Line 8 initializes the constraints $\mathbf{\Sigma}_{\widetilde{\pi}}$ and the queue Q of subgoals.[8] 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).

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

[6] Note that the observation about converting constraints into monotone clauses, which faciliates our lower bound computation, still holds for constraints in (6) and (7).

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

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

(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 7 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 don't 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),[9] the constraints $\mathbf{\Sigma}_{\widetilde{\pi}}$ (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 $\mathbf{\Sigma}_{\widetilde{\pi}}$ 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 newly inserted action into the subgoal queue Q, ignoring those known to be $\mathbf{T}$ (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).

4.3 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 $s_{k+1} = \gamma(\pi_k, I)$ 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 (Hoffmann and Brafman 2006) and Probabilistic-FF (Domshlak and Hoffmann 2006). 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 proposed by Coles, Fox and Smith (2007). 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 (Coles, Fox, and Smith 2007) starting from $s_{k+1} = \gamma(\pi_k, I)$; 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.

5 Experimental Results

We test our planner, PISA, with incomplete domains 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 domains for each IPC domain mentioned above. We test our planner with the first 10 planning problems in each domain, thus 2150 *instances* (i.e., a pair of incomplete domain 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 domains 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 @ 2.83GHz. For exact model counting, we use Cachet model counting software (Sang *et al.*, 2005). For the search in PISA, we use the similar configuration in (Coles, Fox, and Smith 2007), 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, 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:

[9] 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 $s^+_{\rightarrow a_{j+1}} \leftarrow s^+_{\rightarrow a_j} \cup Add(a_j)$.

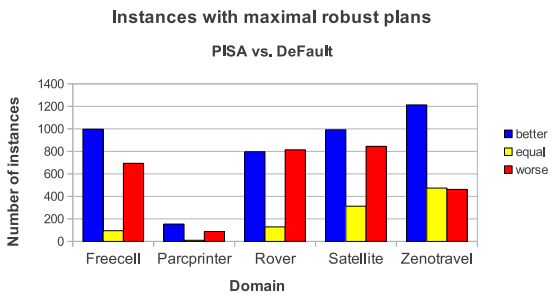


Figure 1: Number of instances for which PISA produces better, equal and worse robust plans than DeFault.

prime implicant heuristics with size $k = 2$ and 2GB RAM as suggested, running it in the anytime mode.

Figure 1 shows the number of instances for which PISA generates plans having better, equal and worse **robustness** values compared to DeFault.[10] 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 retured 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.[11] To demonstrate this, we first consider instances for which the two approaches return best plans with the *same* robustness values. Figure 2 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 654.7x 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 29.6x for Parcprinter, 103 vs. 10 (91.1%) and 1665x for Rover, 281 vs. 28 (90.9%) and 562.7x for Satellite, 329 vs. 86 (79.3%) and 482.9x 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 percentanges 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 $\gamma(\pi, I) \supseteq G$) 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 (Hoffmann and Nebel 2001). 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 $l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\tilde{\pi}})$ 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, 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%.

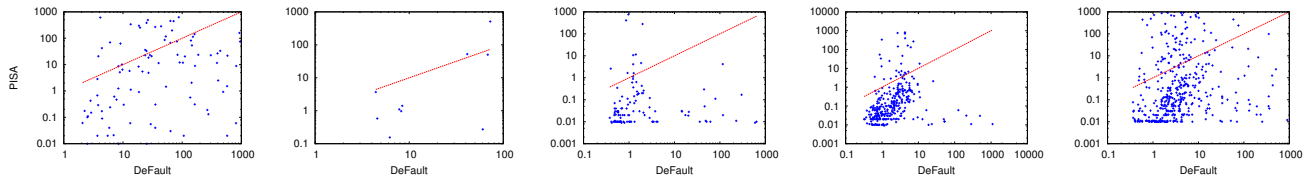
In the second baseline approach, we use exact model counting WMC($\mathbf{\Sigma}$) during the extraction of relaxed plans, replacing the approximation $l(\mathbf{\Sigma})$. This approach, as anticipated, spends most of the running time for the exact model counting. The results are discouraging, thus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

6 Related Work

There are currently very few research efforts in automated planning literature that explicitly consider incompletely specified domain models. Garland and Lesh (2002) were the first discussing incomplete actions and generating robust plans under incomplete domain models. Their notion of plan robustness, however, only has tenuous heuristic connections with the likelihood of successful execution of plans. As mentioned earlier, Weber & Bryce (2011) present a heuristic

[10] 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.

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



(a) Freecell (b) Parcprinter (c) Rover (d) Satellite (e) Zenotravel

Figure 2: 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.

state space planner, DeFault, for synthesizing robust plans. While there are some surface similarities between our relaxed plan extraction techniques (e.g. like us, they also propagate logical formulas for propositions over the planning graph), PISA's extraction procedure is more effective in assessing the robustness of the candidate extensions. In contrast, Default primarily uses the FF heuristic, with robustness considerations used only for breaking ties during the relaxed plan extraction. Empirical results do demonstrate that PISA's approach results in significantly better performance. Zhuo *et al.* (2013a; 2013b) considers planning and learning problems with action models known to be incomplete but without any annotations on the incompleteness; they assume instead a set of successful plan cases. The work by Fox *et al.* (2006) also explores robustness of plans, but their focus is on temporal plans under unforeseen execution-time variations.

The work by Jensen *et al.* (2004) on *k-fault* plans for non-deterministic planning focused on reducing the "faults" in plan execution. It is however based on the context of stochastic/non-deterministic actions rather than incompletely specified ones. In Markov Decision Processes (MDPs) and decision-theoretic planning, a fairly rich body of work has been done for *imprecise* transition probabilities (which essentially define models of actions in a domain) and seeking for max-min or min-max optimal policies, assuming that Nature acts optimally against the agent (Satia and Lave Jr 1973; White III and Eldeib 1994; Givan, Leach, and Dean 2000; Nilim and Ghaoui 2005; Delgado, Sanner, and De Barros 2011).

7 Conclusion and Future Work

This paper addresses the planning problem with incomplete STRIPS action models where annotations are provided for possible preconditions and effects. We show that the problem of assessing plan robustness is $\#P-$complete, and propose an heuristic approach to generating robust plans. Our approach utilizes the structure of plan correctness constraints to derive the lower and upper bounds for the plan robustness, which are then used to effectively extract robust relaxed plans. Our planner outperforms a state-of-the-art planner handling incomplete domains, in most of the tested domains, both in terms of plan quality and planning time. We are developing planning approaches in scenarios where both annotations and successful traces are also available.

Acknowledgements: This research is supported in part by the ARO grant W911NF-13-1-0023, and the ONR grants N00014-13-1-0176 and N0014-13-1-0519.

Algorithm 1: Extracting robust relaxed plan.

Input: A threshold $\delta \in [0,1]$, a plan prefix π_k, its correctness constraints $\mathbf{\Sigma}_{\pi_k}$, the relaxed planning graph $\mathcal{G}$ of T layers.
Output: $\langle h(\pi_k,\delta), r\rangle$ if *success*, or nothing if *failure*.
begin
  $\widetilde{\pi} \leftarrow \langle a_G\rangle$, $s_{\rightarrow a_G} \leftarrow L_1$, $s^+_{\rightarrow a_G} \leftarrow s^+_{k+1}$.
  if $r \leftarrow$ *CheckPlan*(π_k,δ,G) *succeeds* **then**
    return $\langle 0, r\rangle$, *success*.
  end
  $\langle \mathbf{\Sigma}_{\widetilde{\pi}}, Q\rangle \leftarrow$ InitializeConstraintsAndQueue$(G,\mathcal{G})$.
  $r \leftarrow l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}})$.
  while Q *not empty* **do**
    Pop $\langle p, a, t\rangle$ from Q s.t. $p \in L_t$.
    $\langle a_{\text{best}}, l_{\text{best}}\rangle \leftarrow$ best supporting action and its layer in $\mathcal{G}$.
    if $l_{best} = 0$ **or** $\widetilde{\pi}$ *contains* $\langle a_{best}, l_{best}\rangle$ **then**
      continue.
    end
    if $p \in Pre(a)$ **and** $p \notin s_{\rightarrow a}$ **then**
      Insert $\langle a_{\text{best}}, l_{\text{best}}\rangle$ into $\widetilde{\pi}$.
    end
    else
      $r' \leftarrow$ evaluate$(a_{\text{best}}, l_{\text{best}}, \widetilde{\pi})$.
      if $r' > r$ **then**
        Insert $\langle a_{\text{best}}, l_{\text{best}}\rangle$ into $\widetilde{\pi}$.
      end
    end
    if $\langle a_{best}, l_{best}\rangle$ *inserted into* $\widetilde{\pi}$ **then**
      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 $\widetilde{\pi}$.
      Update $\mathbf{\Sigma}_{\widetilde{\pi}}$.
      $r \leftarrow l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}})$.
      count $\leftarrow$ number of unsupported known preconditions in $\widetilde{\pi}$.
      if *count* $= 0$ **and** $r > \delta$ **then**
        return $\langle |\widetilde{\pi}| - 1, r\rangle$, *success*.
      end
      for $q \in Pre(a_{best}) \cup \widetilde{Pre}(a_{best}) \setminus s^+_{\rightarrow a_{best}}$ **do**
        Push $\langle q, a_{\text{best}}, l_{\text{best}}\rangle$ into Q.
      end
    end
  end
  if $r > \delta$ **then return** $\langle |\widetilde{\pi}| - 1, r\rangle$, *success*.
  return *failure*.
end

References

Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research* 33(1):349–402.

Coles, A.; Fox, M.; and Smith, A. 2007. A new local-search algorithm for forward-chaining planning. In *ICAPS*, 89–96.

Delgado, K.; Sanner, S.; and De Barros, L. 2011. Efficient solutions to factored mdps with imprecise transition probabilities. *Artificial Intelligence*.

Domshlak, C., and Hoffmann, J. 2006. Fast probabilistic planning through weighted model counting. In *ICAPS*, 243–252.

Fox, M.; Howey, R.; and Long, D. 2006. Exploration of the robustness of plans. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, 834.

Garland, A., and Lesh, N. 2002. Plan evaluation with incomplete action descriptions. In *Proceedings of the National Conference on Artificial Intelligence*, 461–467.

Givan, R.; Leach, S.; and Dean, T. 2000. Bounded-parameter markov decision processes. *Artificial Intelligence* 122(1-2):71–109.

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

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

Jensen, R.; Veloso, M.; and Bryant, R. 2004. 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)*, volume 4, 235–344.

Kambhampati, S. 2007. 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*, volume 22, 1601.

Nguyen, T. A.; Kambhampati, S.; and Do, M. B. 2010. Assessing and generating robust plans with partial domain models. In *ICAPS-10 Workshop on Planning and Scheduling Under Uncertainty*.

Nilim, A., and Ghaoui, L. 2005. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research* 53(5):780–798.

Sang, T.; Beame, P.; and Kautz, H. 2005. Heuristics for fast exact model counting. In *Theory and Applications of Satisfiability Testing*, 226–240. Springer.

Satia, J., and Lave Jr, R. 1973. Markovian decision processes with uncertain transition probabilities. *Operations Research* 728–740.

Valiant, L. G. 1979. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8(3):410–421.

Weber, C., and Bryce, D. 2011. Planning and acting in incomplete domains. *Proceedings of ICAPS11*.

White III, C., and Eldeib, H. 1994. Markov decision processes with imprecise transition probabilities. *Operations Research* 739–749.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2):107–143.

Zhuo, H. H.; Kambhampati, S.; and Nguyen, T. 2013a. Model-lite case-based planning. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*.

Zhuo, H. H.; Nguyen, T.; and Kambhampati, S. 2013b. Refining incomplete planning domain models through plan traces. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 2451–2457. AAAI Press.