Planning with Incomplete User Preferences and Domain Models

by

Tuan Anh Nguyen

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

Approved May 2014 by the
Graduate Supervisory Committee:

Subbarao Kambhampati, Chair
Chitta Baral
Minh Do
Joohyung Lee
David E. Smith

ARIZONA STATE UNIVERSITY

August 2014

ABSTRACT

Current work in planning assumes that user preferences and/or domain dynamics are completely specified in advance, and aims to search for a single solution plan to satisfy these. In many real world scenarios, however, providing a complete specification of user preferences and domain dynamics becomes a time-consuming and error-prone task. More often than not, a user may provide no knowledge or at best partial knowledge of her preferences with respect to a desired plan. Similarly, a domain writer may only be able to determine certain parts, not all, of the model of some actions in a domain. Such modeling issues requires new concepts on what a solution should be, and novel techniques in solving the problem.

When user preferences are incomplete, rather than presenting a single plan, the planner must instead provide a set of plans containing one or more plans that are similar to the one that the user prefers. This research first proposes the usage of different measures to capture the quality of such plan sets. These are domain-independent distance measures based on plan elements if no knowledge of the user preferences is given, or the Integrated Preference Function measure in case incomplete knowledge of such preferences is provided. It then investigates various heuristic approaches to generate plan sets in accordance with these measures, and presents empirical results demonstrating the promise of the methods.

The second part of this research addresses planning problems with incomplete domain models, specifically those annotated with possible preconditions and effects of actions. It formalizes the notion of plan robustness capturing the probability of success for plans during execution. A method of assessing plan robustness based on the weighted model counting approach is proposed. Two approaches for synthesizing robust plans are introduced. The first one compiles the robust plan synthesis problems to the conformant probabilistic planning problems. The second approximates the robustness measure with lower and upper bounds, incorporating them into a stochastic local search for estimating distance heuristic

to a goal state. The resulting planner outperforms a state-of-the-art planner that can handle incomplete domain models in both plan quality and planning time.

ACKNOWLEDGEMENTS

I would like to express my wholehearted gratitude to my advisor, Professor Subbarao Kambhampati, for his continuous support, technical instructions, advice, and especially his encouragement and patience during the hard times of my study. I deeply thank him for spending his valuable time for me, helping me clarify my thoughts and many times guiding me towards a broader view on my research problems.

I would like to thank Dr. Minh Do for not only supporting me in my research from the beginning, but also sharing with me his advice and experiences on anything I need. I thank him and his wife for always welcoming me and making me feel warm with their family.

I would like to thank Professor Chitta Baral, Dr. Minh Do, Professor Joohyung Lee and Dr. David Smith for serving in my Dissertation Committee, helping me with valuable discussions and comments.

I gratefully thank my friends in the Yochan group for their support and friendship. To name but a few, they are J. Benton, William Cushing, Menkes van den Briel, Raju Balakrishnan, Kartik Talamadupula, Sushovan De, Yuheng Hu. I especially thank J. for always helping me when I was in difficult time. I will also miss the time I had with my friends in the Vinasu group during my time here at ASU.

I would like to thank the Science Foundation of Arizona (SFAz) for supporting me during the first two years of my study.

Finally, I would like to thank my wife and my son for the wonderful time I am having with them. My every day with them has been a true gift!

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

OVERVIEW

In the past several years, significant strides have been made in scaling up plan synthesis techniques. There are technology now to routinely generate plans with hundreds of actions. A significant amount of ongoing work in the community has been directed at building on these advances to provide efficient synthesis techniques under a variety of more expressive formulations on user preferences and domain dynamics. All this work however makes a crucial assumption—that a complete specification of user preferences and/or domain dynamics is given in advance. While there may be scenarios where a user can exactly describe her preferences on desired plans (e.g., a passenger searching for flights between two close cities is very likely interested only in the cost of tickets), very often she can at best partially specify her desires on solution plans in many other situations. Similarly, while knowledge-engineering a detailed domain model is necessary as well as feasible in some applications (e.g., mission planning domains in NASA and factory-floor planning), completely modeling the domain dynamics can be very time-consuming and error-prone in many real-life applications. It is increasingly recognized (c.f. Kambhampati, 2007) that there are also many scenarios where insistence on complete models renders the current planning technology unusable. Such incompleteness modeling issues requires new concepts on what solutions of a planning problem should be, and novel techniques in solving the problem.

The topic of this dissertation is about "model-lite" planning—planning with incomplete user preferences and domain models. While the solution concept for a planning problem is clearly understood when complete user preferences and domain models are available,[1] it is

[1] In particular, it is the single *best* plan with respect to the user known preferences, and it is any *valid* plan which reaches a state satisfying all goals from an initial state given a complete domain model.

not even obvious what a reasonable concept for solutions of a planning problem should be if the planner does not possess such complete specifications, let alone how to find them efficiently. The contribution of this research, therefore, is first to propose solution concepts for planning problems in those scenarios, and second to devise techniques for finding solutions of those problems.

Planning with incomplete user preferences The first part of the dissertation describes the work on planning when user preferences on plans are incomplete, in particular either *completely unknown* or *partially specified*. [2] In such scenarios, the planner's job changes from finding a single plan to finding a set of *representative* plans and presenting them to the user (in the hope that she will find one of them desirable). Quality measures for plan sets, as a result, should be defined to evaluate plan sets with respect to the amount of knowledge about user preferences; and techniques need to be devised to generate high quality plan sets.

- Firstly, when the user is unable to provide any knowledge of her preferences, the *diversity* measures for plan sets are defined based on distance measures between two plans using various syntactic features of plans. The intuition is that by maximizing the diversity of a plan set, the set is more likely to be uniformly distributed in the unknown space of preferences, and thus the chance that one of them is close the plan that she is interested in is increased. To synthesize plan sets with these diversity measures, two representative planning approaches, GP-CSP (Do and Kambhampati, 2001) and LPG (Gerevini *et al.*, 2003) are discussed, respectively typifying the issues involved in generating diverse plans in bounded horizon compilation and heuristic search based planning methods.

- Secondly, this research is interested in situations where the user can only be cer-

[2] This work has been published in (Srivastava *et al.*, 2007; Nguyen *et al.*, 2009). A unified and comprehensive study can be found in (Nguyen *et al.*, 2012), which contains most of the first part of this dissertation.

tain about part of her preferences, but not all. In particular, the assumption is that the user can express a set of plan attributes that she wants to optimize, however does not know how to correctly determine the degree of their relative importance. The idea of *Integrated Preference Function* (IPF) (Carlyle *et al.*, 2003) developed in Operations Research (OR) community in the context of multi-criteria scheduling is adapted, which measures the expected reward or penalty that a user will get or pay by choosing one solution in a given set. A spectrum of approaches for generating plan sets with this measure is presented, which are implemented on top of Metric-LPG (Gerevini *et al.*, 2008), a state-of-the-art planner that can effectively handle metric planning problems.

Planning with incomplete domain models The second part of the dissertation describes the work on planning with incomplete domain models. Following Garland & Lesh (2002), it assumes that 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 the framework of this work, these annotations consist of allowing actions to have *possible* preconditions and effects (in addition to the standard necessary preconditions and effects). Incomplete domain models with such possible preconditions and effects implicitly define an exponential set of complete domain models, with the semantics that the real domain model is guaranteed to be one of these.

Given such an incomplete domain model, a plan generated cannot be guaranteed to succeed during execution; all one can say is that a plan with higher chance to achieve the goals should be considered better. In order to quantify the quality of plans, a *robustness* measure for plans is introduced to capture the probability of their success during execution. Two execution semantics for plans are considered which differs in how an action failure is treated during execution. In the first one, Generous Execution semantics, the user continues to execute the plan even if one of the actions in the plan fails to apply. The second semantics

following STRIPS-style planning (Fikes and Nilsson, 1972) considers the plan failed when one of its actions fails to apply.

Two approaches are proposed for synthesizing robust plans.[3] The first one translates this problem into a conformant probabilistic planning problem (Domshlak and Hoffmann, 2007). 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 (Domshlak and Hoffmann, 2007).

The second approach is a heuristic search method that works well in much larger problem instances. This method is fully investigated and tested under the STRIPS Execution semantics (the similar ideas can be adapted, with some additional challenges, for the Generous Execution semantics). It aims to directly take into account the robustness of plan prefixes during search. There is however 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. To overcome this issue, lower and upper bounds for the robustness measure are derived and incorporated into the extraction of robust relaxed plans. The lengh of these relaxed plans, as a common technique in classical planning, is used to guide the search towards a goal state. The experiments show that our planner, PISA (**P**lanning with **I**ncomplete **S**TRIPS **A**ctions), 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.

Having formalized the two classes of planning problems with incomplete user preferences and domain models in separate settings, I also discuss situations where the two types of incompleteness are simultaneously present. As a first step to tackle such complicated scenarios, I provide an outlook on how to adapt the quality measures for plan sets, proposed in the first part of the dissertation, in order to take the robustness of constituent plans into consideration.

[3]This work has been appeared in (Nguyen *et al.*, 2010, 2013; Nguyen and Kambhampati, 2014).

The dissertation is organized as follows: I describe the work on planning with incomplete user preferences in Chapter 2. I then discuss the work on planning with incomplete domain models in the next Chapter 3. In Chapter 4, I will summarize the work, and for future directions I will describe scenarios when the two types of incompleteness are present, and the extension of the quality measure for those situations.

Chapter 2

PLANNING WITH INCOMPLETE USER PREFERENCES

2.1 Introduction

In many real world planning scenarios, user preferences on plans are either unknown or at best partially specified (c.f. Kambhampati (2007)). In such cases, the planner's task changes from finding a *single* optimal plan to finding a *set* of representative solutions or options. The user must then be presented with this set in the hope that she will find at least one of the constituent plans desirable and in accordance with her preferences. Most work in automated planning ignores this reality, and assumes instead that user preferences (when expressed) will be provided in terms of a completely specified objective function.

This chapter presents a study on the problem of generating a *set of plans* using partial knowledge of the user preferences. This set is generated in the hope that the user will find at least one desirable according to her preferences. Specifically, the following two qualitatively distinct scenarios are considered.

- The planner is aware that the user has some preferences on the solution plan, but it is not provided with any knowledge on those preferences.
- The planner is provided with incomplete knowledge of the user preferences in the form of plan *attributes* (such as the duration or cost of a flight, or the importance of delivering all priority packages on time in a logistics problem). Each of these plan attributes has a different and unknown degree of importance, represented by *weights* or *trade-off* values. In general, users find it hard to indicate the exact value of a trade-off, but are more likely to indicate that one attribute is more (or less) important than

another. For instance, a business executive may consider the duration of a flight as a more important factor than its cost. Incompletely specified preferences such as these can be modeled with a probability distribution on weight values,[1] and can therefore be assumed as input to the planner (together with the attributes themselves).

In both of the cases above, the focus is on returning a set of plans. In principle, a larger plan set implies that the user has a better chance of finding the plan that she desires; however, there are two problems—one computational, and the other comprehensional. Plan synthesis, even for a single plan, is costly in terms of computational resources used; for a large set of plans, this cost only increases. The comprehensional problem, moreover, is that it is unclear if the user will be able to completely inspect a set of plans in order to find the plan she prefers. What is clearly needed, therefore, is the ability to generate a set of plans with the highest chance of including the user's preferred plan among all sets of *bounded* (small) number of plans. An immediate challenge in this direction is formalizing what it means for a *meaningful* set of plans—in other words, we want to define a *quality measure* for plan sets given an incomplete preference specification.

We propose different quality measures for the two scenarios listed above. In the extreme case where the user is unable to provide any knowledge of her preferences, we define a spectrum of distance measures between two plans based on their syntactic features in order to define the *diversity* measure of plan sets. These measures can be used regardless of the user preferences, and by maximizing the diversity of a plan set we increase the chance that the set is uniformly distributed in the unknown preference space. This makes it more likely that the set contains a plan that is close to the one desired by the user.

The quality measure can be refined further when some knowledge of the user preferences is provided. We assume that it is specified as a convex combination of the plan

[1] If there is no prior information about this probability distribution, one option is to initialize it with the uniform distribution and gradually improve it based on interaction with the user.

attributes mentioned above, and incomplete in the sense that a distribution of trade-off weights, not their exact values, is available. The complete set of best plans (plans with the best value function) can then be pictured as the lower convex-hull of the Pareto set on the attribute space. To measure the quality of any (bounded) set of plans on the complete optimal set, we adapt the idea of *Integrated Preference Function* (IPF) (Carlyle *et al.*, 2003), and in particular its special case, the *Integrated Convex Preference* (ICP). This measure was developed in the Operations Research (OR) community in the context of multi-criteria scheduling, and is able to associate a robust measure of representativeness with any set of solution schedules (Fowler *et al.*, 2005).

Armed with these quality measures, we can formulate the problem of planning with partial user preferences as finding a bounded set of the plans that has the best quality value. Our next contribution therefore is to investigate effective approaches for using quality measures to search for a high quality plan set efficiently. For the first scenario—when the preference specification is not provided—two representative planning approaches are considered. The first, GP-CSP (Do and Kambhampati, 2001), typifies the issues involved in generating diverse plans in bounded horizon compilation approaches; while the second, LPG (Gerevini *et al.*, 2003), typifies the issues involved in modifying the heuristic search planners. Our investigations with GP-CSP allow us to compare the relative difficulties of enforcing diversity with each of the three different distance measures defined in the forthcoming sections. With LPG, we find that the proposed quality measure makes it more effective in generating plan sets over large problem instances. For the second case—when part of the user preferences is provided—we also present a spectrum of approaches that can solve this problem efficiently. We implement these approaches on top of Metric-LPG (Gerevini *et al.*, 2008). Our empirical evaluation compares these approaches both among themselves as well as against the methods for generating diverse plans ignoring the partial preference information, and the results demonstrate the promise of our proposed solutions.

This chapter is organized as follows. The related work is first dicussed in the next Section 2.2. Section 2.3 describes fundamental concepts in preferences and formal notations. In Section 2.4, we formalize quality measures of plan set in the two scenarios discussed above. Sections 2.5 and 2.6 discuss our various heuristic approaches to generate plan sets, together with the experimental results. Section 2.7 gives the discussion including the limitations of our work. We finish our discussion on planning with incomplete user preferences with a conclusion in Section 2.8.

2.2 Related Work

There are currently very few research efforts in the planning literature that explicitly consider incompletely specified user preferences during planning. The most common approach for handling multiple objectives is to assume that a specific way of combining the objectives is available (Refanidis and Vlahavas, 2003; Do and Kambhampati, 2003), and then search for an optimal plan with respect to this function. In a sense, such work can be considered as assuming a complete specification of user preferences. Other relevant work includes (Bryce *et al.*, 2007), in which the authors devise a variant of the LAO* algorithm to search for a conditional plan with multiple execution options for each observation branch, such that each of these options is non-dominated with respect to objectives like probability and cost to reach the goal.

Our work can be seen as complementing the current research in planning with preferences. Under the umbrella of planning with preferences, most current work in planning focuses on synthesizing either a single plan under the assumption that the user has no preferences, or a single best solution assuming that a complete knowledge of the preferences is provided. We, on the other hand, address the problem of synthesizing a plan set when the knowledge of user preferences is either completely unknown,[2] or partially specified.

[2]Note that not knowing anything about the user's preferences is different from assuming that the user has

In the context of decision-theoretic planning, some work has considered Markov Decision Processes with *imprecise* reward functions, which are used to represent user preferences on the visited states during execution. These methods however assume that the true reward function is revealed only during the execution of policies, whereas in our setting the incomplete knowledge about user preferences is resolved after the synthesis of plans but before plan execution (with some required effort from the user). Many different notions of optimality for policies have been defined with respect to the incomplete reward function, and the aim is to search for an optimal policy. The *minimax regret* criterion (Regan and Boutilier, 2009, 2010; Xu and Mannor, 2009) has been defined for the quality of policies when the *true* reward function is deterministic but unknown in a given set of functions. This criterion seeks an optimal policy that minimizes the loss (in terms of the expected discounted reward) assuming the presence of an adversary who selects a reward function, among all possible ones, to maximize the loss should a policy be chosen. Another criterion, called *maximin*, maximizes the worst-case expected reward also assuming an adversary acting optimally against the agent (McMahan *et al.*, 2003).

Incomplete knowledge of user preferences can also be resolved with some effort from the user *during* plan generation. This idea unfortunately has not been considered in previous work on automated planning with preferences; there is however some work in two related areas, decision theory and preference elicitation. In (Chajewska *et al.*, 2000), the user is provided with a sequence of queries, one at a time, until an optimal strategy with respect to the refined preference model meets a stopping criterion, which is then output to the user. That work ignores the user's difficulty in answering questions that are posted, and instead emphasizes the construction of those which will give the best value of information at every step. This issue is overcome by (Boutilier, 2002) which takes into account the cost of answering future elicitation questions in order to reduce the user's effort. Boutilier

no preferences.

et al. (2010) consider the preference elicitation problem in which the incompleteness in user preferences is specified on both the set of features and the utility function. In systems implementing the example-critiquing interaction mechanism (e.g., Viappiani et al. (2006), Linden et al. (1997)), a user critiques examples or options presented by the system, and this information is then used to revise the preference model. The process continues until the user can pick a final choice from the k examples presented. There is one important difference between these methods and ours: the "outcomes" or "configurations" in these scenarios are considered given upfront (or can be obtained with low cost), whereas a feasible solution in many planning domains is computationally expensive to synthesize. As a result, an interactive method in which *a sequence of plans or sets of plans* needs to be generated for critiquing may not be suitable for our applications. Our approach, which presents a set of plans to the user to select, requires less effort from the user and at the same time avoids presenting a single optimal plan according to pessimistic or optimistic assumptions, such as those used in the minimax regret and maximin criteria.

The problem of reasoning with partially specified preferences has also long been studied in multi-attribute utility theory, though this work is also different from ours when ignoring the computation cost of "alternatives". Given prior preference statements on how the user compares two alternatives, Hazen (1986) considers additive and multiplicative utility functions with unknown scaling coefficients, which represents the user partial preferences, and proposes algorithms for the consistency problem (i.e., if there exists a complete utility function consistent with the prior preferences), the dominance problem (i.e., whether the prior information implies that one alternative is preferred to another), and the potential optimality problem (i.e., if there exists a complete utility function consistent with the prior preferences under which a particular alternative is preference optimal). Ha and Haddawy (1999) addressed the last two problems for multi-linear utility functions with unknown coefficients. These efforts are similar to ours in how the user preferences are partially rep-

resented. However, similar to the example-critiquing work mentioned above, they assume that the user is able to provide pairwise comparison between alternatives, which is then used to further constrain the set of complete utility functions representing user preferences.

Our approach to generating diverse plan sets to cope with planning scenarios without knowledge of user preferences is in the same spirit as (Tate *et al.*, 1998) and (Myers, 2006; Myers and Lee, 1999), though for different purposes. Myers, in particular, presents an approach to generate diverse plans in the context of an HTN planner by requiring the meta-theory of the domain to be available and by using bias on the meta-theoretic elements to control search (Myers and Lee, 1999). The metatheory of the domain is defined in terms of pre-defined attributes and their possible values covering roles, features and measures. Our work differs from this in two respects. First, we focus on domain-independent distance measures. Second, we consider the computation of diverse plans in the context of domain independent planners.

The problem of finding multiple but *similar* plans has been considered in the context of replanning (Fox *et al.*, 2006a). Our work focuses on the problem of finding *diverse* plans by a variety of measures when the user preferences exist but are either completely unknown or partially specified.

Outside the planning literature, our closest connection is first to the work by Gelain et al. (2010), who consider *soft* constraint satisfaction problems (CSPs) with incomplete preferences. These are problems where quantitative values of some constraints that represent their preferences are unspecified. Given such incomplete preferences, the authors are interested in finding a single solution that is "necessarily" optimal (possibly with some effort from the user), i.e. an assignment of variables that is optimal in all possible ways that the currently unspecified preferences can be revealed. In a sense, this notion of optimality is very similar to the maximin criterion when seeking a solution that is optimal even with the "worst" selection of the unspecified preferences. Hebrard et al. (2005) use a

model closer to ours that focuses on the problem of finding similar/dissimilar solutions for CSPs, assuming that a domain-specific distance measure between two solutions is already defined. It is instructive to note that unlike CSPs with finite variable domains, where the number of potential solutions is finite (albeit exponential), the number of distinct plans for a given problem can be infinite. Thus, effective approaches for generating a good quality set of plans are even more critical.

The challenges in finding a set of interrelated plans also bear some tangential similarities to the work in other research areas and applications. In information retrieval, Zhang et al. (2002) describe how to return relevant as well as novel (non-redundant) documents from a stream of documents; their approach is to first find relevant docs and then find non-redundant ones. In adaptive web services composition, the causal dependencies among some web services might change at execution time, and as a result the web service engine wants to have a set of diverse plans/compositions such that if there is a failure while executing one composition, an alternative may be used which is less likely to be failing simultaneously (Chafle *et al.*, 2006). However, if a user is helping in selecting the compositions, the planner could be first asked for a set of plans that may take into account the user's trust in some particular sources and when she selects one of them, it is next asked to find plans that are similar to the selected one. Another example of the use of diverse plans can be found in (Memon *et al.*, 2001) in which test cases for graphical user interfaces (GUIs) are generated as a set of distinct plans, each corresponding to a sequence of actions that a user could perform, given the user's unknown preferences on how to interact with the GUI to achieve her goals. The capability of synthesizing multiple plans would also have potential application in case-based planning (e.g., Serina (2010)) where it is important to have a plan set satisfying a case instance. These plans can be different in terms of criteria such as resources, makespan and cost that can only be specified in the retrieval phase.

The primary focus of our research are scenarios where the end user is interested in

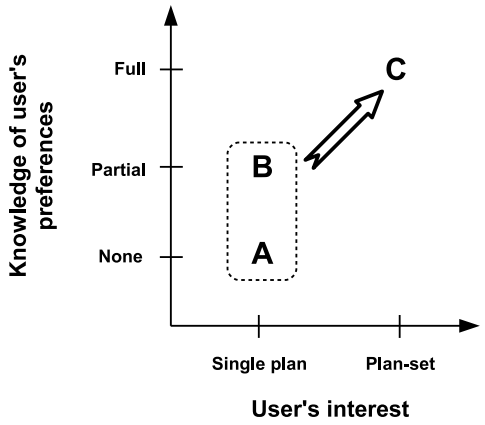


Figure 2.1: The planning problem with unknown (**A**) and partially known (**B**) user preferences can be reformulated as the problem of synthesizing plan-sets with complete preferences over plan-sets (**C**).

single plans, but her preferences on that single plan are either unknown or partially known to the planner. Our work shows that an effective technique for handling these scenarios is to generate a set of diverse plans and present them to the user (so she can select the single plan she is most interested in). While we came to sets of plans as an *intermediate step* for handling lack of preference knowledge about single plans, there are also applications where the end user is in fact interested in *sets of plans* (a.k.a "plan-sets"), and has preferences over these plan-sets. Techniques for handling this latter problem do overlap with the techniques we develop in this proposal, but it is important to remember their distinct motivations. Figure 2.1 makes these distinctions clear by considering two orthogonal dimensions. The X-axis is concerned with whether the end user is interested in single plans or plan-sets. The Y-axis is concerned with the degree of the knowledge of user preferences.

In this space, traditional planning with preferences corresponds to (`single-plan`, `full-knowledge`). The problems we are considering in this proposal are (`single-plan`, `no-knowledge`) and (`single-plan`, `partial-knowledge`), respectively. A contribution of our work is to show that these two latter problems can be reformulated as (`plan-set`,

`full-knowledge`), where the quality of plan-sets is evaluated by the internal diversity measures we will develop. There are also compelling motivations to study the (`plan-set`, `full-knowledge`) problem in its own right if the end user is explicitly interested in plan-sets. This is the case, for example, in applications such as intrusion detection (Boddy *et al.*, 2005), where the objective is to come up with a set of plans that can inhibit system breaches, or option generation in mission planning, where the commander wants a set of options not to immediately commit to one of them, but rather to study their trade-offs.

The techniques we develop in this proposal are related but not equivalent to the techniques and inputs for solving that plan-set generation problem. In particular, when the end users are interested in plan sets, they may have preferences on plan-sets, not on single plans.[3] This means that (i) we need a language support for expressing preferences on plan sets such as the work on DD-PREF language (desJardins and Wagstaff, 2005), and (ii) our planner has to take as input and support a wide variety of plan-set preferences (in contrast to our current system where the plan-set preference is decided internally—in terms of distance measures for unknown (single plan) preference case, and in terms of IPF measure for partially known preference cases).

2.3 Background and Notation

Given a planning problem with the set of solution plans $\mathcal{S}$, a user preference *model* is a transitive, reflexive relation in $\mathcal{S} \times \mathcal{S}$, which defines an ordering between two plans p and p' in $\mathcal{S}$. Intuitively, $p \preceq p'$ means that the user prefers p at least as much as p'. Note that this ordering can be either partial (i.e. it is possible that neither $p \preceq p'$ nor $p' \preceq p$ holds—in other words, they are incomparable), or total (i.e. either $p \preceq p'$ or $p' \preceq p$ holds). A plan p is considered (strictly) more preferred than a plan p', denoted by $p \prec p'$, if $p \preceq p'$,

[3] This is akin to a college having explicit preferences on its freshman classes—such as student body diversity—over and above their preferences on individual students.

$p' \not\preceq p$, and they are equally preferred if $p \preceq p'$ and $p' \preceq p$. A plan p is an optimal (i.e., most preferred) plan if $p \preceq p'$ for any other plan p'. A plan set $\mathcal{P} \subseteq \mathcal{S}$ is considered more preferred than $\mathcal{P}' \subseteq \mathcal{S}$, denoted by $\mathcal{P} \ll \mathcal{P}'$, if $p \prec p'$ for any $p \in \mathcal{P}$ and $p' \in \mathcal{P}'$, and they are incomparable if there exists $p \in \mathcal{P}$ and $p' \in \mathcal{P}'$ such that p and p' are incomparable.

The ordering $\preceq$ implies a partition of $\mathcal{S}$ into disjoint plan sets (or *classes*) $\mathcal{S}_0$, $\mathcal{S}_1$, ... ($\mathcal{S}_0 \cup \mathcal{S}_1 \cup ... = \mathcal{S}$, $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$) such that plans in the same set are equally preferred, and for any set $\mathcal{S}_i$, $\mathcal{S}_j$, either $\mathcal{S}_i \ll \mathcal{S}_j$, $\mathcal{S}_j \ll \mathcal{S}_i$, or they are incomparable. The partial ordering between these sets can be represented as a Hasse diagram (Birkhoff, 1948) where the sets are vertices, and there is an (upward) edge from $\mathcal{S}_j$ to $\mathcal{S}_i$ if $\mathcal{S}_i \ll \mathcal{S}_j$ and there is not any $\mathcal{S}_k$ in the partition such that $\mathcal{S}_i \ll \mathcal{S}_k \ll \mathcal{S}_j$. We denote $l(\mathcal{S}_i)$ as the "layer" of the set $\mathcal{S}_i$ in the diagram, assuming that the most preferred sets are placed at the layer 0, and $l(\mathcal{S}_j) = l(\mathcal{S}_i) + 1$ if there is an edge from $\mathcal{S}_j$ to $\mathcal{S}_i$. A plan in a set at a layer of smaller value, in general, is either more preferred than or incomparable with ones at layers of higher values.[4] Figure 2.2 shows two examples of Hasse diagrams representing a total and partial preference ordering between plans. We will use this representation of plan sets in Section 2.4 to justify the design of our quality measures for plan sets when no knowledge of user preferences is available.

The preference model of a user can be explicitly specified by iterating the set of plans and providing the ordering between any two of them, and in this case answering queries such as comparing two plans, finding a most preferred (optimal) plan becomes an easy task. This is, however, practically infeasible since synthesizing a plan in itself is hard, and the solution space of a planning problem can be infinite. Many preference *languages*, therefore, have been proposed to represent the relation $\preceq$ in a more compact way, and serve as starting points for *algorithms* to answer queries. Most preference languages fall into the

[4]If $\preceq$ is a total ordering, then plans at a layer of smaller value are strictly more preferred than ones at a layer of higher value.

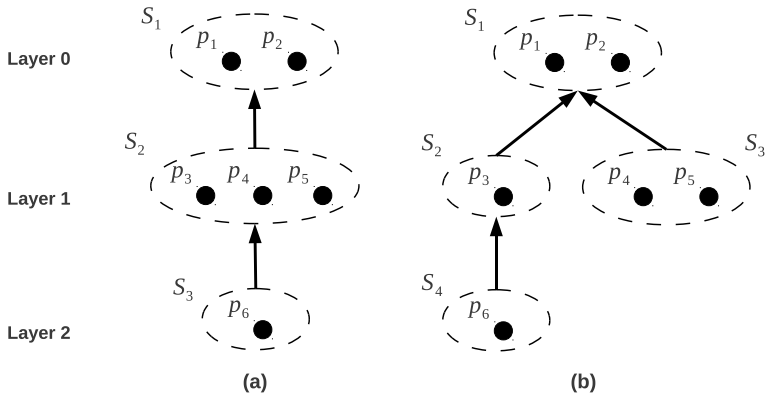


Figure 2.2: The Hasse diagrams and layers of plan sets implied by two preference models. In (a), $\mathcal{S}_1 \ll \mathcal{S}_2 \ll \mathcal{S}_3$, and any two plans are comparable. In (b), on the other hand, $\mathcal{S}_1 \ll \mathcal{S}_2 \ll \mathcal{S}_4$, $\mathcal{S}_1 \ll \mathcal{S}_3$, and each plan in $\mathcal{S}_3$ is incomparable with plans in $\mathcal{S}_2$ and $\mathcal{S}_4$.

following two categories:

- Quantitative languages define a *value function* $V : \mathcal{S} \to R$ which assigns a real number to each plan, with a precise interpretation that $p \preceq p' \iff V(p) \leq V(p')$. Although this function is defined differently in many languages, at a high level it combines the user preferences on various aspects of plan that can be measured quantitatively. For instance, in the context of decision-theoretic planning (Boutilier *et al.*, 1999), the value function of a policy is defined as the expected rewards of states that are visited when the policy executes. In partial satisfaction (over-subscription) planning (PSP) (Smith, 2004; Van Den Briel *et al.*, 2004), the quality of plans is defined as its total rewards of soft goals achieved minus its total action costs. In PDDL2.1 (Fox and Long, 2003), the value function is an arithmetic function of numerical fluents such as plan makespans, fuel used etc., and in PDDL3 (Gerevini *et al.*, 2009) it is enhanced with individual preference specifications over state trajectory constraints, defined as formulae with modal operators having their semantics consistent with that used for modal operators in linear temporal logic (Pnueli, 1977) and other modal

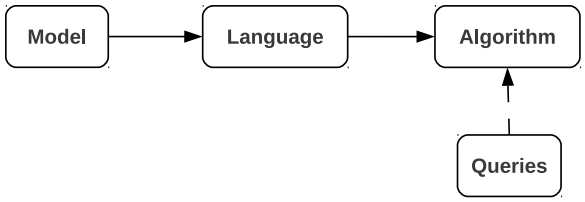


Figure 2.3: The metamodel (Brafman and Domshlak, 2009). The user preference model is compactly represented by a preference language, on which algorithms perform tasks of answering queries.

temporal logics.

- Qualitative languages provide qualitative statements that are more intuitive for lay users to specify. A commonly used language of this type is *CP-networks* proposed in (Boutilier *et al.*, 2004), where the user can specify her preference statements on values of plan attributes, possibly given specification of others (for instance, "Among tickets with the same prices, I prefer airline A to airline B."). Another example is *LPP* (Bienvenu *et al.*, 2006) in which the statements can be specified using LTL formulae, and possibly being aggregated in different ways.

Figure 2.3 shows the conceptual relation of preference models, languages and algorithms. We refer the reader to the work by Brafman and Domshlak (2009) for a more detailed discussion on this metamodel, and by Baier and McIlraith (2009) for an overview of different preference languages used in planning with preferences.

From the modeling point of view, in order to design a suitable language capturing the user preference model, the modeler should be provided with some knowledge of the user's interest that affects the way she evaluates plans (for instance, flight duration and ticket cost in a travel planning scenario). Such knowledge in many cases, however, cannot be completely specified. Our purpose therefore is to present a bounded set of plans to the user in the hope that it will increase the chance that she can find a desired plan. In the next

section, we formalize the quality measures for plan sets in two situations where either no knowledge of the user preferences or only part of them is given.

2.4 Quality Measures for Plan Sets

2.4.1 Syntactic Distance Measures for Unknown Preference Cases

We first consider the situation in which the user has some preferences for solution plans, but the planner is not provided with any knowledge of such preferences. It is therefore impossible for the planner to assume any particular form of preference language representing the hidden preference model. There are two issues that need to be considered in formalizing a quality measure for plan sets:

- What are the elements of plans that can be involved in a quality measure?
- How should a quality measure be defined using those elements?

For the first question, we observe that even though users are normally interested in some *high level* features of plans that are relevant to them, many of those features can be considered as "functions" of *base level* elements of plans. For instance, the set of actions in the plan determines the makespan of a (sequential) plan, and the sequence of states when the plan executes gives the total reward of goals achieved. We consider the following three types of base level features of plans which could be used in defining quality measure, independently of the domain semantics:

- *Actions that are present in plans*, which define various high level features of the plans such as its makespan, execution cost, etc. that are of interest to the user whose preference model could be represented with preference languages such as in PSP and PDDL2.1.

- *Sequence of states that the agent goes through, which captures the behaviors resulting from the execution of plans.* In many preference languages defined using high level features of plans such as the reward of goals collected (e.g., PSP), of the whole state (e.g., MDP), or the temporal relation between propositions occurring in states (e.g. PDDL3, $\mathcal{PP}$ (Son and Pontelli, 2006) and *LPP* (Fritz and McIlraith, 2006)), the sequence of states can affect the quality of plan evaluated by the user.

- *The causal links representing how actions contribute to the goals being achieved, which measures the causal structures of plans.*[5] These plan elements can affect the quality of plans with respect to the languages mentioned above, as the causal links capture both the actions appearing in a plan and the temporal relation between actions and variables.

A similar conceptual separation of features has also been considered recently in the context of case-based planning by Serina (2010), in which planning problems were assumed to be well classified, in terms of costs to adapt plans of one problem to solve another, in some *unknown* high level feature space. The similarity between problems in the space was implicitly defined using kernel functions of their domain-independent graph representations. In our situation, we aim to approximate quality of plan sets on the space of features that the user is interested by using distance between plans with respect to the base level features mentioned above.

Table 2.1 gives the pros and cons of using the different base level elements of plan. We note that if actions in the plans are used in defining quality measure of plan sets, no additional problem or domain theory information is needed. If plan behaviors are used as base level elements, the representation of the plans that bring about state transition becomes irrelevant since only the actual states that an execution of the plan will go through

[5] A causal link $a_1 \rightarrow p - a_2$ records that a proposition p is produced by a_1 and consumed by a_2.

Basis	**Pros**	**Cons**
Actions	Does not require problem information	No problem information is used
States	Not dependent on any specific plan representation	Needs an execution simulator to identify states
Causal links	Considers causal proximity of state transitions (action) rather than positional (physical) proximity	Requires domain theory

Table 2.1: The pros and cons of different base level elements of plan.

are considered. Hence, we can now compare plans of different representations, e.g., four plans where the first is a deterministic plan, the second is a contingent plan, the third is a hierarchical plan and the fourth is a policy encoding probabilistic behavior. If causal links are used, then the causal proximity among actions is now considered rather than just physical proximity in the plan.

Given those base level elements, the next question is how to define a quality measure of plan sets using them. Recall that without any knowledge of the user preferences, there is no way for the planner to assume any particular preference language, thus the motivation behind the choice of quality measure should come from the hidden user preference model. Given a Hasse diagram induced from the user preference model, a k-plan set that will be presented to the user can be considered to be randomly selected from the diagram. The probability of having one plan in the set classified in a class at the optimal layer of the Hasse diagram would increase when the individual plans are more likely to be at different layers, and this chance in turn will increase if they are less likely to be equally preferred

by the user.[6] On the other hand, the effect of base level elements of a plan on high level features relevant to the user suggests that *plans similar with respect to base level features are more likely to be close to each other on the high level feature space determining the user preference model.*

In order to define a quality measure using base level features of plans, we proceed with the following assumption: *plans that are different from each other with respect to the base level features are less likely to be equally preferred by the user, in other words they are more likely to be at different layers of the Hasse diagram.* With the purpose of increasing the chance of having a plan that the user prefers, we propose the quality measure of plan sets as its *diversity* measure, defined using the distance between two plans in the set with respect to a base level element. More formally, the quality measure $\zeta : 2^{\mathcal{S}} \rightarrow \mathbb{R}$ of a plan set $\mathcal{P}$ can be defined as either the minimal, maximal, or average distance between plans:

- minimal distance:

$$\zeta_{min}(\mathcal{P}) = \min_{p,p' \in \mathcal{P}} \delta(p, p'), \tag{2.1}$$

- maximal distance:

$$\zeta_{max}(\mathcal{P}) = \max_{p,p' \in \mathcal{P}} \delta(p, p'), \tag{2.2}$$

- average distance:

$$\zeta_{avg}(\mathcal{P}) = \binom{|\mathcal{P}|}{2}^{-1} \times \sum_{p_i,p_j \in \mathcal{P}, i<j} \delta(p_i, p_j), \tag{2.3}$$

[6] To see this, consider a diagram with $\mathcal{S}_1 = \{p_1, p_2\}$ at layer 0, $\mathcal{S}_2 = \{p_3\}$ and $\mathcal{S}_3 = \{p_4\}$ at layer 1, and $\mathcal{S}_4 = \{p_5\}$ at layer 2. Assuming that we randomly select a set of 2 plans. If those plans are known to be at the same layer, then the chance of having one plan at layer 0 is $\frac{1}{2}$. However, if they are forced to be at different layers, then the probability will be $\frac{3}{4}$.

where $\delta : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ is the distance measures between two plans.

Distance measures between plans There are various choices on how to define the distance measure $\delta(p, p')$ between two plans using plan actions, sequence of states or causal links, and each way can have different impact on the diversity of plan set on the Hasse diagram. In the following, we propose distance measures in which a plan is considered as (i) a set of actions and causal links, or (ii) sequence of states the agent goes through, which could be used independently of plan representation (e.g. total order, partial order plans).

- *Plan as a set of actions or causal links*: given a plan p, let $A(p)$ and $C(p)$ be the set of actions or causal links of p. The distance between two plans p and p' can be defined as the ratio of the number of actions (causal links) that do not appear in both plans to the total number of actions (causal links) appearing in one of them:

$$\delta_a(p, p') = 1 - \frac{|A(p) \cap A(p')|}{|A(p) \cup A(p')|}, \tag{2.4}$$

$$\delta_{cl}(p, p') = 1 - \frac{|C(p) \cap C(p')|}{|C(p) \cup C(p')|}. \tag{2.5}$$

- *Plan as a sequence of states*: given two sequences of states $(s_0, s_1, ..., s_k)$ and $(s'_0, s'_1, ..., s'_{k'})$ resulting from executing two plans p and p', and assume that $k' \leq k$. Since the two sequences of states may have different lengths, there are various options in defining distance measure between p and p', and we consider here two options. In the first one, it can be defined as the average of the distances between state pairs (s_i, s'_i) $(0 \leq i \leq k')$, and each state $s_{k'+1}$,... s_k is considered to contribute maximally (i.e., one unit) into the difference between two plans:

$$\delta_s(p, p') = \frac{1}{k} \times [\sum_{i=1}^{k'} \Delta(s_i, s'_i) + k - k']. \tag{2.6}$$

On the other hand, we can assume that the agent continues to stay at the goal state $s'_{k'}$ in the next $(k - k')$ time steps after executing p', and the measure can be defined as follows:

$$\delta_s(p, p') = \frac{1}{k} \times [\sum_{i=1}^{k'} \Delta(s_i, s'_i) + \sum_{i=k'+1}^{k} \Delta(s_i, s'_{k'})]. \tag{2.7}$$

The distance measure $\Delta(s, s')$ between two states s, s' used in those two measures is defined as:

$$\Delta(s, s') = 1 - \frac{|s \cap s'|}{|s \cup s'|}. \tag{2.8}$$

These distance metrics would consider long plans to be distant from short plans. In the absence of information about user preferences, we cannot rule out the possibility that the unknown preference might actually favor longer plans (e.g., it is possible that a longer plan has cheaper actions, making it attractive for the user). In the implementation of a system for computing diverse plans, while these distance measures affect which part of the (partial plan) search space a planner tends to focus on, in general the length of resulting plans especially depends on whether the search strategy of the planner attempts to minimize it. In our experiments, we will use two types of planners employing exhaustive search and local search, respectively. For the second, which does not attempt to minimize plan length, we will introduce additional constraints into the search mechanism that, by balancing the differences in the generated diverse plans, also attempts to control the relative size of resulting plans.

Example: Figure 2.4 shows three plans p_1, p_2 and p_3 for a planning problem where the initial state is $\{r_1\}$ and the goal propositions are $\{r_3, r_4\}$. The specification of actions are shown in the table. The action sets of the first two plans ($\{a_1, a_2, a_3\}$ and $\{a_1, a_2, a_4\}$)

Initial state: $\{r_1\}$

Goals: $\{r_3, r_4\}$

Action	Pre	Add	Delete
a_1	r_1	r_2	r_1
a_2	r_2	r_3	
a_3	r_3	r_4	
a_4	r_1	r_4	
a_5	r_1	r_2	
a_6	r_2	r_3, r_4	r_1

State sequence:

p_1:$(\{r_1\},\{r_2\},\{r_2,r_3\},\{r_2,r_3,r_4\})$

p_2:$(\{r_1\},\{r_2,r_4\},\{r_2,r_3,r_4\})$

p_3:$(\{r_1\},\{r_1,r_2\},\{r_3,r_4\})$

Causal links:

p_1:$\{a_I \to r_1 - a_1, a_1 \to r_2 - a_2, a_2 \to r_3 - a_3, a_2 \to r_3 - a_G, a_3 \to r_4 - a_G\}$

p_2:$\{a_I \to r_1 - a_1, a_I \to r_1 - a_4, a_1 \to r_2 - a_2, a_2 \to r_3 - a_G, a_4 \to r_4 - a_G\}$

p_3:$\{a_I \to r_1 - a_5, a_5 \to r_2 - a_6, a_6 \to r_3 - a_G, a_6 \to r_4 - a_G\}$

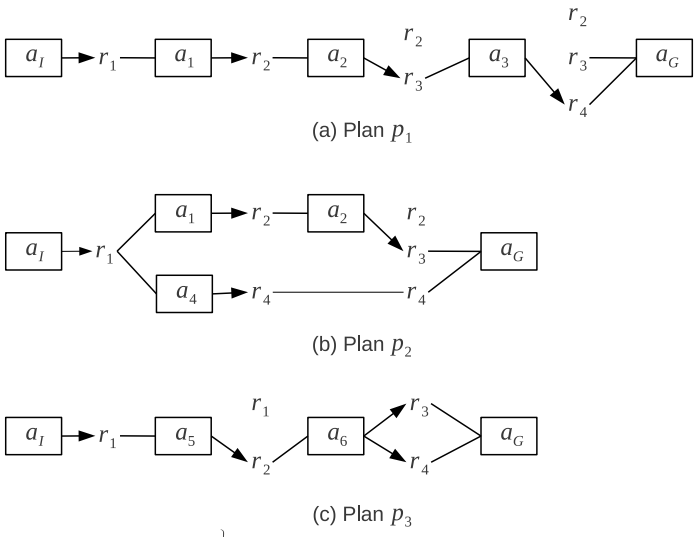


Figure 2.4: Example illustrating plans with base-level elements. a_I and a_G denote dummy actions producing the initial state and consuming the goal propositions, respectively (see text for more details).

are quite similar ($\delta_a(p_1, p_2) = 0.5$), but the causal links which involve a_3 ($a_2 \to r_3 - a_3$, $a_3 \to r_4 - a_G$) and a_4 ($a_I \to r_1 - a_4$, $a_4 \to r_4 - a_G$) make their difference more significant with respect to causal-link based distance ($\delta_{cl}(p_1, p_2) = \frac{4}{7}$). Two other plans p_1 and p_3, on the other hand, are very different in terms of action sets (and therefore the sets of causal links): $\delta_a(p_1, p_3) = 1$, but they are closer in term of state-based distance ($\frac{13}{18}$ as defined in Equation 2.6, and 0.5 if defined in Equation 2.7).

2.4.2 Integrated Preference Function (IPF) for Partial Preference Cases

We now discuss a quality measure for plan sets in the case when the user preference is partially expressed. In particular, we consider scenarios in which the preference model can be represented by some quantitative language with an incompletely specified value function of high level features. As an example, the quality of plans in languages such as PDDL2.1

(Fox and Long, 2003) and PDDL3 (Gerevini *et al.*, 2009) are represented by a metric function combining metric fluents and preference statements on state trajectory with parameters representing their relative importance. While providing a convenient way to represent preference models, such parameterized value functions present an issue of obtaining reasonable values for the relative importance of the features. A common approach to model this type of incomplete knowledge is to consider those parameters as a vector of random variables, whose values are assumed to be drawn from a distribution. This is the representation that we will follow.

To measure the quality of plan sets, we propose the usage of *Integrated Preference Function* (IPF) (Carlyle *et al.*, 2003), which has been used to measure the quality of a solution set in a wide range of multi-objective optimization problems. The IPF measure assumes that the user preference model can be represented by two factors: (1) a probability distribution $h(\alpha)$ of parameter vector α, whose domain is denoted by Λ, such that $\int_{\alpha\in\Lambda} h(\alpha)\, d\alpha = 1$ (in the absence of any special information about the distribution, $h(\alpha)$ can be assumed to be uniform), and (2) a value function $V(p,\alpha) : \mathcal{S} \times \Lambda \rightarrow \mathbb{R}$ combines different objective functions into a single real-valued quality measure for plan p. We assume that such objective functions represent aspects of plans that have to be minimized, such as makespan and execution cost. This incomplete specification of the value function represents a set of candidate preference models, for each of which the user will select a different plan, the one with the best value, from a given plan set $\mathcal{P} \subseteq \mathcal{S}$. The IPF value of solution set $\mathcal{P}$ is defined as:

$$IPF(\mathcal{P}) = \int_{\alpha\in\Lambda} h(\alpha)V(p_\alpha,\alpha)\, d\alpha, \tag{2.9}$$

with $p_\alpha = \underset{p\in\mathcal{P}}{\operatorname{argmin}}\, V(p,\alpha)$, i.e., the best solution in $\mathcal{P}$ according to $V(p,\alpha)$ for each given α value. Let p_α^{-1} be a range of α values for which p is an optimal solution according to $V(p,\alpha)$, i.e., $V(p,\alpha) \leq V(p',\alpha)$ for all $\alpha \in p_\alpha^{-1}$, $p' \in \mathcal{P} \setminus \{p\}$.

As p_α is piecewise constant, the $IPF(\mathcal{P})$ value can be computed as:

$$IPF(\mathcal{P}) = \sum_{p \in \mathcal{P}} \left[\int_{\alpha \in p_\alpha^{-1}} h(\alpha) V(p, \alpha)\, d\alpha \right]. \tag{2.10}$$

Let $\mathcal{P}^* = \{p \in \mathcal{P} : \ p_\alpha^{-1} \neq \emptyset\}$; then we have:

$$IPF(\mathcal{P}) = IPF(\mathcal{P}^*) = \sum_{p \in \mathcal{P}^*} \left[\int_{\alpha \in p_\alpha^{-1}} h(\alpha) V(p, \alpha)\, d\alpha \right]. \tag{2.11}$$

Since $\mathcal{P}^*$ is the set of plans that are optimal for some specific parameter vector, $IPF(\mathcal{P})$ now can be interpreted as the expected value that the user can get by selecting the best plan in $\mathcal{P}$. Therefore, the set $\mathcal{P}^*$ of solutions (known as *lower convex hull* of $\mathcal{P}$) with the minimal IPF value is most likely to contain the desired solutions that the user wants, and in essence it is a good representative of the plan set $\mathcal{P}$.

The requirement for $IPF(\mathcal{P})$ to exist is that the function $h(\alpha)V(p, \alpha)$ needs to be integrable over the p_α^{-1} domains.[7] The complication in computing the $IPF(\mathcal{P})$ value is in deriving a partition of Λ, the domain of α, into the ranges p_α^{-1} for $p \in \mathcal{P}^*$, and the computation of integration over those ranges of the parameter vector. As we will describe, the computational effort to obtain $IPF(\mathcal{P})$ is negligible in our work with two objectives. Although it is beyond the scope of our work, we refer the readers to (Kim *et al.*, 2006) for the calculation of the measure when the value function is a convex combination of high number of objectives, and to (Bozkurt *et al.*, 2010) for the weighted Tchebycheff value function with two and three criteria.

In this work, in order to make our discussion on generating plan sets concrete, we will concentrate on metric temporal planning where each action $a \in A$ has a duration d_a and an

[7] Although a value function can take any form satisfying axioms about preferences, the user preferences in many real-world scenarios can be represented or approximated with an *additive value function* (Russell and Norvig, 2010), including the setting in our application, which is integrable over the parameter domain. Since $h(\alpha)$ is integrable, so is the product $h(\alpha)V(p, \alpha)$ in those situations.

execution cost c_a. The planner needs to find a plan $p = \langle a_1, \ldots, a_n \rangle$, which is a sequence of actions that is executable and achieves all goals. The two most common plan quality measures are: *makespan*, the total execution time of p, *plan cost*, the total execution cost of all actions in p. Both of them are high level features that can be affected by the actions in the plan. In most real-world applications, these two criteria compete with each other in that shorter plans usually have higher cost and vice versa. We use the following assumptions:

- The desired objective function involves minimizing both components: $time(p)$ measures the makespan of the plan p and $cost(p)$ measures its execution cost.
- The quality of a plan p is a convex combination: $V(p, w) = w \times time(p) + (1 - w) \times cost(p)$, where weight $w \in [0, 1]$ represents the trade-off between the two competing objective functions.
- The belief distribution of w over the range $[0, 1]$ is known. If the user does not provide any information or we have not learnt anything about the preference on the trade-off between *time* and *cost* of the plan, then the planner can assume a uniform distribution (and improve it later using techniques such as preference elicitation).

Given that the exact value of w is unknown, our purpose is to find a bounded representative set of non-dominated[8] plans minimizing the expected value of $V(p, w)$ with regard to the given distribution of w over $[0, 1]$.

IPF for Metric Temporal Planning: The user preference model in our target domain of temporal planning is represented by a convex combination of the *time* and *cost* quality measures, and the IPF measure now is called *Integrated Convex Preference* (ICP). Given a set of plans $\mathcal{P}^*$, let $t_p = time(p)$ and $c_p = cost(p)$ be the makespan and total execution

[8] A plan p_1 is dominated by p_2 if $time(p_1) \geq time(p_2)$ and $cost(p_1) \geq cost(p_2)$ and at least one of the inequalities is strict.

cost of plan $p \in \mathcal{P}^*$, the ICP value of $\mathcal{P}^*$ with regard to the objective function $V(p, w) = w \times t_p + (1 - w) \times c_p$ and the parameter vector $\alpha = (w, 1 - w)$ ($w \in [0, 1]$) is defined as:

$$ICP(\mathcal{P}^*) = \sum_{i=1}^{k} \int_{w_{i-1}}^{w_i} h(w)(w \times t_{p_i} + (1 - w) \times c_{p_i})dw, \tag{2.12}$$

where $w_0 = 0$, $w_k = 1$ and $V(p_i, w) \leq V(p, w)$ for all $p \in \mathcal{P}^* \setminus \{p_i\}$ and every $w \in [w_{i-1}, w_i]$. In other words, we divide $[0, 1]$ into k non-overlapping regions such that in each region (w_{i-1}, w_i) there is an optimal solution $p_i \in \mathcal{P}^*$ according to the value function.

We select the IPF/ICP measure to evaluate our solution set for the following reasons:

- From the perspective of *decision theory*, presenting a plan set $\mathcal{P} \subseteq \mathcal{S}$ to the user, among all possible subsets of $\mathcal{S}$, can be considered as an "action" with possible "outcomes" $p \in \mathcal{P}$ that can occur (i.e., being selected by the user) with probability $\int_{\alpha \in p_\alpha^{-1}} h(\alpha)\, d\alpha$. Since the $IPF(\mathcal{P})$ measures the expected utility of $\mathcal{P}$, presenting a set of plans with an optimal IPF value is a rational action consistent with the current knowledge of the user preferences.

- If $\mathcal{P}_1$ dominates $\mathcal{P}_2$ in the *set Pareto dominance* sense, then $IPF(\mathcal{P}_1) \leq IPF(\mathcal{P}_2)$ for any type of weight density function $h(\alpha)$ (Carlyle *et al.*, 2003), and this property also holds with any scaling of the objective values for ICP measure (Fowler *et al.*, 2005). Intuitively, this means that if we "merge" those two plan sets, all nondominated plans "extracted" from the resulting set are those in $\mathcal{P}_1$.

- The value of $IPF(\mathcal{P})$ is monotonically nonincreasing over increasing sequences of solution sets, and the set of plans optimal according to the utility function $V(p, \alpha)$, i.e., the efficient frontier, has the minimal IPF value (Carlyle *et al.*, 2003). Thus, the measure can be used as an indicator for the quality of a plan set during the search towards the efficient frontier.

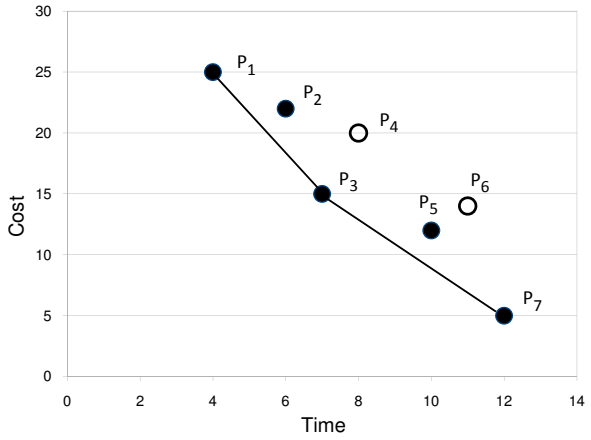


Figure 2.5: Solid dots represent plans in the Pareto set $(p_1, p_2, p_3, p_5, p_7)$. Connected dots represent plans in the lower convex hull (p_1, p_3, p_7) giving optimal ICP value for any distribution on trade-off between *cost* and *time*.

Empirically, extensive results on scheduling problems in (Fowler *et al.*, 2005) have shown that ICP measure *"evaluates the solution quality of approximation robustly (i.e., similar to visual comparison results) while other alternative measures can misjudge the solution quality"*.

Example: Figure 2.5 shows our running example in which there are a total of 7 plans with their $time(p)$ and $cost(p)$ values as follows: $p_1 = \{4, 25\}$, $p_2 = \{6, 22\}$, $p_3 = \{7, 15\}$, $p_4 = \{8, 20\}$, $p_5 = \{10, 12\}$, $p_6 = \{11, 14\}$, and $p_7 = \{12, 5\}$. Among these 7 plans, 5 of them belong to a Pareto optimal set of non-dominated plans: $\mathcal{P}_p = \{p_1, p_2, p_3, p_5, p_7\}$. The other two plans are dominated by some plans in $\mathcal{P}_p$: p_4 is dominated by p_3 and p_6 is dominated by p_5. Plans in $\mathcal{P}_p$ are depicted in solid dots, and the set of plans $\mathcal{P}^* = \{p_1, p_3, p_7\}$ that are optimal for some specific value of w is highlighted by connected dots. In particular, p_7 is optimal when $w \in [w_0 = 0, w_1 = \frac{2}{3}]$ where $w_1 = \frac{2}{3}$ can be derived from the satisfaction of the constraints $V(p_7, w) \leq V(p, w)$, $p \in \{p_1, p_3\}$. Similarly, p_3 and p_1 are respectively optimal for $w \in [w_1 = \frac{2}{3}, w_2 = \frac{10}{13}]$ and $w \in [w_2 = \frac{10}{13}, w_3 = 1]$. Assuming that $h(w)$ is a uniform distribution, the value of $ICP(\mathcal{P})$ can therefore be computed as

follows:

$$\begin{aligned} ICP(\mathcal{P}^*) &= \int_0^{\frac{2}{3}} h(w)V(p_7,w)dw + \int_{\frac{2}{3}}^{\frac{10}{13}} h(w)V(p_3,w)dw + \int_{\frac{10}{13}}^{1} h(w)V(p_1,w)dw \\ &= \int_0^{\frac{2}{3}} [12w + 5(1-w)]dw + \int_{\frac{2}{3}}^{\frac{10}{13}} [7w + 15(1-w)]dw + \\ &\quad + \int_{\frac{10}{13}}^{1} [4w + 25(1-w)]dw \\ &\approx 7.32. \end{aligned}$$

In the next two Sections 2.5 and 2.6, we investigate the problem of generating high quality plan sets for two cases mentioned: when no knowledge about the user preferences is given, and when part of it is given as input to the planner.

2.5 Generating Diverse Plan Sets in the Absence of Preference Knowledge

In this section, we describe approaches to searching for a set of diverse plans with respect to a measure defined with base level elements of plans as discussed in the previous section. In particular, we consider the quality measure of plan set as the minimal pair-wise distance between any two plans, and generate a set of plans containing k plans with the quality of at least a predefined threshold d. As discussed earlier, by diversifying the set of plans on the space of base level features, it is likely that plans in the set would cover a wide range of space of unknown high level features, increasing the possibility that the user can select a plan close to the one that she prefers. The problem is formally defined as follows:

$$d\text{DISTANT}k\text{SET}: \text{Find } \mathcal{P} \text{ with } \mathcal{P} \subseteq \mathcal{S}, \mid \mathcal{P} \mid = k \text{ and } \zeta(\mathcal{P}) = \min_{p,q \in \mathcal{P}} \delta(p,q) \geq d,$$

where any distance measure between two plans formalized in Section 2.4.1 can be used to implement $\delta(p, p')$.

We now consider two representative state-of-the-art planning approaches in generating diverse plan sets. The first one is GP-CSP (Do and Kambhampati, 2001) representing

constraint-based planning approaches, and the second one is LPG (Gerevini *et al.*, 2003) that uses an efficient local-search based approach. We use GP-CSP to compare the relation between different distance measures in diversifying plan sets. On the other hand, with LPG we stick to the action-based distance measure, which is shown experimentally to be the most difficult measure to enforce diversity (see below), and investigate the scalability of heuristic approaches in generating diverse plans.

2.5.1 Finding Diverse Plan Set with GP-CSP

The GP-CSP planner converts the planning graph of Graphplan (Blum and Furst, 1997) into a CSP encoding, and solves it using a standard CSP solver. A planning graph is a data structure consisting of alternating levels of proposition set and action set. The set of propositions present in the initial state is the proposition set at the zero-th level of the graph. Given a k-level planning graph, all actions whose preconditions are present in the proposition set of level k are introduced into the next level $k + 1$. In addition, one "noop" action is also added for each proposition at level k, which are both the precondition and effect of the action. The set of propositions at level $k + 1$ is then constructed by taking the union of additive effects of all actions at the same level. This expansion process also computes and propagates a set of "mutex" (i.e., mutually exclusive) constraints between pairs of propositions and actions at each level. At the first level, the computation starts by marking as mutex the actions that are statically interfering with each other (i.e., their preconditions and effects are inconsistent). The mutex constraints are then propagated as follows: (i) at level k, two propositions are mutually exclusive if any action at level k achieving one of them is mutually exclusive with all actions at the same level supporting the other one; (ii) two actions at level $k + 1$ are mutex if they are statically interfering or if one of the precondition of the first action is mutually exclusive with one of the precondition of the second action.

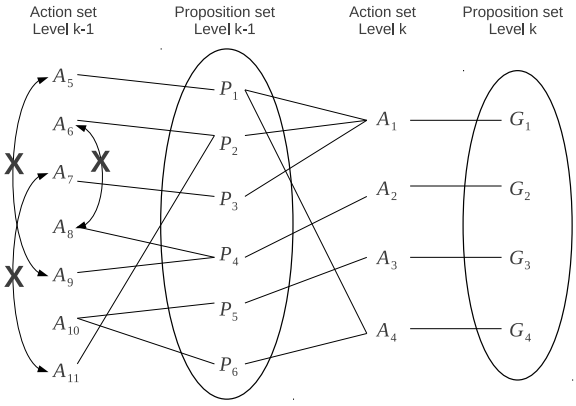


Figure 2.6: An example of (a portion of) a planning graph. At each level, propositions presenting in a previous one and noop actions are omitted, and at level k only the actions used to support the goals are shown for simplification.

The planning graph construction stops at a level T at which one of the following conditions is satisfied: (i) all goal propositions are present in the proposition set of level T without any mutex constraints between them, or (ii) two consecutive levels of the graph have the same sets of actions, propositions and mutex constraints. In the first case, the Graphplan algorithm searches this graph backward (i.e., from level T) for a valid plan, and continuing the planning graph expansion before a new search if no solution exists. In the second condition, the problem is provably unsolvable. Figure 2.6, which is taken from (Do and Kambhampati, 2001), shows an example of two levels of a planning graph. The top-level goals are $G_1, ..., G_4$ supported by actions $A_1, ..., A_4$ at the same level k. Each of these actions has preconditions in the set $\{P_1, ..., P_6\}$ appearing at level $k-1$, which are in turn supported by actions $A_5, ..., A_{11}$ at that level. The action pairs $\{A_5, A_9\}$, $\{A_7, A_{11}\}$ and $\{A_6, A_8\}$ are mutually exclusive, however these mutex relations are not enough to make any pair of propositions at level $k-1$ mutually exclusive.

The GP-CSP planner replaces the search algorithm in Graphplan by first converting the planning graph data structure into a constraint satisfaction problem, and then invoking a

solver to find an assignment of the encoding, which represents a valid plan for the original planning problem. In the encoding, the CSP variables correspond to the predicates that have to be achieved at different levels in the planning graph (different planning steps) and their possible values are the actions that can support the predicates. For each CSP variable representing a predicate p, there are two special values: i) $\perp$: indicates that a predicate is not supported by any action and is *false* at a particular level/planning-step; ii) "noop": indicates that the predicate is true at a given level i because it was made true at some previous level $j < i$ and no other action deletes p between j and i. Constraints encode the relations between predicates and actions: 1) mutual exclusion relations between predicates and actions; and 2) the causal relationships between actions and their preconditions. Figure 2.7 shows the CSP encoding corresponding the portion of the planning graph in Figure 2.6.

2.5.1.1 Adapting GP-CSP to Different Distance Metrics

When the above planning encoding is solved by any standard CSP solver, it will return a solution containing $\langle$*var, value*$\rangle$ of the form $\{\langle x_1, y_1\rangle, ...\langle x_n, y_n\rangle\}$. The collection of x_i where $y_i \neq \perp$ represents the facts that are made true at different time steps (plan trajectory) and can be used as a basis for the *state-based* distance measure;[9] the set of $(y_i \neq \perp) \wedge (y_i \neq noop)$ represents the set of actions in the plan and can be used for *action-based* distance measure; lastly, the assignments $\langle x_i, y_i\rangle$ themselves represent the causal relations and can be used for the *causal-based* distance measure.

However, there are some technical difficulties we need to overcome before a specific distance measure between plans can be computed. First, the same action can be represented by different values in the domains of different variables. Consider a simple example in which there are two facts p and q, both supported by two actions a_1 and a_2. When setting up the CSP encoding, we assume that the CSP variables x_1 and x_2 are used to

[9] We implement the state-based distance between plans as defined in Equation 2.6.

represent p and q. The domains for x_1 and x_2 are $\{v_{11}, v_{12}\}$ and $\{v_{21}, v_{22}\}$, both representing the two actions $\{a_1, a_2\}$ (in that order). The assignments $\{\langle x_1, v_{11}\rangle, \langle x_2, v_{21}\rangle\}$ and $\{\langle x_1, v_{12}\rangle, \langle x_2, v_{22}\rangle\}$ have a distance of 2 in traditional CSP because different values are assigned for each variable x_1 and x_2. However, they both represent the same action set $\{a_1, a_2\}$ and thus lead to the plan distance of 0 if we use the action-based distance in our plan comparison. Therefore, we first need to translate the set of values in all assignments back to the set of action instances before doing comparison using action-based distance. The second complication arises for the causal-based distance. A causal link $a_1 \rightarrow p - a_2$ between two actions a_1 and a_2 indicates that a_1 supports the precondition p of a_2. However, the CSP assignment $\langle p, a_1\rangle$ only provides the first half of each causal-link. To complete the causal-link, we need to look at the values of other CSP assignments to identify action a_2 that occurs at the later level in the planning graph and has p as its precondition. Note that there may be multiple "valid" sets of causal-links for a plan, and in the implementation we simply select causal-links based on the CSP assignments.

2.5.1.2 Making GP-CSP Return a Set of Plans

To make GP-CSP return a set of plans satisfying the *d*DISTANT*k*SET constraint using one of the three distance measures, we add "global" constraints to each original encoding to enforce d-diversity between every pair of solutions. When each global constraint is called upon by the normal forward checking and arc-consistency checking procedures inside the default solver to check if the distance between two plans is over a predefined value d, we first map each set of assignments to an actual set of actions (action-based), predicates that are true at different plan-steps (state-based) or causal-links (causal-based) using the method discussed in the previous section. This process is done by mapping all $\langle var, value\rangle$ CSP assignments into action sets using a call to the planning graph, which is outside of the CSP solver, but works closely with the general purpose CSP solver in GP-CSP. The comparison

is then done within the implementation of the global constraint to decide if two solutions are diverse enough.

We investigate two different ways to use the global constraints:

1. The *parallel* strategy to return the set of k plans all at once. In this approach, we create one encoding that contains k identical copies of each original planning encoding created using the GP-CSP planner. The k copies are connected together using $k(k-1)/2$ pair-wise global constraints. Each global constraint between the i^{th} and j^{th} copies ensures that two plans represented by the solutions of those two copies will be at least d distant from each other. If each copy has n variables, then this constraint involves $2n$ variables.

2. The *greedy* strategy to return plans one after another. In this approach, the k copies are not setup in parallel up-front, but sequentially. We add to the i^{th} copy one global constraint to enforce that the solution of the i^{th} copy should be d-diverse from any of the earlier $i-1$ solutions. The advantage of the greedy approach is that each CSP encoding is significantly smaller in terms of the number of variables (n vs. $k \times n$), smaller in terms of the number of global constraints (1 vs. $k(k-1)/2$), and each global constraint also contains lesser number of variables (n vs. $2 \times n$).[10] Thus, each encoding in the greedy approach is easier to solve. However, because each solution depends on all previously found solutions, the encoding can be unsolvable if the previously found solutions comprise a bad initial solution set.

2.5.1.3 Empirical Evaluation

We implemented the parallel and greedy approaches discussed earlier for the three distance measures and tested them with the benchmark set of Logistics problems provided with

[10] However, each constraint is more complicated because it encodes $(i-1)$ previously found solutions.

the Blackbox planner (Kautz and Selman, 1998). All experiments were run on a Linux Pentium 4, 3Ghz machine with 512MB RAM. For each problem, we test with different d values ranging from 0.01 (1%) to 0.95 (95%)[11] and k increases from 2 to n where n is the maximum value for which GP-CSP can still find solutions within the plan horizon. The horizon (parallel plan steps) limit is 30.

We found that the greedy approach outperformed the parallel approach and solved significantly higher number of problems. Therefore, we focus on the greedy approach hereafter. For each combination of d, k, and a given distance measure, we record the solving time and output the average/min/max pairwise distances of the solution sets.

Baseline Comparison: As a baseline comparison, we have also implemented a *randomized* approach. In this approach, we do not use global constraints but use random value ordering in the CSP solver to generate k different solutions without enforcing them to be pairwise d-distance apart. For each distance d, we continue running the random algorithm until we find k_{max} solutions where k_{max} is the maximum value of k that we can solve for the greedy approach for that particular d value. In general, we want to compare with our approach of using global constraint to see if the random approach can effectively generate diverse set of solutions by looking at: (1) the average time to find a solution in the solution set; and (2) the maximum/average pairwise distances between $k \geq 2$ randomly generated solutions.

Table 2.2 shows the comparison of average solving time to find one solution in the greedy and random approaches. The results show that on average, the random approach takes significantly more time to find a single solution, regardless of the distance measure used by the greedy approach. To assess the diversity in the solution sets, Table 2.3 shows the comparison of: (1) the average pairwise minimum distance between the solutions in sets returned by the random approach; and (2) the maximum d for which the greedy approach still can find a set of diverse plans. The comparisons are done for all three distance

[11] Increments of 0.01 from 0.01 to 0.1 and of 0.05 thereafter.

	log-easy	**rocket-a**	**log-a**	**log-b**	**log-c**	**log-d**
δ_a	0.087	7.648	1.021	6.144	8.083	178.633
δ_s	0.077	9.354	1.845	6.312	8.667	232.475
δ_{cl}	0.190	6.542	1.063	6.314	8.437	209.287
Random	0.327	15.480	8.982	88.040	379.182	6105.510

Table 2.2: Average solving time (in seconds) to find a plan using greedy (first 3 rows) and by random (last row) approaches

	log-easy	**rocket-a**	**log-a**	**log-b**	**log-c**	**log-d**
δ_a	0.041/0.35	0.067/0.65	0.067/0.25	0.131/0.1*	0.126/0.15	0.128/0.2
δ_s	0.035/0.4	0.05/0.8	0.096/0.5	0.147/0.4	0.140/0.5	0.101/0.5
δ_{cl}	0.158/0.8	0.136/0.95	0.256/0.55	0.459/0.15*	0.346/0.3*	0.349/0.45

Table 2.3: Comparison of the diversity in the plan sets returned by the random and greedy approaches. Cases where random approach is better than greedy approach are marked with *.

measures. For example, the first cell $(0.041/0.35)$ in Table 2.3, implies that the minimum pairwise distance averaged for all solvable $k \geq 2$ using the random approach is $d = 0.041$ while it is 0.35 (i.e., 8x more diverse) for the greedy approach using the δ_a distance measure. Except for 3 cases, using global constraints to enforce minimum pairwise distance between solutions helps GP-CSP return significantly more diverse set of solutions. On average, the greedy approach returns 4.25x, 7.31x, and 2.79x more diverse solutions than the random approach for δ_a, δ_s and δ_{cl}, respectively.

Analysis of the different distance-bases: Overall, we were able to solve 1264 (d, k) combinations for three distance measures $\delta_a, \delta_s, \delta_{cl}$ using the greedy approach. We were partic-

ularly interested in investigating the following issues:

- **Q1: Computational efficiency** - Is it easy or difficult to find a set of diverse solutions using different distance measures? Thus, (1) for the same d and k values, which distance measure is more difficult (time consuming) to solve; and (2) given an encoding horizon limit, how high is the value of d and k for which we can still find a set of solutions for a given problem using different distance measures.

- **Q2: Solution diversity** - What, if any, is the correlation/sensitivity between different distance measures? Thus, how the comparative diversity of solutions is when using different distance measures.

Regarding *Q1*, Table 2.4 shows the highest solvable k value for each distance d and base δ_a, δ_s, and δ_{cl}. For a given (d, k) pair, enforcing δ_a appears to be the most difficult, then δ_s, and δ_{cl} is the easiest. GP-CSP is able to solve 237, 462, and 565 combinations of (d, k) respectively for δ_a, δ_s and δ_{cl}. GP-CSP solves *d*DISTANT*k*SET problems more easily with δ_s and δ_{cl} than with δ_a due to the fact that solutions with different action sets (diverse with regard to δ_a) will likely cause different trajectories and causal structures (diverse with regard to δ_s and δ_{cl}). Between δ_s and δ_{cl}, δ_{cl} solves more problems for easier instances (log-easy, rocket-a and log-a) but less for the harder instances, as shown in Table 2.4. We conjecture that for solutions with more actions (i.e., in bigger problems) there are more causal dependencies between actions and thus it is harder to reorder actions to create a different causal-structure.

For running time comparisons, among 216 combinations of (d, k) that were solved by all three distance measures, GP-CSP takes the least amount of time for δ_a in 84 combinations, for δ_s in 70 combinations and in 62 for δ_{cl}. The first three lines of Table 2.2 show the average time to find one solution in d-diverse k-set for each problem using δ_a, δ_s and δ_{cl} (which we call t_a, t_s and t_c respectively). In general, t_a is the smallest and $t_s > t_c$ in most

d	log-easy	rocket-a	log-a	log-b	log-c	log-d
0.01	11,5, **28**	8,**18**,12	9,8,**18**	3,4,**5**	4,6,**8**	**8**,7,7
0.03	6,3,**24**	8,**13**,9	7,7,**12**	2,**4**,3	4,**6**,**6**	4,**7**,6
0.05	5,3,**18**	6,**11**,9	5,7,**10**	2,**4**,3	4,**6**,5	3,**7**,5
0.07	2,3,**14**	6,**10**,8	4,**7**,6	2,**4**,2	4,**6**,5	3,**7**,5
0.09	2,3,**14**	6,**9**,6	3,**6**,**6**	2,**4**,2	3,**6**,4	3,**7**,4
0.1	2,3,**10**	6,**9**,6	3,**6**,**6**	2,**4**,2	2,**6**,4	3,**7**,4
0.2	2,3,**5**	5,**9**,6	2,**6**,**6**	1,**3**,1	1,**5**,2	2,**5**,3
0.3	2,2,**3**	4,**7**,5	1,**4**,**4**	1,**2**,1	1,**3**,2	1,**3**,**3**
0.4	1,2,**3**	3,**6**,5	1,**3**,**3**	1,**2**,1	1,**2**,1	1,2,**3**
0.5	1,1,**3**	2,4,**5**	1,**2**,**2**	-	1,**2**,1	1,**2**,1
0.6	1,1,**2**	2,3,**4**	-	-	-	-
0.7	1,1,**2**	1,**2**,**2**	-	-	-	-
0.8	1,1,**2**	1,**2**,**2**	-	-	-	-
0.9	-	1,1,**2**	-	-	-	-

Table 2.4: For each given d value, each cell shows the largest solvable k for each of the three distance measures δ_a, δ_s, and δ_{cl} (in this order). The maximum values in cells are in bold.

problems. Thus, while it is harder to enforce δ_a than δ_s and δ_{cl} (as indicated in Table 2.4), when the encodings for all three distances can be solved for a given (d, k), then δ_a takes less time to search for one plan in the diverse plan set; this can be due to tighter constraints (more pruning power for the global constraints) and simpler global constraint setting.

To test *Q2*, in Table 2.5, we show the cross-comparison between different distance measures δ_a, δ_s, and δ_{cl}. In this table, cell $\langle$*row, column*$\rangle$ $= \langle \delta', \delta'' \rangle$ indicates that over all combinations of (d, k) solved for distance δ', the average value d''/d' where d'' and

	δ_a	δ_s	δ_{cl}
δ_a	-	1.262	1.985
δ_s	0.485	-	0.883
δ_{cl}	0.461	0.938	-

Table 2.5: Cross-validation of distance measures δ_a, δ_s, and δ_{cl}.

d' are distance measured according to δ'' and δ', respectively ($d' \geq d$). For example, $\langle \delta_s, \delta_a \rangle = 0.485$ means that over 462 combinations of (d, k) solvable for δ_s, for each d, the average distance between k solutions measured by δ_a is $0.485 \times d_s$. The results indicate that when we enforce d for δ_a, we will likely find even more diverse solution sets according to δ_s ($1.26 \times d_a$) and δ_{cl} ($1.98 \times d_a$). However, when we enforce d for either δ_s or δ_{cl}, we are not likely to find a more diverse set of solutions measured by the other two distance measures. Nevertheless, enforcing d using δ_{cl} will likely give comparable diverse degree d for δ_s ($0.94 \times d_c$) and vice versa. We also observe that d_s is highly dependent on the difference between the parallel lengths of plans in the set. The distance d_s seems to be the smallest (i.e., $d_s < d_a < d_c$) when all k plans have the same/similar number of time steps. This is consistent with the fact that δ_a and δ_{cl} do not depend on the steps in the plan execution trajectory while δ_s does.

2.5.2 Finding Diverse Plan Set with LPG

In this section, we consider the problem of generating diverse sets of plans using another planning approach, in particular the LPG planner which is able to scale up to bigger problems, compared to GP-CSP. We focus on the action-based distance measure between plans, which has been shown in the previous section to be the most difficult to enforce diversity. LPG is a local-search-based planner, that incrementally modifies a partial plan in a search for a plan that contains no flaws (Gerevini *et al.*, 2003). The behavior of LPG is con-

trolled by an evaluation function that is used to select between different plan candidates in a neighborhood generated for local search. At each search step, the elements in the search neighborhood of the current partial plan π are the alternative possible plans repairing a selected flaw in π. The elements of the neighborhood are evaluated according to an *action evaluation function* E (Gerevini *et al.*, 2003). This function is used to estimate the cost of either adding or of removing an action node a in the partial plan p being generated.

2.5.2.1 Revised Evaluation Function for Diverse Plans

In order to manage dDISTANCEkSET problems, the function E has been extended to include an additional evaluation term that has the purpose of penalizing the insertion and removal of actions that *decrease* the distance of the current partial plan p under adaptation from a reference plan p_0. In general, E consists of four weighted terms, evaluating four aspects of the quality of the current plan that are affected by the addition ($E(a)^i$) or removal ($E(a)^r$) of a

$$E(a)^i = \alpha_E \cdot Execution\_cost(a)^i + \alpha_T \cdot Temporal\_cost(a)^i + \\ + \alpha_S \cdot Search\_cost(a)^i + \alpha_D \cdot |(p_0 - p) \cap p^i_{\mathsf{R}}|,$$

$$E(a)^r = \alpha_E \cdot Execution\_cost(a)^r + \alpha_T \cdot Temporal\_cost(a)^r + \\ + \alpha_S \cdot Search\_cost(a)^r + \alpha_D \cdot |(p_0 - p - a) \cap p^r_{\mathsf{R}}|.$$

The first three terms of the two forms of E are unchanged from the standard behavior of LPG. The fourth term, used only for computing diverse plans, is the new term estimating how the proposed plan modification will affect the distance from the reference plan p_0. Each cost term in E is computed using a relaxed temporal plan p_R (Gerevini *et al.*, 2003).

The p_R plans are computed by an algorithm, called RelaxedPlan, formally described and illustrated in (Gerevini *et al.*, 2003). We have slightly modified this algorithm to penalize the selection of actions decreasing the plan distance from the reference plan. The specific change to RelaxedPlan for computing diverse plans is very similar to the change described in (Fox *et al.*, 2006a), and it concerns the heuristic function for selecting the actions for achieving the subgoals in the relaxed plans. In the modified function for RelaxedPlan, we have an extra 0/1 term that penalizes an action b for p_R if its addition decreases the distance of $p + p_R$ from p_0 (in the plan repair context investigated in (Fox *et al.*, 2006a), b is penalized if its addition *increases* such a distance).

The last term of the modified evaluation function E is a measure of the decrease in plan distance caused by adding or removing a: $|(p_0 - p) \cap p_R^i|$ or $|(p_0 - p - a) \cap p_R^r|$, where p_R^i contains the new action a. The α-coefficients of the E-terms are used to weigh their relative importance.[12] The values of the first 3 terms are automatically derived from the expression defining the plan metric for the problem (Gerevini *et al.*, 2003). The coefficient for the fourth new term of E (α_D) is automatically set during search to a value proportional to $d/\delta_a(p, p_0)$, where p is the current partial plan under construction. The general idea is to dynamically increase the value of α_D according to the number of plans n that have been generated so far: if n is much higher than k, the search process consists of finding many solutions with not enough diversification, and hence the importance of the last E-term should increase.

2.5.2.2 Making LPG Return a Set of Plans

In order to compute a set of k d-distant plans solving a dDISTANCEkSET problem, we run LPG multiple times, until the problem is solved, with the following two additional changes

[12] These coefficients are also normalized to a value in $[0, 1]$ using the method described in (Gerevini *et al.*, 2003).

to the standard version of LPG: (i) the preprocessing phase computing mutex relations and other reachability information exploited during the relaxed plan construction is done only once for all runs; (ii) we maintain an incremental set of valid plans, and we dynamically select one of them as the reference plan p_0 for the next search. Concerning (ii), let $\mathcal{P} = \{p_1, ..., p_n\}$ be the set of n valid plans that have been computed so far, and ***CPlans***(p_i) the subset of $\mathcal{P}$ containing all plans that have a distance greater than or equal to d from a reference plan $p_i \in P$.

The reference plan p_0 used in the modified heuristic function E is a plan $p_{max} \in \mathcal{P}$ which has a maximal set of diverse plans in $\mathcal{P}$, i.e.,

$$p_{max} = \underset{p_i \in \mathcal{P}}{\operatorname{argmax}} \{|CPlans(p_i)|\}. \tag{2.13}$$

The plan p_{max} is incrementally computed each time the local search finds a new solution. In addition to being used to identify the reference plan in E, p_{max} is also used for defining the initial state (partial plan) of the search process. Specifically, we initialize the search using a (partial) plan obtained by randomly removing some actions from a (randomly selected) plan in the set ***CPlans***$(p_{max}) \cup \{p_{max}\}$.

The process of generating diverse plans starting from a dynamically chosen reference plan continues until at least k plans that are all d-distant from each other have been produced. The modified version of LPG to compute diverse plans is called LPG-d.

2.5.2.3 Experimental Analysis with LPG-d

Recall that the distance function δ_a, using set-difference, can be written as the sum of two terms:

$$\delta_a(p_i, p_j) = \frac{|A(p_i) - A(p_j)|}{|A(p_i) \cup A(p_j)|} + \frac{|A(p_j) - A(p_i)|}{|A(p_i) \cup A(p_j)|}. \tag{2.14}$$

The first term represents the contribution of the actions in p_i to the plan difference, while the second term indicates the contribution of p_j to δ_a. We experimentally observed that in some cases the differences between two diverse plans computed using δ_a are mostly concentrated in only one of the δ_a components. This asymmetry means that one of the two plans can have many more actions than the other one, which could imply that the quality of one of the two plans is much worse than the quality of the other plan. In order to avoid this problem, we can parametrize δ_a by imposing the two extra constraints

$$\delta_a^A \geq d/\gamma \text{ and } \delta_a^B \geq d/\gamma,$$

where δ_a^A and δ_a^B are the first and second terms of the RHS of Equation 2.14, respectively, and γ is an integer parameter "balancing" the diversity of p_i and p_j.

In this section, we analyze the performance of LPG-d in four different benchmark domains: DriverLog, Satellite, Storage and FloorTile from the 3rd, 5th and 7th IPCs.[13] The main goals of the experimental evaluation were (i) showing that LPG-d can efficiently solve a large set of (d, k)-combinations, (ii) investigating the impact of the δ_a γ-constraints on performance, (iii) comparing LPG-d and the standard LPG.

We tested LPG-d using both the default and parametrized versions of δ_a, with $\gamma = 2$ and $\gamma = 3$. We give detailed results for $\gamma = 3$ and a more general evaluation for $\gamma = 2$ and the original δ_a. We consider d that varies from 0.05 to 0.95, using 0.05 increment step, and with $k = 2...5$, 6, 8, 10, 12, 14, 16, 20, 24, 28, 32 (overall, a total of 266 (d, k)-combinations). Since LPG-d is a stochastic planner, we use the median of the CPU times (in seconds) and the median of the average plan distances (over five runs). The average plan distance for a set of k plans solving a specific (d, k)-combination (δ^{av}) is the average of the plans distances between all pairs of plans in the set. The tests were performed on an Intel(R) Xeon(TM) CPU 3.00 GHz, 3Gb RAM. The CPU-time limit was 300 seconds.

[13] We have similar results for other domains: Rovers (IPC3-5), Pathways (IPC5), Logistics (IPC2), Zeno-Travel (IPC3).

Figure 2.8 gives the results for the largest problem in the IPC-3 DriverLog-Time domain (fully-automated track). LPG-d solves 161 (d, k)-combinations, including combinations $d \leq 0.4$ and $k = 20$, and $d = 0.95$ and $k = 2$. The average CPU time (top plots) is 151.85 seconds. The average δ^{av} (bottom plots) is 0.73, with δ^{av} always greater than 0.57. With the original δ_a function LPG-d solves 168 (d, k)-combinations, the average CPU time is 149.5 seconds, and the average δ^{av} is 0.73; while with $\gamma = 2$ LPG-d solves 139 combinations, the average CPU time is 144.2 seconds, and the average δ^{av} is 0.72.

Figure 2.9 shows the results for the largest problem in the IPC-3 Satellite-Strips domain. LPG-d solves 242 (k, d)-combinations; 153 of them require less than 10 seconds. The average CPU time is 5.46 seconds, and the average δ^{av} is 0.69. We observed similar results when using the original δ_a function or the parametrized δ_a with $\gamma = 2$ (in the second case, LPG-d solves 230 problems, while the average CPU time and the average δ^{av} are nearly the same as with $\gamma = 3$).

Figure 2.10 shows the results for a middle-size problem in the IPC-5 Storage-Propositional domain. With $\gamma = 3$ LPG-d solves 252 (k, d)-combinations, 58 of which require less than 10 seconds, while 178 of them require less than 50 seconds. The average CPU time is 25.4 seconds and the average δ^{av} is 0.91. With the original δ_a, LPG-d solves 257 (k, d)-combinations, the average CPU time is 14.5 seconds, and the average δ^{av} is 0.9; with $\gamma = 2$, LPG-d solves 201 combinations, the average CPU time is 31 seconds and the average δ^{av} is 0.93.

Figure 2.11 gives the results for the largest problem in the IPC-7 FloorTile-MetricTime domain. LPG-d solves 210 (d, k)-combinations; 171 of them require less than 10 seconds. The average CPU time is 3.6 seconds, and the average δ^{av} is 0.7. We observed similar results when using the original δ_a function or the parametrized δ_a with $\gamma = 2$ (in the second case, LPG-d solves 191 problems, while the average CPU time and the average δ^{av} are nearly the same as with $\gamma = 3$).

The local search in LPG is randomized by a "noise" parameter that is automatically set and updated during search (Gerevini *et al.*, 2003). This randomization is one of the techniques used for escaping local minima, but it also can be useful for computing diverse plans: if we run the search multiple times, each search is likely to consider different portions of the search space, which can lead to different solutions. It is then interesting to compare LPG-d and a method in which we simply run the standard LPG until k d-diverse plans are generated. An experimental comparison of the two approaches show that in many cases LPG-d performs better. In particular, the new evaluation function E is especially useful for planning problems that are easy to solve for the standard LPG, and that admit many solutions. In these cases, the original E function produces many valid plans with not enough diversification. This problem is significantly alleviated by the new term in E. An example of domain where we observed this behavior is Logistics.[14]

2.6 Generating Plan Sets with Partial Preference Knowledge

In this section, we consider the problem of generating plan sets when the user preferences are only partially expressed. In particular, we focus on metric temporal planning where the preference model is assumed to be represented by an incomplete value function specified by a convex combination of two features: *plan makespan* and *execution cost*, with the exact trade-off value w drawn from a given distribution. The quality value of plan sets is measured by the ICP value, as formalized in Equation 2.12. Our objective is to find a set of plans $\mathcal{P} \subseteq \mathcal{S}$ where $|\mathcal{P}| \leq k$ and $ICP(\mathcal{P})$ is the lowest.

Notice that we restrict the size of the solution set returned, not only for the compre-

[14]E.g., LPG-d solved 176 instances for the log_a problem, 47 of them in less than 1 CPU second and 118 of them in less than 10 CPU seconds; the average CPU time was 3.75 seconds and the average δ^{av} was 0.47. While using the standard LPG, only 107 instances were solved, 27 of them in less than 1 CPU seconds and 73 of them in less than 10 CPU seconds; the average CPU time was 5.14 seconds and the average δ^{av} was 0.33.

hension issue discussed earlier, but also for an important property of the ICP measure: it is a monotonically non-increasing function of the solution set (specifically, given two solution sets $\mathcal{P}_1$ and $\mathcal{P}_2$ such that the latter is a superset of the former, it is easy to see that $ICP(\mathcal{P}_2) \leq ICP(\mathcal{P}_1)$).

2.6.1 Sampling Weight Values

Given that the distribution of trade-off value w is known, the straightforward way to find a set of representative solutions is to first sample a set of k values for w: $\{w_1, w_2, ..., w_k\}$ based on the distribution $h(w)$. For each value w_i, we can find an (optimal) plan p_i minimizing the value of the overall value function $V(p, w_i) = w_i \times t_p + (1 - w_i) \times c_p$. The final set of solutions $\mathcal{P} = \{p_1, p_2, ...p_k\}$ is then filtered to remove duplicates and dominated solutions, thus selecting the plans making up the lower-convex hull. The final set can then be returned to the user. While intuitive and easy to implement, this sampling-based approach has several potential flaws that can limit the quality of its resulting plan set.

First, given that k solution plans are searched sequentially and independently of each other, even if the plan p_i found for each w_i is optimal, the final solution set $\mathcal{P} = \{p_1, p_2...p_k\}$ may not even be the optimal set of k solutions with regard to the ICP measure. More specifically, for a given set of solutions $\mathcal{P}$, some trade-off value w, and two non-dominated plans p, q such that $V(p, w) < V(q, w)$, it is possible that $ICP(\mathcal{P} \cup \{p\}) > ICP(\mathcal{P} \cup \{q\})$. In our running example (Figure 2.5), let $\mathcal{P} = \{p_2, p_5\}$ and $w = 0.8$ then $V(p_1, w) = 0.8 \times 4 + 0.2 \times 25 = 8.2 < V(p_7, w) = 0.8 \times 12 + 0.2 \times 5 = 10.6$. Thus, the planner will select p_1 to add to $\mathcal{P}$ because it looks locally better given the weight $w = 0.8$. However, $ICP(\{p_1, p_2, p_5\}) \approx 10.05 > ICP(\{p_2, p_5, p_7\}) \approx 7.71$ so indeed by taking previous set into consideration then p_7 is a much better choice than p_1.

Second, the values of the trade-off parameter w are sampled based on a given distribution, and independently of the particular planning problem being solved. As there is no

relation between the sampled w values and the solution space of a given planning problem, sampling approach may return very few distinct solutions even if we sample a large number of weight values w. In our example, if all w samples have values $w \leq 0.67$ then the optimal solution returned for any of them will always be p_7. However, we know that $\mathcal{P}^* = \{p_1, p_3, p_7\}$ is the optimal set according to the ICP measure. Indeed, if $w \leq 0.769$ then the sampling approach can only find the set $\{p_7\}$ or $\{p_3, p_7\}$ and still not be able to find the optimal set $\mathcal{P}^*$.

2.6.2 ICP Sequential Approach

Given the potential drawbacks of the sampling approach outlined above, we also pursued an alternative approach that takes into account the ICP measure more actively. Specifically, we incrementally build the solution set $\mathcal{P}$ by finding a solution p such that $\mathcal{P} \cup \{p\}$ has the lowest ICP value. We can start with an empty solution set $\mathcal{P} = \emptyset$, then at each step try to find a new plan p such that $\mathcal{P} \cup \{p\}$ has the lowest ICP value.

While this approach directly takes the ICP measure into consideration at each step of finding a new plan and avoids the drawbacks of the sampling-based approach, it also has its own share of potential flaws. Given that the set is built incrementally, the earlier steps where the first "seed" solutions are found are very important. The closer the seed solutions are to the global lower convex hull, the better the improvement in the ICP value. In our example (Figure 2.5), if the first plan found is p_2 then the subsequent plans found to best extend $\{p_2\}$ can be p_5 and thus the final set does not come close to the optimal set $\mathcal{P}^* = \{p_1, p_3, p_7\}$.

2.6.3 Hybrid Approach

In this approach, we aim to combine the strengths of both the sampling and ICP-sequential approaches. Specifically, we use sampling to find several plans optimizing for different weights. The plans are then used to seed the subsequent ICP-sequential runs. By seeding

Algorithm 1: Incrementally find solution set $\mathcal{P}$

1 **Input:** A planning problem with a solution space $\mathcal{S}$; maximum number of plans required k; number of sampled trade-off values k_0 ($0 < k_0 < k$); time bound t;
2 **Output**: A plan set $\mathcal{P}$ ($|\mathcal{P}| \leq k$);
3 **begin**
4 $W \leftarrow$ sample k_0 values for w;
5 $\mathcal{P} \leftarrow$ find good quality plans in $\mathcal{S}$ for each $w \in W$;
6 **while** $|\mathcal{P}| < k$ *and* $search\_time < t$ **do**
7 Search for p s.t. $ICP(\mathcal{P} \cup \{p\}) < ICP(\mathcal{P})$
8 $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$
9 **end**
10 Return $\mathcal{P}$
11 **end**

the hybrid approach with good quality plan set scattered across the pareto optimal set, we hope to gradually expand the initial set to a final set with a much better overall ICP value. Algorithm 1 shows the pseudo-code for the hybrid approach. We first independently sample the set of k_0 values (with k_0 pre-determined) of w given the distribution on w (step 4). We then run a heuristic planner multiple times to find an optimal (or good quality) solution for each trade-off value w (step 5). We then collect the plans found and seed the subsequent runs when we incrementally update the initial plan set with plans that lower the overall ICP value (steps 6-8). The algorithm terminates and returns the latest plan set (step 9) if k plans are found or the time bound exceeds.

2.6.4 Making LPG Search Sensitive to ICP

We use a modified version of the Metric-LPG planner (Gerevini *et al.*, 2008) in implementing our algorithms, introducing the $totalcost$ numerical fluent into the domain to represent the plan cost that we are interested in.[15] Not only is Metric-LPG equipped with a very flexible local-search framework that has been extended to handle various objective functions, but it can also be made to search for single or multiple solutions. Specifically, for the sampling-based approach, we first sample the w values based on a given distribution. For each w value, we set the metric function in the domain file to: $w \times makespan + (1 - w) \times totalcost$, and run the original LPG in the quality mode to heuristically find the best solution within the time limit for that metric function. The final solution set is filtered to remove any duplicate solutions, and returned to the user.

For the ICP-sequential and hybrid approach, we can not use the original LPG implementation as is and need to modify the neighborhood evaluation function in LPG to take into account the ICP measure and the current plan set $\mathcal{P}$. For the rest of this section, we will explain this procedure in detail.

Background: Metric-LPG uses local search to find plans within the space of *numerical action graphs* (NA-graph). This leveled graph consists of a sequence of interleaved proposition and action layers. The proposition layers consist of a set of propositional and numerical nodes, while each action layer consists of at most one action node, and a number of no-op links. An NA-graph G represents a valid plan if all actions' preconditions are supported by some actions appearing in the earlier level in G. The search neighborhood for each local-search step is defined by a set of graph modifications to fix some remaining inconsistencies (unsupported preconditions) p at a particular level l. This can be done by

[15] Although we are interested in the plan cost as summation of action costs, our implementation can also be extended for planning problems where plan cost is an expression involving numerical fluents.

either inserting a new action a supporting p or removing from the graph the action a that p is a precondition of (which can introduce new inconsistencies).

Each local move creates a new NA-graph G', which is evaluated as a weighted combination of two factors: $SearchCost(G')$ and $ExecCost(G')$. Here, $SearchCost(G')$ is the amount of search effort to resolve inconsistencies newly introduced by inserting or removing action a; it is measured by the number of actions in a relaxed plan R resolving all such inconsistencies. The total cost $ExecCost(G')$, which is a default function to measure plan quality, is measured by the total *action execution costs* of all actions in R. The two weight adjustment values α and β are used to steer the search toward either finding a solution quickly (higher α value) or better solution quality (higher β value). Metric-LPG then selects the local move leading to the smallest $E(G')$ value.

Adjusting the evaluation function $E(G')$ for finding set of plans with low ICP measure: To guide Metric-LPG towards optimizing our ICP-sensitive objective function instead of the original minimizing cost objective function, we need to replace the default plan quality measure $ExecCost(G')$ with a new measure $ICPEst(G')$. Specifically, we adjust the function for evaluating each new NA-graph generated by local moves at each step to be a combination of $SearchCost(G')$ and $ICPEst(G')$. Given the set of found plans $\mathcal{P} = \{p_1, p_2, ..., p_n\}$, $ICPEst(G')$ guides Metric-LPG's search toward a plan p generated from G' such that the resulting set $\mathcal{P} \cup \{p\}$ has a minimum ICP value: $p = \underset{p}{\operatorname{argmin}}\, ICP(\mathcal{P} \cup \{p\})$. Thus, $ICPEst(G')$ estimates the expected total ICP value if the best plan p found by expanding G' is added to the current found plan set $\mathcal{P}$. Like the original Metric-LPG, p is estimated by $p_R = G' \bigcup R$ where R is the relaxed plan resolving inconsistencies in G' caused by inserting or removing a. The $ICPEst(G')$ for a given NA-graph G' is calculated as: $ICPEst(G') = ICP(\mathcal{P} \cup p_R)$ with the ICP measure as described in Equation 2.12. Notice here that while $\mathcal{P}$ is the set of valid plans, p_R is not. It is an invalid plan represented by a NA-graph containing some unsupported preconditions. However, Equation 2.12 is

still applicable as long as we can measure the time and cost dimensions of p_R. To measure the makespan of p_R, we estimate the time points at which unsupported facts in G' would be supported in $p_R = G' \cup R$ and propagate them over actions in G' to its last level. We then take the earliest time point at which all facts at the last level appear to measure the makespan of p_R. For the cost measure, we just sum the individual costs of all actions in p_R. At each step of Metric-LPG's local search framework, combining two measures $ICPEst(G')$ and $SearchCost(G')$ gives us an evaluation function that fits right into the original Metric-LPG framework and prefers a NA-graph G' in the neighborhood of G that gives the best trade-off between the estimated effort to repair and the estimated decrease in quality of the next resulting plan set.

2.6.5 Experimental Results

We have implemented several approaches based on our algorithms discussed in the previous sections: Sampling (Section 2.6.1), ICP-sequential (Section 2.6.2) and Hybrid that combines both (Section 2.6.3) with both the uniform and triangular distributions. We consider two types of distributions in which the most probable weight for plan makespan are 0.2 and 0.8, which we will call "w02" and "w08" distributions respectively (Figure 2.12 shows these distributions). We test all implementations against a set of 20 problems in each of several benchmark temporal planning domains used in the previous International Planning Competitions (IPC): ZenoTravel, DriverLog, and Depots. The only modification to the original benchmark set is the added action costs. The descriptions of these domains can be found at the IPC website (*ipc.icaps-conference.org*). The experiments were conducted on an Intel Core2 Duo machine with 3.16GHz CPU and 4Gb RAM. For all approaches, we search for a maximum of $k = 10$ plans within the 10-minute time limit for each problem (i.e., $t = 10$ minutes), and the resulting plan set is used to compute the ICP value. In the Sampling approach, we generate ten trade-off values w between *makespan* and *plan cost*

based on the distribution, and for each one we search for a plan p subject to the value function $V(p, w) = w \times t_p + (1 - w) \times c_p$. In the Hybrid approach, on the other hand, the first Sampling approach is used with $k_0 = 3$ generated trade-off values to find an initial plan set, which is then improved by the ICP-Sequential runs. As Metric-LPG is a stochastic local search planner, we run it three times for each problem and average the results. In 77% and 70% of 60 problems in the three tested domains for the Hybrid and Sampling approaches respectively, the standard deviation of ICP values of plan sets are at most 5% of the average values. This indicates that ICP values of plan set in different runs are quite stable. As the Hybrid approach is an improved version of ICP-sequential and gives better results in almost all tested problems, we omit the ICP-Sequential in discussions below. We now analyze the results in more detailed.

The utility of using the partial knowledge of user's preferences: To evaluate the utility of taking partial knowledge of user preferences into account, we first compare our results against the naive approaches that generate a plan set without explicitly taking into account the partial knowledge. Specifically, we run the default LPG planner with different random seeds to find multiple non-dominated plans. The LPG planner was run with both *speed* setting, which finds plans quickly, and *diverse* setting, which takes longer time to find better set of diverse plans. Figure 2.13 shows the comparison between quality of plan sets returned by Sampling and those naive approaches when the distribution of the trade-off value w between *makespan* and *plan cost* is assumed to be uniform. Overall, among 20 tested problems for each of the ZenoTravel, DriverLog, and Depots domains, the Sampling approach is better than LPG-speed in 19/20, 20/20 and 20/20 and is better than LPG-d in 18/20, 18/20, and 20/20 problems respectively. We observed similar results comparing Hybrid and those two approaches: in particular, the Hybrid approach is better than LPG-speed in all 60 problems and better than LPG-d in 19/20, 18/20, and 20/20 problems respectively.

These results support our intuition that taking into account the partial knowledge about user preferences (if it is available) increases the quality of plan set.

Comparing the Sampling and Hybrid approaches: We now compare the effectiveness of the Sampling and Hybrid approaches in terms of the quality of returned plan sets with the uniform, w02 and w08 distributions.

ICP value: We first compare the two approaches in terms of the ICP values of plan sets returned indicating their quality evaluated by the user. Table 2.6, 2.7, and 2.8 show the results in the three domains. In general, Hybrid tends to be better than Sampling in this criterion for most of the domains and distributions. In particular, in the ZenoTravel domain it returns higher quality plan sets in 15/20 problems when the distribution is uniform, 10/20 and 13/20 problems when it is w02 and w08 respectively (both approaches return plan sets with equal ICP values for two problems with the w02 distribution and one problem with the w08 distribution). In the DriverLog domain, Hybrid returns better plan sets for 11/20 problems with the uniform distribution (and for other three problems the plan sets have equal ICP values), but worse with the triangular distributions: 8/20 (another 2 equals) and 9/20 (another one equals) with w02 and w08. The improvement on the quality of plan sets that Hybrid contributes is more significant in the Depots domain: it is better than Sampling in 11/20 problems with the uniform distribution (and equal in 3 problems), in 12/20 problems with the w02 and w08 distributions (with w02 both approaches return plan sets with equal ICP values for 4 problems, and for 2 problems when it is w08).

In many large problems of the ZenoTravel and DriverLog domains where Sampling performs better than Hybrid, we notice that the first phase of the Hybrid approach that searches for the first 3 initial plans normally takes most of the allocated time, and therefore there is not much time left for the second phase to improve the quality of plan set. We also observe that among the three settings of the trade-off distributions, the positive effect of the second phase in Hybrid approach (which is to improve the quality of the initial plan sets) tends to

Prob	Sampling	Hybrid	Prob	Sampling	Hybrid	Prob	Sampling	Hybrid
1*	840.00	839.98	1	972.00	972.00	1	708.00	708.00
2*	2,661.43	2,661.25	2	3,067.20	3,067.20	2*	2,255.792	2,255.788
3*	1,807.84	1,805.95	3*	2,083.91	2,083.83	3*	1,535.54	1,535.32
4*	3,481.31	3,477.49	4*	4,052.75	4,026.92	4*	2,960.84	2,947.66
5*	3,007.97	2,743.85	5*	3,171.86	3,171.73	5*	2,782.16	2,326.94
6*	3,447.37	2,755.25	6*	4,288.00	3,188.61	6*	2,802.00	2,524.18
7*	4,006.38	3,793.44	7*	4,644.40	4,377.40	7*	3,546.95	3,235.63
8*	4,549.90	4,344.70	8*	5,060.81	5,044.43	8*	3,802.60	3,733.90
9*	6,397.32	5,875.13	9*	7,037.87	6,614.30	9*	5,469.24	5,040.88
10*	7,592.72	6,826.60	10*	9,064.40	7,472.37	10*	6,142.68	5,997.45
11*	5,307.04	5,050.07	11*	5,946.68	5,891.76	11*	4,578.09	4,408.36
12*	7,288.54	6,807.28	12*	7,954.74	7,586.28	12	5,483.19	5,756.89
13*	10,208.11	9,956.94	13*	11,847.13	11,414.88	13*	8,515.74	8,479.09
14	11,939.22	13,730.87	14	14,474.00	15,739.19	14*	11,610.38	11,369.46
15	9,334.68	13,541.28	15	16,125.70	16,147.28	15*	11,748.45	11,418.59
16*	16,724.21	13,949.26	16	19,386.00	19,841.67	16	14,503.79	15,121.77
17*	27,085.57	26,822.37	17	29,559.03	32,175.66	17	21,354.78	22,297.65
18	23,610.71	25,089.40	18	28,520.17	29,020.15	18	20,107.03	21,727.75
19	29,114.30	29,276.09	19	34,224.02	36,496.40	19	23,721.90	25,222.24
20	34,939.27	37,166.29	20	39,443.66	42,790.97	20	28,178.45	28,961.51
	(a)			(b)			(c)	

Table 2.6: The ICP value of plan sets in the ZenoTravel domain returned by the Sampling and Hybrid approaches with the distributions (a) uniform, (b) w02 and (c) w08. The problems where Hybrid returns plan sets with better quality than Sampling are marked with *.

Prob	Sampling	Hybrid	Prob	Sampling	Hybrid	Prob	Sampling	Hybrid
1	212.00	212.00	1	235.99	236.00	1	188.00	188.00
2*	363.30	348.38	2*	450.07	398.46	2*	333.20	299.70
3	176.00	176.00	3	203.20	203.20	3	148.80	148.80
4*	282.00	278.45	4*	336.01	323.79	4*	238.20	233.20
5*	236.83	236.33	5	273.80	288.51	5*	200.80	199.52
6*	222.00	221.00	6	254.80	254.80	6*	187.47	187.20
7	176.50	176.50	7*	226.20	203.80	7	149.20	149.20
8*	338.96	319.43	8	387.53	397.75	8	300.54	323.87
9*	369.18	301.72	9*	420.64	339.05	9*	316.80	263.92
10*	178.38	170.55	10*	196.44	195.11	10*	158.18	146.12
11*	289.04	232.65	11*	334.13	253.09	11*	245.38	211.60
12	711.48	727.65	12*	824.17	809.93	12*	605.86	588.82
13*	469.50	460.99	13	519.92	521.05	13	388.80	397.67
14	457.04	512.11	14	524.56	565.94	14	409.02	410.53
15*	606.81	591.41	15*	699.49	643.72	15	552.79	574.95
16	4,432.21	4,490.17	16	4,902.34	6,328.07	16	3,580.32	4,297.47
17	1,310.83	1,427.70	17	1,632.86	1,659.46	17	1,062.03	1,146.68
18*	1,800.49	1,768.17	18	1,992.32	2,183.13	18	1,448.36	1,549.09
19	3,941.08	4,278.67	19	4,614.13	7,978.00	19*	3,865.54	2,712.08
20	2,225.66	2,397.61	20	2,664.00	2,792.90	20	1,892.28	1,934.11
	(a)			(b)			(c)	

Table 2.7: The ICP value of plan sets in the DriverLog domain returned by the Sampling and Hybrid approaches with the distributions (a) uniform, (b) w02 and (c) w08. The problems where Hybrid returns plan sets with better quality than Sampling are marked with *.

Prob	Sampling	Hybrid	Prob	Sampling	Hybrid	Prob	Sampling	Hybrid
1	27.87	27.87	1	28.56	28.56	1*	28.50	27.85
2	39.22	39.22	2	41.12	41.12	2	38.26	38.26
3*	51.36	50.43	3*	54.44	52.82	3*	49.49	48.58
4	43.00	43.00	4	46.00	46.00	4*	40.87	40.00
5	80.36	81.01	5	82.93	84.45	5	75.96	78.99
6	99.40	111.11	6	102.58	110.98	6	94.79	98.40
7*	38.50	38.49	7*	40.53	40.40	7*	37.04	36.60
8*	59.08	58.41	8*	62.15	62.08	8*	55.89	54.67
9	95.29	103.85	9	100.59	106.00	9	87.93	95.05
10*	52.04	50.00	10	52.40	52.40	10*	47.86	47.60
11	101.43	107.66	11*	110.18	108.07	11	97.56	99.06
12	123.09	129.34	12*	144.67	135.80	12	124.58	128.01
13*	57.37	57.22	13*	60.83	60.72	13	54.66	54.66
14*	62.75	62.33	14*	70.32	69.87	14*	65.20	62.02
15	116.82	117.86	15	113.15	124.28	15	101.09	124.43
16*	50.77	49.36	16*	54.98	54.12	16*	47.04	46.35
17*	38.38	37.77	17*	42.86	41.50	17*	37.56	36.92
18*	88.28	85.55	18*	94.53	90.02	18*	76.73	75.29
19*	82.60	82.08	19*	94.21	89.28	19*	74.73	72.45
20*	137.13	133.47	20*	150.80	135.93	20*	122.43	120.31
(a)			(b)			(c)		

Table 2.8: The ICP value of plan sets in the Depots domain returned by the Sampling and Hybrid approaches with the distributions (a) uniform, (b) w02 and (c) w08. The problems where Hybrid returns plan sets with better quality than Sampling are marked with *.

be more stable across different domains with uniform distribution, but less with the triangular, in particular Sampling wins Hybrid in the DriverLog domain when the distribution is w02. Perhaps this is because with the triangular distributions, the chance that LPG planner (that is used in our Sampling approach) returns the same plans even with different trade-off values would increase, especially when the most probable value of makespan happens to be in a (wide) range of weights in which one single plan is optimal. This result agrees with the intuition that when the knowledge about user preferences is *almost* complete (i.e., the distribution of trade-off value is "peak"), then the Sampling approach with smaller number of generated weight values may be good enough (assuming that a good planner optimizing a complete value function is available).

Since the quality of a plan set depends on how the two features makespan and plan cost are optimized, and how the plans "span" the space of time and cost, we also compare the Sampling and Hybrid approaches in terms of those two criteria. In particular, we compare plan sets returned by the two approaches in terms of (i) their *median* values of makespan and cost, which represent how "close" the plan sets are to the origin of the space of makespan and cost, and (ii) their *standard deviation* of makespan and cost values, which indicate how the sets span each feature axis.

Table 2.9 summarizes for each domain, distribution and feature the number of problems in which each approach (either Sampling or Hybrid) generates plan sets with better median of each feature value (makespan and plan cost) than the other. There are 60 problems across 3 different distributions, so in total, 180 cases for each feature. Sampling and Hybrid return plan sets with better makespan in 40 and 62 cases, and with better plan cost in 52 and 51 cases (respectively), which indicates that Hybrid is slightly better than Sampling on optimizing makespan but is possibly worse on optimizing plan cost. In ZenoTravel domain, for all distributions Hybrid likely returns better plan sets on the makespan than Sampling, and Sampling is better on the plan cost feature. In the DriverLog domain, Sampling is

		Median of makespan		Median of cost	
Domain	**Distribution**	$S > H$	$H > S$	$S > H$	$H > S$
ZenoTravel	uniform	3	**17**	**16**	4
	w02	6	**12**	**14**	4
	w08	6	**13**	**13**	6
DriverLog	uniform	6	**11**	7	**11**
	w02	**10**	8	8	**10**
	w08	**10**	7	9	9
Depots	uniform	**9**	8	**9**	7
	w02	7	**9**	5	**9**
	w08	**11**	7	7	**11**

Table 2.9: Number of problems for each domain, distribution and feature where Sampling (Hybrid) returns plan sets with better (i.e., smaller) *median* of feature value than that of Hybrid (Sampling), denoted in the table by $S > H$ ($H > S$, respectively). We mark bold the numbers of problems that indicate the outperformance of the corresponding approach.

better on the makespan feature with both non-uniform distributions, but worse than Hybrid with the uniform. On the plan cost feature, Hybrid returns plan sets with better median than Sampling on the uniform and w02 distributions, and both approaches perform equally well with the w08 distribution. In the Depots domain, Sampling is better than Hybrid on both features with the uniform distribution, and only better than Hybrid on the makespan with the distribution w08.

In terms of spanning plan sets, Hybrid performs much better than Sampling on both features across three domains, as shown in Table 2.10. In particular, over 360 cases for both makespan and plan cost features, there are only 10 cases where Sampling produces plan sets with better standard deviation than Hybrid on each feature. Hybrid, on the other hand, generates plan sets with better standard deviation on makespan in 91 cases, and in 85

		SD of makespan		SD of cost	
Domain	**Distribution**	$S > H$	$H > S$	$S > H$	$H > S$
ZenoTravel	uniform	8	**12**	6	**14**
	w02	4	**14**	7	**11**
	w08	6	**13**	8	**11**
DriverLog	uniform	5	**11**	6	**10**
	w02	7	**10**	7	**9**
	w08	8	**9**	**10**	7
Depots	uniform	**10**	7	7	**9**
	w02	7	**9**	5	**10**
	w08	5	**13**	7	**11**

Table 2.10: Number of problems for each domain, distribution and feature where Sampling (Hybrid) returns plan sets with better (i.e., larger) *standard deviation* of feature value than that of Hybrid (Sampling), denoted in the table by $S > H$ ($H > S$, respectively). We mark bold the numbers of problems that indicate the outperformance of the corresponding approach.

cases on the plan cost.

These experimental results support our arguments in Section 2.6.1 about the limits of sampling idea. Since one single plan could be optimal for a wide range of weight values, the search in the Sampling approach with different trade-off values may focus on looking for plans only at the same region of the feature space (specified by the particular value of the weight), which can reduce the chance of having plans with better value on some particular feature. On the opposite side, the Hybrid approach tends to be better in spanning plan sets to a larger region of the space, as the set of plans that have been found is taken into account during the search.

Contribution to the lower convex hull: The comparison above between Sampling and Hy-

brid considers the two features separately. We now examine the relation between plan sets returned by those approaches on the joint space of both features, in particular taking into account the the dominance relation between plans in the two sets. In other words, we compare the relative total number of plans in the lower convex-hull (LCH) found by each approach. Given that this is the set that should be returned to the user (to select one from), the higher number tends to give her a better expected utility value. To measure the relative performance of both approaches with respect to this criterion, we first create a set S combining the plans returned by them. We then compute the set $S_{lch} \subseteq S$ of plans in the lower convex hull among all plans in S. Finally, we measure the percentages of plans in S_{lch} that are actually returned by each of our tested approaches. Figures 2.14, 2.15 and 2.16 show the contribution to the LCH of plan sets returned by Sampling and Hybrid in the ZenoTravel, DriverLog and Depots domains.

In general, we observe that the plan set returned by Hybrid contributes more into the LCH than that of Sampling for most of the problems (except for some large problems) with most of the distributions and domains. Specifically, in the ZenoTravel domain, Hybrid contributes more plans to the LCH than Sampling in 15/20, 13/20 (and another 2 equals), 13/20 (another 2 equals) problems for the uniform, w02 and w08 distributions respectively. In the DriverLog domain, it is better than Sampling in 10/20 (another 6 equals), 10/20 (another 4 equals), 8/20 (another 5 equals) problems; and Hybrid is better in 11/20 (another 6 equals), 11/20 (another 4 equals) and 11/20 (another 4 equals) for the uniform, w02 and w08 distributions in the Depots domain. Again, similar to the ICP value, the Hybrid approach is less effective on problems with large size (except with the w08 distribution in the Depots domain) in which the searching time is mostly used for finding initial plan sets. We also note that a plan set with higher contribution to the LCH is *not* guaranteed to have better quality, except for the extreme case where one plan set contributes 100% and completely dominates the other which contributes 0% to the LCH. For example, consider problem 14

in the ZenoTravel domain: even though the plan sets returned by Hybrid contribute more than those of Sampling in all three distributions, it is only the w08 where it has a better ICP value. The reason for this is that the ICP value depends also on the range of the trade-off value (and its density) for which a plan in the LCH is optimal, whereas the LCH is constructed by simply comparing plans in terms of their makespan and cost separately (i.e., using the dominance relation), ignoring their relative importance.

The sensitivity of plan sets to the distributions: All analysis having been done so far is to compare the effectiveness of approaches with respect to a particular distribution of the trade-off value. In this part, we examine how sensitive the plan sets are with respect to different distributions.

Optimizing high-priority feature: We first consider how plan sets are optimized on each feature (makespan and plan cost) by each approach with respect to two non-uniform distributions w02 and w08. Those are the distributions representing scenarios where the users have different priority on the features, and plan sets should be biased to optimizing the feature that has higher priority (i.e., larger value of weight). In particular, plans generated using the w08 distribution should have better (i.e., *smaller*) makespan values than those found with the w02 distribution (since in the makespan has higher priority in w08 than it is in w02); on the other hand, plan set returned with w02 should have better values of plan cost than those with w08.

Table 2.11 summarizes for each domain, approach and feature, the number of problems in which plan sets returned with one distribution (either w02 or w08) have better *median* value than with the other. We observe that for both features, the Sampling approach is very likely to "push" plan sets to regions of the space of makespan and cost with better value of more interested feature. On the other hand, the Hybrid approach tends to be more sensitive to the distributions on both the features in the ZenoTravel domain, and is more sensitive only on the makespan feature in the DriverLog and Depots domains. Those results

		Median of makespan		Median of cost	
Approach	**Domain**	$w02 > w08$	$w08 > w02$	$w02 > w08$	$w08 > w02$
Sampling	ZenoTravel	**5**	**13**	**11**	**8**
	DriverLog	**6**	**10**	**13**	**5**
	Depots	**6**	**12**	**10**	**7**
Hybrid	ZenoTravel	**5**	**10**	**10**	**4**
	DriverLog	**4**	**10**	6	9
	Depots	**8**	**10**	4	11

Table 2.11: Number of problems for each approach, domain and feature where the plan sets returned with the w02 (w08) distribution with better (i.e., smaller) *median* of feature value than that with w08 (w02), denoted in the table by $w02 > w08$ ($w08 > w02$, respectively). For each approach, we mark bold the numbers for domains in which there are more problems whose plan sets returned with w08 (w02) have better makespan (plan cost) median than those with w02 (w08, respectively).

generally show that our approaches can bias the search towards optimizing features that are more desired by the user.

Spanning plan sets on individual features: Next, we examine how plan sets span each feature, depending on the degree of incompleteness of the distributions. Specifically, we compare the *standard deviation* of plan sets returned using the uniform distribution with those generated using the w02 and w08 distributions. Intuitively, we expect that the plan sets returned with the uniform distribution will have higher standard deviation than those with the distributions w02 and w08.

Table 2.12 shows for each approach, domain and feature, the number of problems generated with the uniform distribution that have better standard deviation on the feature than those found with the distribution w02. We observe that with the makespan feature, both ap-

		SD of makespan		SD of cost	
Approach	**Domain**	$U > w02$	$w02 > U$	$U > w02$	$w02 > U$
Sampling	ZenoTravel	9	10	**10**	**7**
	DriverLog	6	8	7	8
	Depots	**9**	**6**	**8**	**7**
Hybrid	ZenoTravel	9	10	**12**	**7**
	DriverLog	6	9	**8**	**7**
	Depots	**8**	**6**	**9**	**4**

Table 2.12: Number of problems for each approach, domain and feature where the plan sets returned with the uniform (w02) distribution have better (i.e., higher) *standard deviation* of the feature value than that with w02 (uniform), denoted in the table by $U > w02$ ($w02 > U$, respectively). For each approach and feature, we mark bold the numbers for domains in which there are more problems whose plan sets returned with the uniform distribution have better standard deviation value of the feature than those with the w02 distribution.

proaches return plan sets that are more "spanned" on makespan in the Depots domain, but not with ZenoTravel and DriverLog. With the plan cost feature, Hybrid shows its positive impact on all three domains, whereas Sampling shows it with the ZenoTravel and Depots domains. Similarly, table 2.13 shows the results comparing the uniform and w08 distributions. This time, Sampling returns plan sets with better standard deviation on both features in the ZenoTravel and Depots domains, but not in DriverLog. Hybrid also shows this in the ZenoTravel domain, but for the remaining two domains, it tends to return plan sets with expected standard deviation on the plan cost feature only. From all of these results, we observe that with the uniform distribution, both approaches likely generate plan sets that span better than with non-uniform distributions, especially on the plan cost feature.

In summary, the experimental results in this section support the following hypotheses:

		SD of makespan		SD of cost	
Approach	**Domain**	$U > w08$	$w08 > U$	$U > w08$	$w08 > U$
Sampling	ZenoTravel	**11**	**8**	**15**	**4**
	DriverLog	5	10	5	9
	Depots	**12**	**7**	**12**	**6**
Hybrid	ZenoTravel	**10**	**9**	**15**	**4**
	DriverLog	7	7	**8**	**6**
	Depots	5	8	**11**	**4**

Table 2.13: Number of problems for each approach, domain and feature where the plan sets returned with the uniform (w08) distribution with better (i.e., higher) *standard deviation* of feature value than that with w08 (uniform), denoted in the table by $U > w08$ ($w08 > U$, respectively). For each approach and feature, we mark bold the numbers for domains in which there are more problems whose plan sets returned with the uniform distribution have better standard deviation value of the feature than those with the w08 distribution.

- Instead of ignoring the user preferences which are partially specified, one should take them into account while synthesizing plans, as plan sets returned would have better quality.
- In generating plan sets sequentially to cope with the partial user preferences, the Sampling approach that searches for plans separately and independently of the solution space tends to return worse quality plan sets than the Hybrid approach.
- The resulting plan sets returned by the Hybrid approach tend to be more sensitive to the user preferences than those found by the Sampling approach.

2.7 Discussion

To the best of our knowledge, this work is a first step in domain-independent planning with preferences when the user preferences are not completely specified, in the same spirit of *model-lite* planning (Kambhampati, 2007). Our "language" to represent the partial preference model assumes a *complete* set of attributes of interest and a parameterized value function with unknown parameter values. Although in our work the unknown values are restricted in a *continuous* range, they can also be represented by a set of possible *discrete* values. These two representations of parameters' incompleteness are also the ways imprecise parameters are modeled in bounded-parameter MDPs (Givan *et al.*, 2000) and MDPs with imprecise reward functions (Regan and Boutilier, 2009, 2010; Xu and Mannor, 2009). Boutilier et al. (2010) consider the preference elicitation problem with a more general framework where both the set of attributes and the utility function are incomplete.

Our current representation and plan synthesis approach do have some limitations:

- The representation of the underlying complete preference model in our setting, i.e., the convex combination of metric quantities, is a subset of the preference language defined by PDDL3 (Gerevini *et al.* (2009)), which has been commonly used to represent preferences in planning domains. In PDDL3, preferences are constraints on the state trajectory of plans with "penalty" values (or weights) of being violated, and a plan is more preferable if it has lower total penalty value. While one can model partially specified "penalty" for preferences in PDDL3 with a distribution over continuous range or set of discrete values, it is unclear how to represent incompleteness for other constructs of the language. Similarly, it is an interesting question on how incompleteness can be extended for conditional preferences (Boutilier *et al.*, 2004).

- Using a convex combination of attributes as a utility function in our setting assumes that the criteria of interest are *mutual preferential independence*: although each at-

tribute is important, it does not affect how the user trades off one other objectives to the other. This property may be violated, for instance when we want to extend this setting to include preference statements in PDDL3 as attributes of interest. In a travel domain, for example, a passenger might be more willing to accept a more expensive ticket for a non-stop flight if she has to fly at night (i.e., the weight on the importance of "cost" is smaller).

- Our current implementation ignores the fact that changing the scale on objectives (e.g. from "hours" to "minutes" in the makespan of plans) may change the bias of the distribution of the Pareto set of plans on the objective axis. In other words, the set may look more uniform on the objective space using one scale than it is with a different scale (Branke, 2008). Although the ICP value agrees with the set Pareto dominance relation regardless of the scaling mechanism used (Fowler *et al.*, 2005), this effect can introduce a wrong evaluation about the distribution of the entire Pareto set of plans in the objective space to a user observing the representative set of plans (which may be biased towards some region of an axis due to the scaling mechanism used).

- Given that IPF is a nonlinear function, it is a challenge to modify the Metric-LPG planner to efficiently search for a set of plans optimizing such a quality measure. We believe that the current modification of Metric-LPG used for our experiments can be improved by designing new specific heuristics that are more effective for optimizing the measure. In addition, as observed by Kim et al. (2006), the computation time for IPF measure increases roughly exponentially with the number of objectives, and thus it is also challenging as to how to effectively incorporate the measure into the search for planning problems with a high number of criteria.

2.8 Summary

In the first part of the dissertation, we consider planning problems with incomplete user preferences in two scenarios where the knowledge about preference is completely unknown or only part of it is given. We propose a general approach to this problem where a set of plans is presented to the user from which she can select. For each situation of the incompleteness, we define a different quality measure for plan sets and investigate approaches to generating plan sets with respect to the quality measure. In the first scenario when the user is known to have preferences over plans, but the details are completely unknown, we define the quality of plan sets as their diversity value, specified with syntactic features of plans (its action set, sequence of states, and set of causal links). We then consider generating diverse set of plans using two state-of-the-art planners, GP-CSP and LPG. The approaches we developed for supporting the generation of diverse plans in GP-CSP are broadly applicable to other planners based on bounded horizon compilation approaches for planning. Similarly, the techniques we developed for LPG, such as biasing the relaxed plan heuristics in terms of distance measures, could be applied to other heuristic planners. The experimental results with GP-CSP explicate the relative difficulty of enforcing the various distance measures, as well as the correlation among the individual distance measures (as assessed in terms of the sets of plans they find). The experiments with LPG demonstrate the potential of planning using heuristic local search in producing large sets of highly diverse plans.

When part of the user preferences is given, in particular the set of features that the user is interested in and the distribution of weights representing their relative importance, we propose the usage of *Integrated Preference Function*, and its special case *Integrated Convex Preference* function, to measure the quality of plan sets, and propose various heuristic approaches based on the Metric-LPG planner (Gerevini *et al.*, 2008) to find a good plan set with respect to this measure. We show empirically that taking partial knowledge of

user preferences into account does improve the quality of plan set returned to the users, and that our proposed approaches are sensitive to the degree of preference incompleteness, represented by the distribution.

Variables: $G_1, ..., G_4, P_1, ..., P_6$

Domains:

$$G_1 : \{A_1, \bot\}, G_2 : \{A_2, \bot\}, G_3 : \{A_3, \bot\}, G_4 : \{A_4, \bot\}$$
$$P_1 : \{A_5, \bot\}, P_2 : \{A_6, A_{11}, \bot\}, P_3 : \{A_7, \bot\}$$
$$P_4 : \{A_8, A_9, \bot\}, P_5 : \{A_{10}, \bot\}, P_6 : \{A_{10}, \bot\}$$

Constraints (Mutex):

$$P_1 = A_5 \implies P_4 \neq A_9$$
$$P_2 = A_6 \implies P_4 \neq A_8$$
$$P_2 = A_{11} \implies P_3 \neq A_7$$

Constraints (Activity):

$$G_1 = A_1 \implies P_1 \neq \bot \wedge P_2 \neq \bot \wedge P_3 \neq \bot$$
$$G_2 = A_2 \implies P_4 \neq \bot$$
$$G_3 = A_3 \implies P_5 \neq \bot$$
$$G_4 = A_4 \implies P_1 \neq \bot \wedge P_6 \neq \bot$$

Initial state: $G_1 \neq \bot \wedge G_2 \neq \bot \wedge G_3 \neq \bot \wedge G_4 \neq \bot$

Figure 2.7: The CSP encoding for the example planning graph.

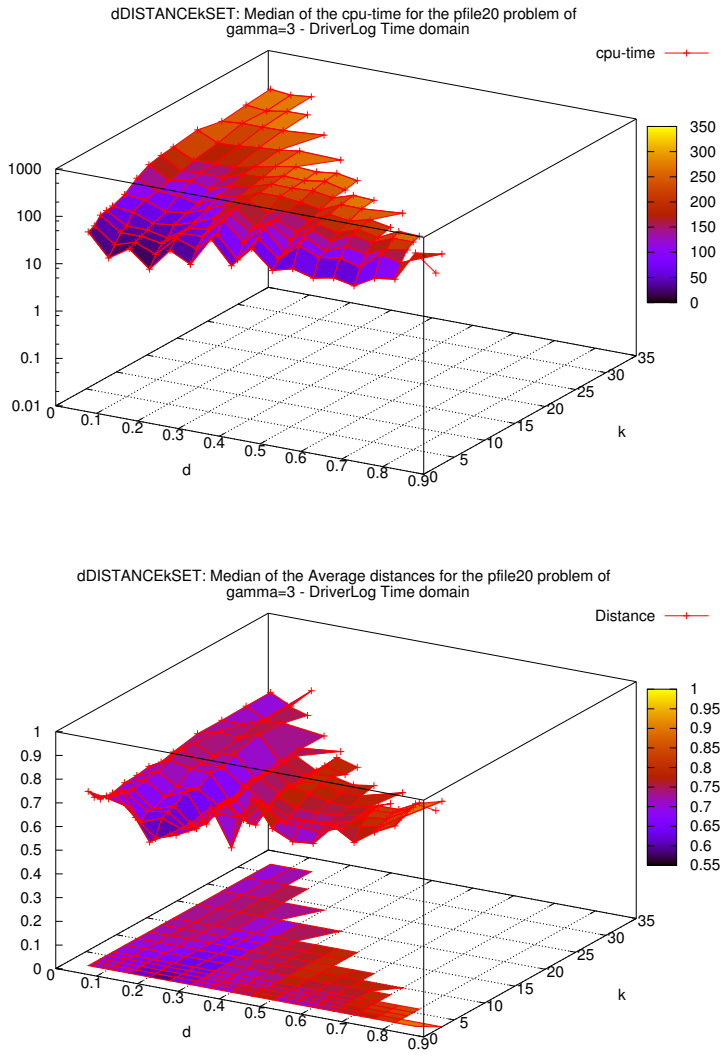


Figure 2.8: Performance of LPG-d (CPU-time and plan distance) for the problem pfile20 in the DriverLog-Time domain.

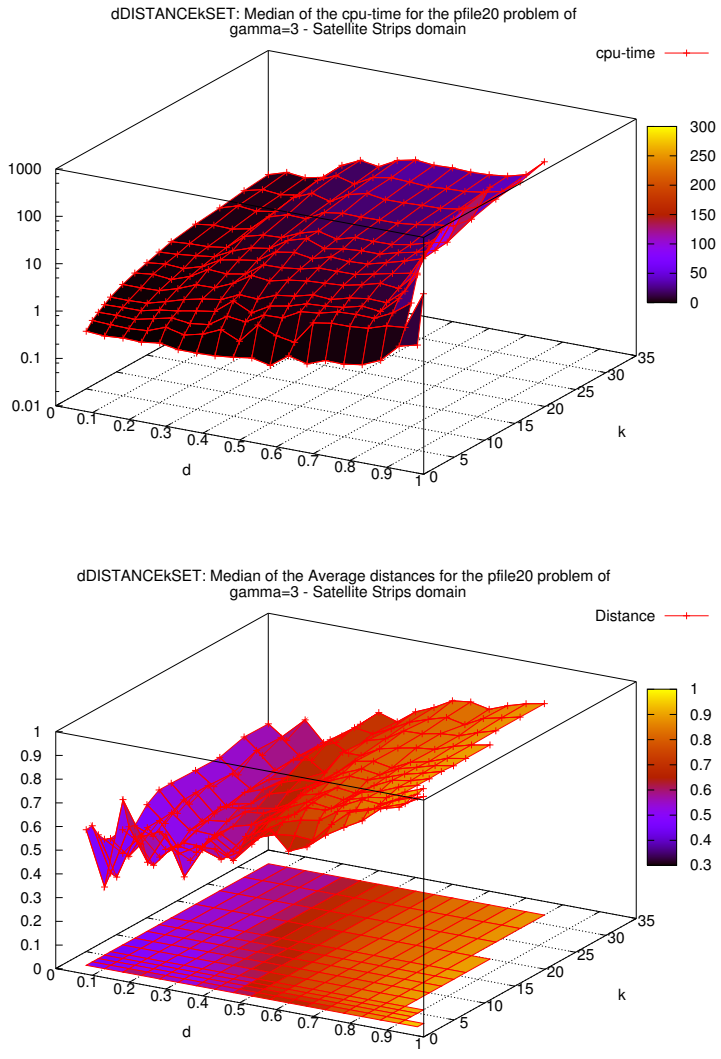


Figure 2.9: Performance of LPG-d (CPU-time and plan distance) for the problem pfile20 in the Satellite-Strips domain.

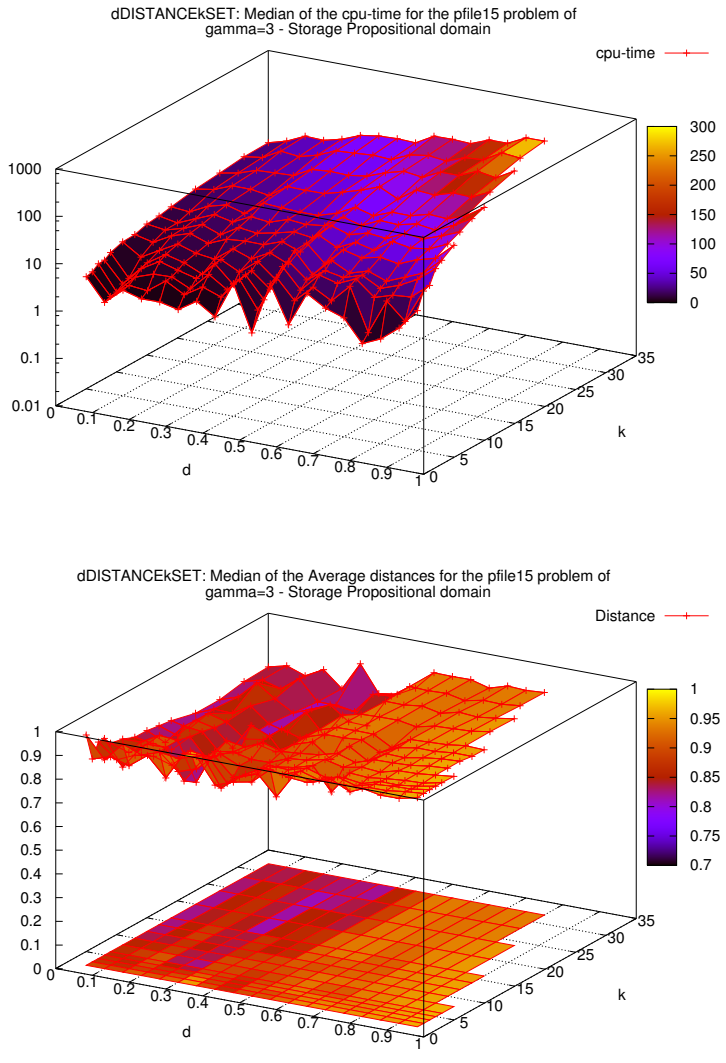


Figure 2.10: Performance of LPG-d (CPU-time and plan distance) for the problem pfile15 in the Storage-Propositional domain.

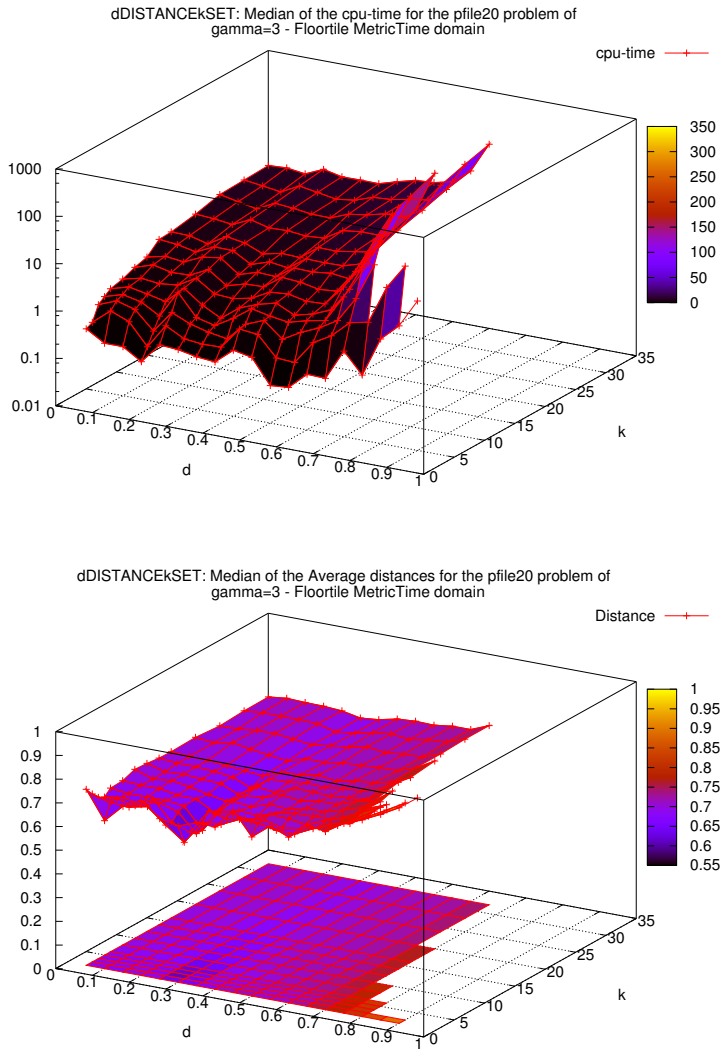


Figure 2.11: Performance of LPG-d (CPU-time and plan distance) for the problem pfile20 in the FloorTile-MetricTime domain.

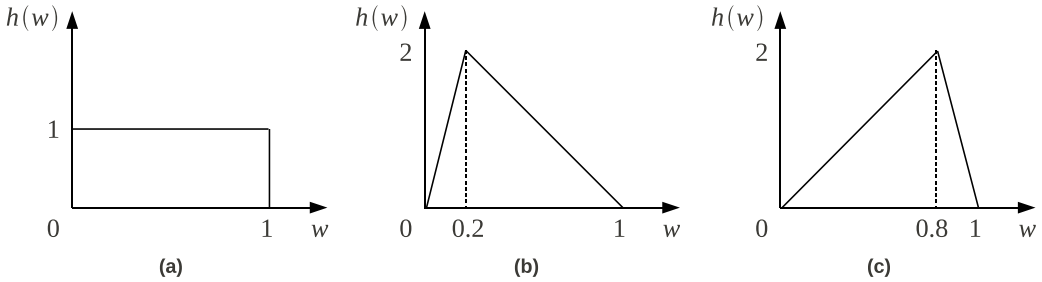


Figure 2.12: The distributions: (a) uniform, (b) w02, (c) w08 (see text).

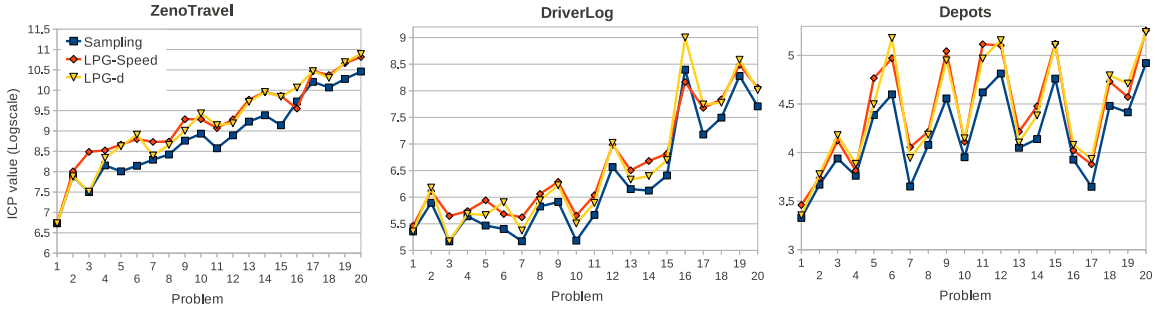


Figure 2.13: Results for the ZenoTravel, DriverLog and Depots domains comparing the Sampling and baseline LPG approaches on the overall ICP value (log scale) with the uniform distribution.

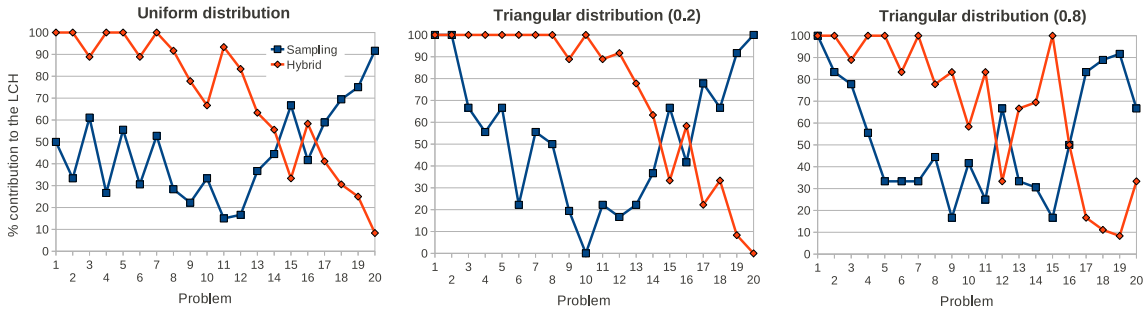


Figure 2.14: The contribution into the common lower convex hull of plan sets in the ZenoTravel domain with different distributions.

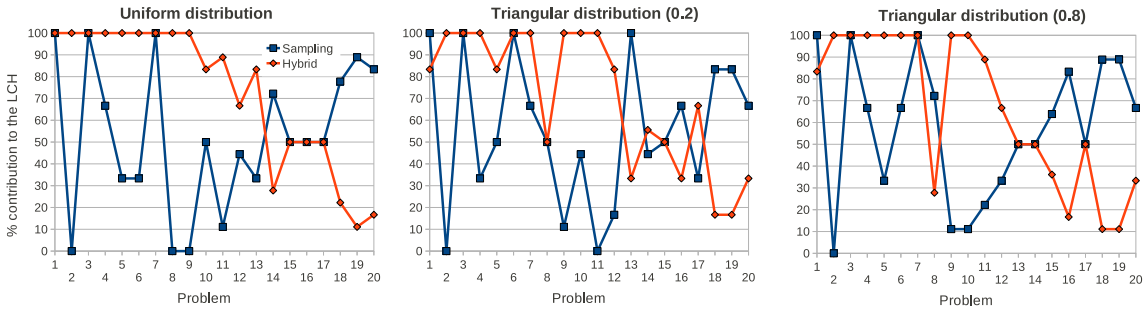


Figure 2.15: The contribution into the common lower convex hull of plan sets in the DriverLog domain with different distributions.

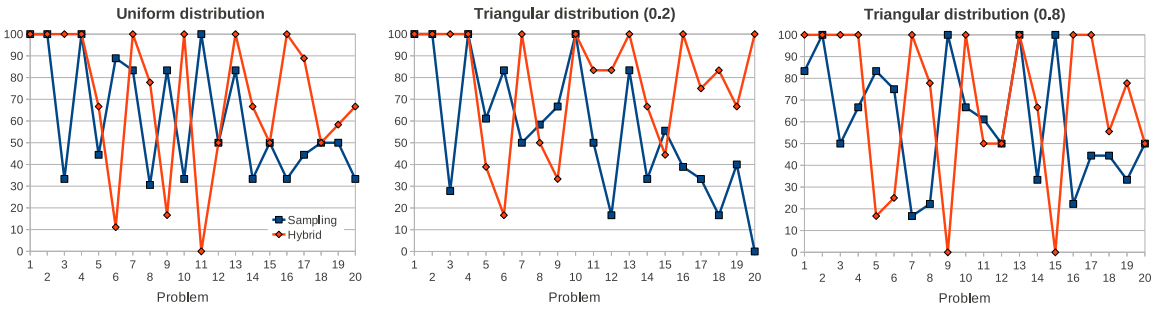


Figure 2.16: The contribution into the common lower convex hull of plan sets in the Depots domain with different distributions.

Chapter 3

PLANNING WITH INCOMPLETE DOMAIN MODELS

3.1 Introduction

The second part of the dissertation addresses the planning problems with partially specified domain models. The incompleteness 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. The users however often don't have the luxury of delaying their decision making. For example, although there exist efforts (Amir and Chang, 2008; Yang *et al.*, 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.

In this research, annotations on the incompleteness of the domains specify *possible* preconditions and effects of actions. 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 (Fikes and Nilsson, 1972) 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 (Domshlak and Hoffmann, 2007). 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 (Domshlak and Hoffmann, 2007). 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 (Weber and Bryce, 2011), in most of the tested domain both in terms of plan robustness and in planning time.

This chapter discusses the related work in Section 3.2. Section 3.3 formulates the planning problems with incomplete domain models. Section 3.4 formalized the robustness measure for plans under incomplete domain models. Section 3.5 presents a spectrum of problems given an incomplete domain model. Section 3.6 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 3.7 presents a compilation approach to synthesizing robust plans under the Generous Exeuction semantics. Section 3.8 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. This work is summarized in Section 3.9.

3.2 Related Work

As mentioned earlier, Garland & Lesh (2002) 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 & Bryce (2009) focuses on the plan generation in Garland & Lesh model, but their approach still relies on

the same unsatisfactory formulation of robustness. Weber and Bryce (2011) use the same formulation with (Nguyen *et al.*, 2010) 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.* (2013a; 2013b) consider 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 (2006b) also explores robustness of plans, but their focus is on temporal plans under unforeseen execution-time variations rather than on incompletely specified domains. 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 (Jensen *et al.*, 2004), 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.

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, 1973), 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, 1994). With similar incompleteness representation, Givan et al. (Givan *et al.*, 2000) considered bounded parameter Markov Desion 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. (2011) introduce factored MDPs with similar incompleteness representation and propose efficient dynamic programming exploiting their structures.

This work can also be categorized as one particular instance of the general model-lite planning problem, as defined in (Kambhampati, 2007), 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.

3.3 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 precoditions $Pre(o) \subseteq \mathcal{R}$, 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 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. 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 $\widetilde{\mathcal{D}}$ 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

[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.

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 $\langle\langle\widetilde{\mathcal{D}}\rangle\rangle$:

$$\gamma(\pi, s) = \bigcup_{\mathcal{D} \in \langle\langle\widetilde{\mathcal{D}}\rangle\rangle} \gamma^{\mathcal{D}}(\pi, s). \tag{3.1}$$

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:

$$\gamma^{\mathcal{D}}(\pi, s) = \begin{cases} s & \text{if } \pi = \langle\rangle; \\ \gamma^{\mathcal{D}}(\langle a\rangle, \gamma^{\mathcal{D}}(\pi', s)) & \text{if } \pi = \pi' \circ \langle a\rangle. \end{cases} \tag{3.2}$$

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 $\gamma^{\mathcal{D}}_{GE}(\langle a\rangle, s)$ is defined as follows:

$$\gamma^{\mathcal{D}}_{GE}(\langle a\rangle, s) = \begin{cases} (s \setminus Del^{\mathcal{D}}(a)) \cup Add^{\mathcal{D}}(a) & \text{if } Pre^{\mathcal{D}}(a) \subseteq s; \\ s & \text{otherwise.} \end{cases} \tag{3.3}$$

The *STRIPS execution* (SE) semantics follows the definition in STRIPS-style planning (Fikes and Nilsson, 1972), 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:

$$\gamma^{\mathcal{D}}_{SE}(\langle a \rangle, s) = \begin{cases} (s \setminus Del^{\mathcal{D}}(a)) \cup Add^{\mathcal{D}}(a) & \text{if } Pre^{\mathcal{D}}(a) \subseteq s; \\ s_\perp & \text{otherwise.} \end{cases} \tag{3.4}$$

The definition of $\gamma(\pi, s)$ in Eq. 3.1 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: Figure 3.1 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 domains (assuming all other action schemas in the domain are completely described).

```
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))
```

Figure 3.1: Decription of the incomplete operator *pick-up(?b - ball,?r - room)* in the *Gripper* domain.

3.4 A Robustness Measure for Plans

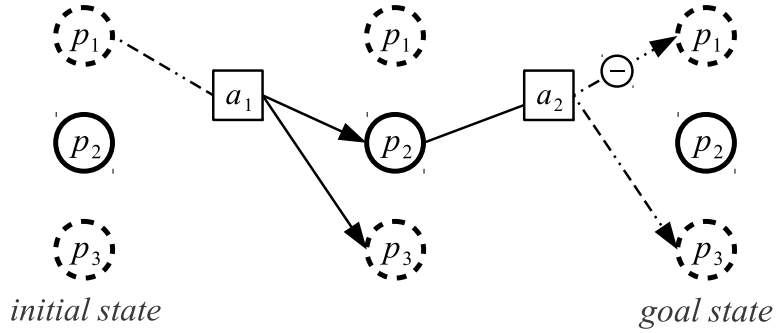
The robustness of a plan π for the problem $\widetilde{\mathcal{P}}$ is defined as the cumulative probability mass of the completions of $\widetilde{\mathcal{D}}$ under which π succeeds (in achieving the goals). More formally, let $Pr(\mathcal{D})$ be the modeler's estimate of the probability that a 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:

$$R(\pi) \stackrel{def}{\equiv} \sum_{\mathcal{D}\in\langle\langle\widetilde{\mathcal{D}}\rangle\rangle, \gamma^{\mathcal{D}}(\pi,I)\models G} Pr(\mathcal{D}) \tag{3.5}$$

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 $w_o^{pre}(r)$, $w_o^{add}(r)$, and $w_o^{del}(r)$ for all $o \in \mathcal{O}$ and its possible preconditions/effects r if r *is* realized as a precondition, add and delete effect of the operator in $\mathcal{D}$ (or the product of their "complement" $1 - w_o^{pre}(r)$, $1 - w_o^{add}(r)$, and $1 - w_o^{del}(r)$ 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: Figure 3.2 shows an example with an incomplete domain model $\widetilde{\mathcal{D}} = \langle F, A\rangle$



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

Figure 3.2: 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).

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 $\widetilde{\mathcal{P}} = \langle F, A, I = \{p_2\}, G = \{p_3\} \rangle$. The incomplete model is: $Pre(a_1) = \emptyset$, $\widetilde{Pre}(a_1) = \{p_1\}$, $Add(a_1) = \{p_2, p_3\}$, $\widetilde{Add}(a_1) = \emptyset$, $Del(a_1) = \emptyset$, $\widetilde{Del}(a_1) = \emptyset$; $Pre(a_2) = \{p_2\}$, $\widetilde{Pre}(a_2) = \emptyset$, $Add(a_2) = \emptyset$, $\widetilde{Add}(a_2) = \{p_3\}$, $Del(a_2) = \emptyset$, $\widetilde{Del}(a_2) = \{p_1\}$. 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 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 the uniform distribution. However, if the domain writer thinks that p_1 is very likely to be a precondition of a_1 and provides $w_{a_1}^{pre}(p_1) = 0.9$, 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 intutively, 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).

3.5 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 3.6 that RA with uniform

distribution of candidate complete models is complete for $\#P$ complexity class (Valiant, 1979a), 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.

3.6 Assessing Plan Robustness

Given an incomplete domain model $\widetilde{\mathcal{D}}$ and a 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 (c.f. Sang *et al.* (2005)). At a high level, we set up a set of logical constraints $\mathbf{\Sigma}_\pi$ 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.

3.6.0.1 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 ($1 \leq i \leq n+1$), we create a constraint:

$$a_i \Leftrightarrow exe(a_i), \tag{3.6}$$

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

$$exe(a_i) \equiv \bigwedge_{p \in Pre(a_i)} p_{i-1} \wedge \bigwedge_{p \in \widetilde{Pre}(a_i)} (p_{a_i}^{pre} \Rightarrow p_{i-1}). \tag{3.7}$$

We note that, slightly different from a state-based encoding for plan synthesis (c.f., Rintanen *et al.* (2004)), 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:

$$a_i \Rightarrow \bigwedge_{p \in Add(a_i)} p_i \wedge \bigwedge_{p \in Del(a_i)} \neg p_i \wedge \bigwedge_{p \in \widetilde{Add}(a_i)} (p_{a_i}^{add} \Rightarrow p_i) \wedge \bigwedge_{p \in \widetilde{Del}(a_i)} (p_{a_i}^{del} \Rightarrow \neg p_i). \quad (3.8)$$

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 ($1 \leq i \leq n$). For the changes from $\mathbf{F}$ to $\mathbf{T}$:

$$\neg p^{i-1} \wedge p^i \Rightarrow a_i \wedge \phi_i, \quad (3.9)$$

in which ϕ_i is defined as follows:

$$\phi_i = \begin{cases} \mathbf{T} & \text{if } p \in Add(a_i); \\ p_{a_i}^{add} & \text{if } p \in \widetilde{Add}(a_i); \\ \mathbf{F} & \text{otherwise.} \end{cases} \quad (3.10)$$

The constraints for changing from $\mathbf{T}$ to $\mathbf{F}$ are defined similarly.

Let $\mathbf{\Sigma}_\pi$ be the set of all constraints above. For each assigment 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 $\mathcal{M}(\mathbf{\Sigma}_\pi)$ be the set of all models satisfying $\mathbf{\Sigma}_\pi$.

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

Proof. Each model $\sigma \in \mathcal{M}(\mathbf{\Sigma}_\pi)$ 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.*, 2004)), 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 $\mathbf{\Sigma}_\pi$.

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

$$R(\pi) = 2^{|Z \setminus \widetilde{Z}|} \times \mathit{WMC}(\mathbf{\Sigma}_\pi),$$

where $\mathit{WMC}(\mathbf{\Sigma}_\pi)$ is the weighted model count of $\mathbf{\Sigma}_\pi$.

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:

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

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

3.6.0.2 SE Semantics

Variables: The variables are similar to the first set of variables in the correctness constraints under the GE semantics; we list them here again to make the discussion complete. They represent 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. Again, 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:

$$\bigvee_{C_p^i \leq k \leq i-1, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}. \tag{3.11}$$

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:

$$p_{a_m}^{del} \Rightarrow \bigvee_{m < k < i, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}, \tag{3.12}$$

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 $\mathbf{T}$ and protected.

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

$$p_{a_i}^{pre} \Rightarrow \bigvee_{C_p^i \leq k \leq i-1, p \in \widetilde{Add}(a_k)} p_{a_k}^{add}. \tag{3.13}$$

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

$$p_{a_i}^{pre} \Rightarrow \left(p_{a_m}^{del} \Rightarrow \bigvee_{m < k < i, p \in \widetilde{Add}(a_k)} p_{a_k}^{add} \right). \tag{3.14}$$

We again denote the set of constraints (3.11) - (3.14) 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 thus the robustness of the plan. We will therefore use $R(\pi)$ and WMC($\mathbf{\Sigma}_\pi$) interchangeably under the SE semantics.

3.6.1 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 (sketch). The following proof applies for both two semantics. 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, 1979b), 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 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 $R(\pi) = \frac{|\mathcal{M}(\mathbf{\Sigma})|}{2^n}$ can be established by verifying that an assignment σ satisfies $\mathbf{\Sigma}$ if and only if π succeeds under the corresponding complete model $\mathcal{D}_\sigma$. □

3.7 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 (2007). We focus

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

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 (2007). We briefly discuss how to adapt the compilation presented for the SE semantics.

3.7.1 Conformant Probabilistic Planning

Following the formalism used by Domshlak and Hoffmann (2007), a domain 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 $\mathcal{D}'$, 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 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.

3.7.2 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 3.6.0.1.) 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 $\mathcal{D}'$ 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.[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

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

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 our 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 $(\{s'_{i1}\}, ..., \{s'_{i(n+1)}\})$. With the note that $p \in s'_{i0}$ 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 $s'_{i(n+1)} \models G = G'$.

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 3.5. 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. Figure 3.3 shows both the original and the compiled specification of the action.

```
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)))))
```

Figure 3.3: An example of compiling the action *pick-up* in an incomplete domain model (top) into CPP domain (bottom). The hidden propositions $p^{pre}_{pick-up}$, $q^{add}_{pick-up}$ 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.

3.7.3 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 area. The source of incompleteness in this domain 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, 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.

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

[6]The *uncorrelated incompleteness* assumption applies for possible preconditions of action schemas 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.

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 3.1: The results of generating robust plans in Logistics domain (see text).

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.

Table 3.1 and 3.2 shows respectively the results in the Logistics and Satellite domains with $\rho \in \{0.1, 0.2, ..., 0.9\}$ and $m = \{1, 2, ..., 5\}$. The number of complete domain models

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	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Table 3.2: The results of generating robust plans in Satellite domain (see text).

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 $\rho = 0.6$—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 (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 is also a *valid plan*, as defined earlier, in the corresponding incomplete

domain. 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.

3.8 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 $\text{WMC}(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}}) > \delta$. (We note that all the correctness constraints for plans π, $\mathbf{\Sigma}_\pi$, plan prefixes π_k, $\mathbf{\Sigma}_{\pi_k}$ and relaxed plans $\widetilde{\pi}$, $\mathbf{\Sigma}_{\widetilde{\pi}}$ used in this approach for the SE semantics are presented in Section 3.6.0.2.)

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 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 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 $\mathbf{T}$ 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 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.

3.8.1 An Approximate Transition Function

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

$$\widetilde{\gamma}_{SE}(\langle a \rangle, s) = \begin{cases} (s \setminus Del(a)) \cup Add(a) \cup \widetilde{Add}(a) & \text{if } Pre(a) \subseteq s; \\ s_\perp & \text{otherwise.} \end{cases} \tag{3.15}$$

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 $\gamma^{\mathcal{D}}_{SE}(\pi, s) \neq s_\perp$, $\gamma^{\mathcal{D}}_{SE}(\pi, s) \subseteq \widetilde{\gamma}_{SE}(\pi, s)$.*

Proof. We prove the theorem by induction on the length of π. For the base case when

$\pi = \langle\rangle$, $\gamma^{\mathcal{D}}_{SE}(\langle\rangle, s) = \widetilde{\gamma}_{SE}(\langle\rangle, s) = s$ 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 $\gamma^{\mathcal{D}}_{SE}(\pi, s) \neq s_\perp$. 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 $\gamma^{\mathcal{D}}_{SE}(\pi, s) \neq s_\perp$. Let $s^{\mathcal{D}}_{\pi'} = \gamma^{\mathcal{D}}_{SE}(\pi', s)$, $\tilde{s}^{\mathcal{D}}_{\pi'} = \widetilde{\gamma}_{SE}(\pi', s)$. It must hold that $s^{\mathcal{D}}_{\pi'} \neq s_\perp$ and $Pre(a) \subseteq \gamma^{\mathcal{D}}_{SE}(\pi', s)$, because otherwise $\gamma^{\mathcal{D}}_{SE}(\pi, s) = s_\perp$. Thus,

$$\gamma^{\mathcal{D}}_{SE}(\pi, s) = (s^{\mathcal{D}}_{\pi'} \setminus Del^{\mathcal{D}}(a)) \cup Add^{\mathcal{D}}(a). \tag{3.16}$$

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

$$\widetilde{\gamma}_{SE}(\pi, s) = (\tilde{s}^{\mathcal{D}}_{\pi'} \setminus Del(a)) \cup Add(a) \cup \widetilde{Add}(a). \tag{3.17}$$

Since $s^{\mathcal{D}}_{\pi'} \subseteq \tilde{s}^{\mathcal{D}}_{\pi'}$, $Del^{\mathcal{D}}(a) \supseteq Del(a)$, $Add^{\mathcal{D}}(a) \subseteq Add(a) \cup \widetilde{Add}(a)$, from Eq. 3.16 and 3.17 we have $\gamma^{\mathcal{D}}_{SE}(\pi, s) \subseteq \widetilde{\gamma}_{SE}(\pi, s)$. 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 $\gamma^{\mathcal{D}}_{SE}(\pi, I) \models G$. From Proposition 1, $\gamma^{\mathcal{D}}_{SE}(\pi, I) \subseteq \widetilde{\gamma}_{SE}(\pi, I)$. 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.

3.8.2 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($\mathbf{\Sigma}$) of clause set $\mathbf{\Sigma}$, which can be computed in polynomial time.

Lower bound: We observe that the set of constraints $\mathbf{\Sigma}$, constructed as in (3.11)-(3.14), 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 $l_{\mathbf{\Sigma}} = \prod_{c_i} Pr(c_i)$ can be used as a lower bound for 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 6. *Given a set of monotone clauses* $\mathbf{\Sigma} = \{c_1, ..., c_k\}$, $l_{\mathbf{\Sigma}} = \prod_{c_i \in \mathbf{\Sigma}} Pr(c_i) \leq$ *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, 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 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: $u_{\mathbf{\Sigma}} = \prod_{\mathbf{\Sigma}^i} \min_{c \in \mathbf{\Sigma}^i} Pr(c)$.

3.8.3 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 $\widetilde{\pi}$.[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 $\mathbf{\Sigma}_p(t)$ and $\mathbf{\Sigma}_a(t)$.

- $L_1 \equiv s_{k+1}$, where $s_{k+1} = \widetilde{\gamma}_{SE}(\pi_k, I)$. The clause set $\mathbf{\Sigma}_p(1)$ for each $p \in L_1$, constructed with Constraints (3.11) and (3.12), 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

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

appear in L_t (i.e., $Pre(a) \subseteq L_t$), and a complete action, $\textit{noop}_p$, 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:

$$\mathbf{\Sigma}_a(t) = \bigwedge_{p \in Pre(a)} \mathbf{\Sigma}_p(t) \wedge \bigwedge_{q \in \widetilde{Pre}(a)} (q_a^{pre} \Rightarrow \mathbf{\Sigma}_q(t)). \tag{3.18}$$

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

$$\mathbf{\Sigma}_p(t+1) = \underset{\mathbf{\Sigma} \in S_1 \cup S_2}{\operatorname{argmax}}\, l(\mathbf{\Sigma} \wedge \mathbf{\Sigma}_{\pi_k}), \tag{3.19}$$

where $S_1 = \{\mathbf{\Sigma}_a(t) \mid p \in Add(a)\}$ and $S_2 = \{p_{a'}^{add} \wedge \mathbf{\Sigma}_{a'}(t) \mid p \in \widetilde{Add}(a')\}$.[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) $l(\mathbf{\Sigma}_G) > \delta$, 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.

[8]Note that the observation about converting constraints into monotone clauses, which faciliates our lower bound computation, still holds for constraints in (3.18) and (3.19).

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.[9] 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 3.11-3.14 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.

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 propostion p at layer t has a corresponding "best supporting action" at layer $t-1$, defined from Equation 3.19, 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 $\mathbf{\Sigma}_p(t)$ into defining the robustness of an incomplete relaxed

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

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 2 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* $s_{\to a_G}$ before a_G and the set of propositions $s^+_{\to a_G}$ 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 $C_p^{k+1} < k + 1$, 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 3, 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.[10] This procedure, presented in Algorithm 4, 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 2 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 $\mathcal{G}$

[10] 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.

Algorithm 2: Extracting robust relaxed plan.

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

(retrieved from Equation 3.19 and possibly a backward traversal over *noop* actions).

Line 16-18 handles a situation where the current relaxed plan $\widetilde{\pi}$ is actually incomplete—

Algorithm 3: 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 $\mathbf{\Sigma}_{\widetilde{\pi}}$ using Constraints (3.11)-(3.14).
    if $u(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}}) > \delta$ **then**
      Compute $R(\pi_k) = \text{WMC}(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}})$.
      if $R(\pi_k) > \delta$ **then**
        $r \leftarrow R(\pi_k)$; **return** *success*.
      end
    end
  end
  return *failure*
end

Algorithm 4: InitializeConstraintsAndQueue

Input: Goals G, relaxed planning graph $\mathcal{G}$.
Output: $\langle \mathbf{\Sigma}_{\widetilde{\pi}}, Q \rangle$
begin
  $\mathbf{\Sigma}_{\widetilde{\pi}} \leftarrow \mathbf{T}$.
  for $g \in G \setminus s^{+}_{\rightarrow a_G}$ **do**
    if $g \in s_{\rightarrow a_G}$ **then**
      $c \leftarrow$ Constraints (3.11)-(3.14) for g.
      $\mathbf{\Sigma}_{\widetilde{\pi}} \leftarrow \mathbf{\Sigma}_{\widetilde{\pi}} \wedge c$.
    end
    else
      $\mathbf{\Sigma}_{\widetilde{\pi}} \leftarrow \mathbf{\Sigma}_{\widetilde{\pi}} \wedge \mathbf{\Sigma}_g(T)$.
    end
    Push $\langle g, a_G, T \rangle$ into Q.
  end
  return $\langle \mathbf{\Sigma}_{\widetilde{\pi}}, Q \rangle$.
end

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),[11] 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).

3.8.4 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 re-

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

laxed 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 (Algorithm 5) 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 *et al.* (2007) 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.

Algorithm 5: Local Search with Restarts (Coles *et al.*, 2007)

```
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
```

3.8.5 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 *et al.* (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: prime implicant heuristics with size $k = 2$ and 2GB RAM as suggested, running it in the anytime mode.

Figure 3.4 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 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.[13] To demonstrate this, we first consider instances for which the

[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

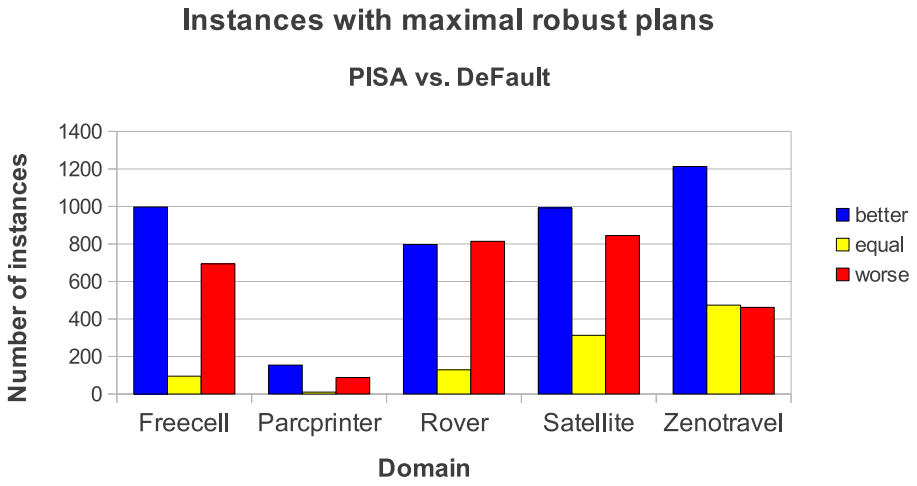
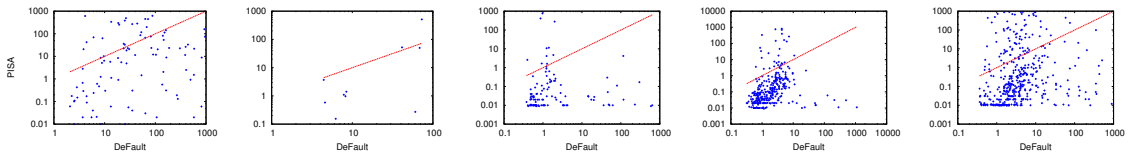


Figure 3.4: Number of instances for which PISA produces better, equal and worse robust plans than DeFault.

two approaches return best plans with the *same* robustness values. Figure 3.5 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 654.7x 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

within the time bound.



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

Figure 3.5: 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.

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 $\widetilde{\gamma}_{SE}(\pi, I) \supseteq G$) 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 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 $l(\mathbf{\Sigma}_{\pi_k} \wedge \mathbf{\Sigma}_{\widetilde{\pi}})$ 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, 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, c.f. (Wei and Selman, 2005; Gomes *et al.*, 2006), for the encoding for plan correctness under the GE semantics in Section 3.6.0.1.

3.9 Summary

This chapter addresses planning problems with incomplete domain models. The incompleteness of the domains is annotated with possible preconditions and effects of actions with the meaning that some of these will be the real ones. To quantify the quality of plans, a robustness measure is defined to capture the probability of success for plans during execution. We consider two semantics for the execution of plans, and propose two methods for generating robust plans under these semantics. The first approach which compiles the synthesis problem into conformant probabilistic planning problems can be used for both two semantics. The heuristic approach is proposed to directly use the robustness measure during search, utilizing specific properties of the correctness constraints for plans under

the STRIPS Execution semantics. The resulting planner is much more scalable than the compilation approach, and outperforms DeFault, which can handle incomplete STRIPS models.

Chapter 4

CONCLUSION AND FUTURE DIRECTIONS

4.1 Conclusion

This dissertation addresses planning problems in which either the user preferences or the domain models are incompletely specified. For the incomplete user preferences, we consider two scenarios: (1) when the user has preferences about plans but is completely unknown about her preferences, and (2) when the user can only partially express her preferences. For the incomplete domain models, we study planning situations when the domain is annotated with possible preconditions and effects (in addition to the known preconditions and effects).

For each of these above scenarios, the first issue is how to define the solution of the planning problems. For the incomplete user preferences, the planner's job is to present a set of plans to the user so that she can select one plan, and thus we propose several measures for evaluating plan sets with respect to the amount of knowledge about the user preferences. In particular, they are diversity measures for plan sets based on distance measures between two plans for the case of unknown preferences, and the Integrated Preference Function (IPF) (Carlyle *et al.*, 2003) in the situations of partially specified preferences. We investigate methods for generating plan sets with these measures. In particular, approaches based on GP-CSP (Do and Kambhampati, 2001) and LPG (Gerevini *et al.*, 2003) are discussed for diversity measure; and a spectrum of approaches for generating plan sets with the IPF measure is presented and implemented on top of Metric-LPG (Gerevini *et al.*, 2008) for partially specified user preferences.

The solution concept for planning problems with incomplete domain models is intro-

duced 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 considerred 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 (Weber and Bryce, 2011), in both plan quality and planning time.

4.2 Future Directions

The discussion in Section 2.7 outlines some limitations, challenges and also opportunities for further studies on planning with incomplete user preferences. We also presented in Section 3.5 a spectrum of planning problems given incomplete domain models; this work addresses two of them, the problems of assessing plan robustness and synthesizing robust plans. There are still several other interesting problems and directions for further research on planning with the presence of incomplete user preferences and/or domain models; in this section we will discuss in more detailed the situations when both the user preferences and domain models are incompletely specified.

In particular, we discuss an outlook on quality measures for plan sets for planning scenarios when both the user preferences and domain models are incomplete. The repre-

sentations of the two types of incompleteness follow the formulations used throughout this dissertation. In particular, we consider a planning problem with the user preferences and the domain model specified as follows.

- A user has preferences on solution plans of a problem, however she either does not know how to express her preferences ("unknown preferences"), or is able to specify only part of it ("partially specified preferences"). In the latter case, we consider a situation where the user is interested in minimizing values of a set of plan attributes such as plan makespan and execution cost, and their trade-off vector $\alpha \in \Lambda$ is unknown and modeled with a distribution function $h(\alpha)$. A specific vector α represents the preferences of a particular user in that she will compare plan using a value function $V(\pi, \alpha)$—assuming lower value is better. To make our discussion general, we do not assume a convex combination value function in this section.

- The domain model $\widetilde{\mathcal{D}}$ is incompletely specified with incomplete actions a having possible preconditions and effects $\widetilde{Pre}(a)$, $\widetilde{Add}(a)$, $\widetilde{Del}(a)$ (in addition to certain ones $Pre(a)$, $Add(a)$ and $Del(a)$). Given such an incomplete model, the robustness of a plan π for a planning problem $\widetilde{\mathcal{P}}$ is denoted by $R(\pi)$, which represents the probability that π succeeds to achieve all goals of $\widetilde{\mathcal{P}}$.

Recall that our research proposes *plan sets* as a solution concept for planning problems with incomplete user preferences. When the user preferences are completely unknown and the domain model is known to be incomplete, one can include plan robustness on top of the current formulation of diversity measures. It results in for example the notion of *d*DISTANT*k*SET for plans with *at least* a robustness value ρ, which can be chosen by the user.

We now discuss the case of partially specified preferences. Given a plan set $\Pi = \{\pi_1, ..., \pi_{|\Pi|}\}$, the user selects one plan π^* in the set based on a value function $V(\pi, \alpha)$

and her unknown trade-off value α^*. The value (or *penalty* in our "minimization" case) she pays will be $V(\pi^*, \alpha^*)$. As described in Section 2.4.2, the IPF measures the expected penalty the user will pay from a given plan set. When domain incompleteness comes into play, plan π^* may not be guaranteed to achieve the goals. We assume that when π^* turns out to fail during execution, a penalty $M > 0$ will be incurred. A "natural" question then is how the expected penalty of a plan set should be computed.

The presence of domain incompleteness suggests that, in addition to the random variable α, we also need to consider a second random variables e_π for each plan $\pi \in \Pi$. The variables e_π take value 1 for the case that π succeeds with probability $Pr(e_\pi = 1) = R(\pi)$, and 0 for its failure during execution with probability $Pr(e_\pi = 0) = 1 - R(\pi)$. Note that with the assumption that the two sources of incompleteness, user preferences and domain models, are irrelevant, the two random variables α and e_π are independent.

Using the notations in Section 2.4.2, we denote $\Pi^* \subseteq \Pi$ as the set of plans that are non-dominated with some value of α, and π_α^{-1} as the non-empty range of α for which $\pi \in \Pi^*$ is the best plan. A plan $\pi \in \Pi^*$ will be selected by the user with a probability $\int_{\alpha \in \pi_\alpha^{-1}} h(\alpha)\, d\alpha$. Since the robustness value of π does not change after it is selected, for a given $\alpha \in \pi_\alpha^{-1}$, the user pays a penalty $V(\pi, \alpha)$ with a probability $R(\pi)$, and M with probability $1 - R(\pi)$; thus $V'(\pi, \alpha) = R(\pi)\, V(\pi, \alpha) + (1 - R(\pi))\, M$ in average (given α). The expected penalty she will pay when selecting plan $\pi \in \Pi^*$ therefore is equal to:

$$\int_{\alpha \in \pi_\alpha^{-1}} h(\alpha) V'(\pi, \alpha)\, d\alpha. \tag{4.1}$$

Taking plan robustness into account, our *robustness-sensitive* IPF measure for a planning problem with incomplete domain model $\widetilde{\mathcal{P}}$, denoted by IPF$(\Pi, \widetilde{\mathcal{P}})$, calculates the expected penalty that a user will pay by selecting one plan in Π. It can be computed as follows:

$$\begin{aligned}\text{IPF}(\Pi, \widetilde{\mathcal{P}}) &= \sum_{\pi \in \Pi^*} \int_{\alpha \in \pi_\alpha^{-1}} h(\alpha) V'(\pi, \alpha)\, d\alpha \\ &= \left[\sum_{\pi \in \Pi^*} R(\pi) \int_{\alpha \in \pi_\alpha^{-1}} h(\alpha)\, V(\pi, \alpha)\, d\alpha\right] + M\left[1 - \sum_{\pi \in \Pi^*} R(\pi) \int_{\alpha \in \pi_\alpha^{-1}} h(\alpha)\, d\alpha\right].\end{aligned}$$

The last equation gives an intuition on the two expected penalty quantities that the user will pay in two different situations: the first term is the expected penalty incurred when the chosen plan turns out to be successful, and the second term is the expected cost paid when it fails. Also note that the first equation means that $\text{IPF}(\Pi, \widetilde{\mathcal{P}})$ is in fact the IPF defined with new value function V', instead of V.

Having established a variant of IPF measure for planning problems with both incomplete preferences and domain models, one question arises is what interesting property that can be guaranteed by the measure. In the following, we will show that under certain conditions, similar to IPF, the robustness-sensitive IPF measure does not contradicts with the *set Pareto dominance relation* between plan sets, defined as follows.

Set Pareto Dominance (Carlyle *et al.*, 2003) *Let $\{A^p\}$ and $\{B^p\}$ be approximate sets generated by competing heuristics, and $\{A^p\} \cup \{B^p\} \equiv \{C\}$. Let $\{C^p\}$ denote the set of nondominated solutions from $\{C\}$. If $\{C^p\} \equiv \{A^p\}$ and $\{C^p\} \cap \{B^p\} \subset \{A^p\}$, then set $\{A^p\}$ is said to dominate set $\{B^p\}$ in set Pareto dominance sense and vice versa.*

Carlyle et al. show, in Theorem 1 (Carlyle *et al.*, 2003), that for *regular* function $V(\pi, \alpha)$ (i.e., if π_1 is dominated by π_2, then $V(\pi_1, \alpha) > V(\pi_2, \alpha)$ for any α), if solution set Π_1 is dominated by Π_2 in the set Pareto dominance sense, then the IPF value of Π_2 is less than or equal to that of Π_1. This monotonicity property is desirable for quality measures for solution sets (Zitzler *et al.*, 2008); there are currently only two set measures satisfying this property: IPF and Hypervolume. We will show that the robustness-sensitive IPF measure also has this property under a slightly stronger condition on V and a choice of M.

Theorem 7. *Given a regular function $V(\pi, \alpha)$ such that $0 < V(\pi, \alpha) < V_{max}$ (for all plans π and vector α), two plan sets Π_1 and Π_2 for problem $\widetilde{\mathcal{P}}$ such that Π_1 is dominated by Π_2 in the set Pareto dominance sense with respect to the plan robustness attribute*[1] *and those of the user's interest, then IPF$(\Pi_1, \widetilde{\mathcal{P}}) \geq$ IPF$(\Pi_2, \widetilde{\mathcal{P}})$ with $M = V_{max}$.*

Proof. Consider the value function $V'(\pi, \alpha) = R(\pi)\, V(\pi, \alpha) + (1 - R(\pi))\, V_{\max}$. Given two plans π_1, π_2 such that π_1 is dominated by π_2 and $R(\pi_1) < R(\pi_2)$, it can be shown that $V'(\pi_1, \alpha) > V'(\pi_2, \alpha)$. In other words, $V'(\pi, \alpha)$ is regular with respect to the plan robustness and attributes of the user's interest.

Now following the proof of Theorem 1 (Carlyle *et al.*, 2003) for regular function V', instead of V, we can prove that IPF$(\Pi_1, \widetilde{\mathcal{P}}) \geq$ IPF$(\Pi_2, \widetilde{\mathcal{P}})$. □

The the extended value function $V'(\pi, \alpha)$ is both intuitive (in the sense that it captures the expected penalty incurred in two scenarios: the plan succeeds or fails), and can be tuned so that the resulting IPF measure has the nice monotonicity property.

[1]Different from attributes of the user's interest, for plan robustness attribute, higher value is consider better.

REFERENCES

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

Baier, J. and S. McIlraith, "Planning with preferences", AI Magazine **29**, 4, 25 (2009).

Bienvenu, M., C. Fritz and S. McIlraith, "Planning with qualitative temporal preferences", in "Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR)", pp. 134–144 (2006).

Birkhoff, G., "Lattice Theory, volume XXV of American Mathematical Society Colloquium Publications", American Mathematical Society, New York, 2nd (revised) edition (1948).

Blum, A. and M. Furst, "Fast planning through planning graph analysis", Artificial intelligence **90**, 1-2, 281–300 (1997).

Boddy, M., J. Gohde, T. Haigh and S. Harp, "Course of action generation for cyber security using classical planning", in "International Conference on Automated Planning and Scheduling (ICAPS)", pp. 12–21 (2005).

Boutilier, C., "A POMDP formulation of preference elicitation problems", in "Proceedings of the National Conference on Artificial Intelligence (AAAI)", pp. 239–246 (2002).

Boutilier, C., R. Brafman, C. Domshlak, H. Hoos and D. Poole, "CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements", Journal of Artificial Intelligence Research **21**, 1, 135–191 (2004).

Boutilier, C., T. Dean and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage", Journal of Artificial Intelligence Research **11**, 1, 94 (1999).

Boutilier, C., K. Regan and P. Viappiani, "Simultaneous Elicitation of Preference Features and Utility", in "Proceedings of the National Conference on Artificial Intelligence (AAAI)", pp. 1160–1197 (2010).

Bozkurt, B., J. Fowler, E. Gel, B. Kim, M. Köksalan and J. Wallenius, "Quantitative comparison of approximate solution sets for multicriteria optimization problems with weighted tchebycheff preference function", Operations research **58**, 3, 650–659 (2010).

Brafman, R. and C. Domshlak, "Preference handling–An introductory tutorial", AI magazine **30**, 1, 58–86 (2009).

Branke, J., "Consideration of partial user preferences in evolutionary multiobjective optimization", Multiobjective Optimization pp. 157–178 (2008).

Bryce, D., W. Cushing and S. Kambhampati, "Model-lite planning: Diverse multi-option plans & dynamic objective functions", in "ICAPS 2007 Workshop on Planning and Plan Execution for Real World Systems", (2007).

Carlyle, W. M., J. W. Fowler and B. K. E. S. Gel, "Quantitative comparison of approximate solution sets for bi-criteria optimization problems", Decision Sciences **34**, 1 (2003).

Chafle, G., K. Dasgupta, A. Kumar, S. Mittal and B. Srivastava, "Adaptation inWeb Service Composition and Execution", in "International Conference on Web Services, 2006. ICWS'06", pp. 549–557 (2006).

Chajewska, U., D. Koller and R. Parr, "Making rational decisions using adaptive utility elicitation", in "Proceedings of the National Conference on Artificial Intelligence (AAAI)", pp. 363–369 (2000).

Coles, A., M. Fox and A. Smith, "A new local-search algorithm for forward-chaining planning.", in "ICAPS", pp. 89–96 (2007).

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

desJardins, M. and K. Wagstaff, "Dd-pref: A language for expressing preferences over sets", in "Proceedings of the National Conference on Artificial Intelligence", (2005).

Do, M. and S. Kambhampati, "Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP", Artificial Intelligence **132**, 2, 151–182 (2001).

Do, M. and S. Kambhampati, "Sapa: A multi-objective metric temporal planner", Journal of Artificial Intelligence Research **20**, 1, 155–194 (2003).

Domshlak, C. and J. Hoffmann, "Fast probabilistic planning through weighted model counting.", in "ICAPS", pp. 243–252 (2006).

Domshlak, C. and J. Hoffmann, "Probabilistic planning via heuristic forward search and weighted model counting", Journal of Artificial Intelligence Research **30**, 1, 565–620 (2007).

Fikes, R. E. and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving", Artificial intelligence **2**, 3, 189–208 (1972).

Fowler, J., B. Kim, W. Carlyle, E. Gel and S. Horng, "Evaluating solution sets of a posteriori solution techniques for bi-criteria combinatorial optimization problems", Journal of Scheduling **8**, 1, 75–96 (2005).

Fox, M., A. Gerevini, D. Long and I. Serina, "Plan stability: Replanning versus plan repair", in "Proc. ICAPS", pp. 212–221 (AAAI Press, 2006a).

Fox, M., R. Howey and D. Long, "Exploration of the robustness of plans", in "Proceedings of the National Conference on Artificial Intelligence", vol. 21, p. 834 (2006b).

Fox, M. and D. Long, "PDDL2. 1: An extension to PDDL for expressing temporal planning domains", Journal of Artificial Intelligence Research **20**, 1, 61–124 (2003).

Fritz, C. and S. McIlraith, "Decision-theoretic golog with qualitative preferences", in "Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning, Lake District, UK, June", (2006).

Garland, A. and N. Lesh, "Plan evaluation with incomplete action descriptions", in "Proceedings of the National Conference on Artificial Intelligence", pp. 461–467 (2002).

Gelain, M., M. Pini, F. Rossi, K. Venable and T. Walsh, "Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies", Artificial Intelligence **174**, 3-4, 270–294 (2010).

Gerevini, A., P. Haslum, D. Long, A. Saetti and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners", Artificial Intelligence **173**, 5-6, 619–668 (2009).

Gerevini, A., A. Saetti and I. Serina, "Planning through stochastic local search and temporal action graphs in LPG", Journal of Artificial Intelligence Research **20**, 1, 239–290 (2003).

Gerevini, A., A. Saetti and I. Serina, "An approach to efficient planning with numerical fluents and multi-criteria plan quality", Artificial Intelligence **172**, 8-9, 899–944 (2008).

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

Gomes, C. P., A. Sabharwal and B. Selman, "Model counting: A new strategy for obtaining good bounds", in "Proceedings of the National Conference on Artificial Intelligence", vol. 21, p. 54 (Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006).

Ha, V. and P. Haddawy, "A hybrid approach to reasoning with partially elicited preference models", in "Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence", pp. 263–270 (1999).

Hazen, G., "Partial information, dominance, and potential optimality in multiattribute utility theory", Operations research pp. 296–310 (1986).

Hebrard, E., B. Hnich, B. O Sullivan and T. Walsh, "Finding diverse and similar solutions in constraint programming", in "Proceedings of the National Conference on Artificial Intelligence (AAAI)", vol. 20, p. 372 (2005).

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

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

Jensen, R., M. Veloso and 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, pp. 235–344 (2004).

Kambhampati, S., "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, p. 1601 (2007).

Kautz, H. and B. Selman, "BLACKBOX: A new approach to the application of theorem proving to problem solving", in "AIPS98 Workshop on Planning as Combinatorial Search", vol. 58260 (Citeseer, 1998).

Kim, B., E. Gel, J. Fowler, W. Carlyle and J. Wallenius, "Evaluation of nondominated solution sets for k-objective optimization problems: An exact method and approximations", European journal of operational research **173**, 2, 565–582 (2006).

Linden, G., S. Hanks and N. Lesh, "Interactive assessment of user preference models: The automated travel assistant", Courses And Lectures-International Centre For Mechanical Sciences pp. 67–78 (1997).

McMahan, H., G. Gordon and A. Blum, "Planning in the presence of cost functions controlled by an adversary", in "Proceedings of the Twentieth International Conference on Machine Learning (ICML)", vol. 20, p. 536 (2003).

Memon, A., M. Pollack and M. Soffa, "Hierarchical GUI test case generation using automated planning", IEEE Transactions on Software Engineering **27**, 2, 144–155 (2001).

Myers, K., "Metatheoretic plan summarization and comparison", in "International Conference on Automated Planning and Scheduling (ICAPS-06)", (2006).

Myers, K. and T. Lee, "Generating qualitatively different plans through metatheoretic biases", in "Proceedings of the National Conference on Artificial Intelligence", pp. 570–576 (1999).

Nguyen, T., M. Do, A. Gerevini, I. Serina and S. Kambhampati, "Generating diverse plans to handle unknown and partially known user preferences", Artificial Intelligence URL `http://dx.doi.org/10.1016/j.artint.2012.05.005` (2012).

Nguyen, T., M. Do, S. Kambhampati and B. Srivastava, "Planning with partial preference models", in "Proceedings of the 21st International Jont Conference on Artifical Intelligence (IJCAI)", pp. 1772–1777 (Morgan Kaufmann Publishers Inc., 2009).

Nguyen, T. and S. Kambhampati, "A heuristic approach to planning with incomplete strips action models", in "ICAPS", (2014).

Nguyen, T. A., S. Kambhampati and M. Do, "Synthesizing robust plans under incomplete domain models", in "Advances in Neural Information Processing Systems", pp. 2472–2480 (2013).

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

Pnueli, A., "The temporal logic of programs", in "18th Annual Symposium on Foundations of Computer Science", pp. 46–57 (IEEE, 1977).

Refanidis, I. and I. Vlahavas, "Multiobjective heuristic state-space planning", Artificial Intelligence **145**, 1-2, 1–32 (2003).

Regan, K. and C. Boutilier, "Regret-based reward elicitation for markov decision processes", in "Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence", pp. 444–451 (AUAI Press, 2009).

Regan, K. and C. Boutilier, "Robust Policy Computation in Reward-uncertain MDPs using Nondominated Policies", in "Proceedings of the 24rd AAAI Conference on Artificial Intelligence", pp. 1127–1133 (2010).

Rintanen, J., K. Heljanko and I. Niemelä, "Parallel encodings of classical planning as satisfiability", Logics in Artificial Intelligence pp. 307–319 (2004).

Robertson, J. and D. Bryce, "Reachability heuristics for planning in incomplete domains", in "ICAPS09 Workshop on Heuristics for Domain Independent Planning", (2009).

Russell, S. and P. Norvig, *Artificial intelligence: a modern approach* (Prentice Hall, 2010).

Sang, T., P. Beame and H. Kautz, "Heuristics for fast exact model counting", in "Theory and Applications of Satisfiability Testing", pp. 226–240 (Springer, 2005).

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

Serina, I., "Kernel functions for case-based planning", Artificial Intelligence **174**, 16-17 (2010).

Smith, D., "Choosing objectives in over-subscription planning", in "Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling", pp. 393–401 (2004).

Son, T. and E. Pontelli, "Planning with preferences using logic programming", Theory and Practice of Logic Programming **6**, 05, 559–607 (2006).

Srivastava, B., S. Kambhampati, T. Nguyen, M. Do, A. Gerevini and I. Serina, "Domain-independent approaches for finding diverse plans", in "IJCAI", (2007).

Tate, A., J. Dalton and J. Levine, "Generation of multiple qualitatively different plan options", Technical Report - University of Edinburgh (1998).

Valiant, L., "The complexity of computing the permanent", Theoretical computer science **8**, 2, 189–201 (1979a).

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

Van Den Briel, M., R. Sanchez, M. Do and S. Kambhampati, "Effective approaches for partial satisfaction (over-subscription) planning", in "Proceedings of the National Conference on Artificial Intelligence", pp. 562–569 (2004).

Viappiani, P., B. Faltings and P. Pu, "Preference-based search using example-critiquing with suggestions", Journal of artificial intelligence Research **27**, 1, 465–503 (2006).

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

Wei, W. and B. Selman, "A new approach to model counting", in "Theory and Applications of Satisfiability Testing", pp. 96–97 (Springer, 2005).

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

Xu, H. and S. Mannor, "Parametric regret in uncertain markov decision processes", in "Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on", pp. 3606–3613 (IEEE, 2009).

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

Zhang, Y., J. Callan and T. Minka, "Novelty and redundancy detection in adaptive filtering", in "Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval", pp. 81–88 (ACM, 2002).

Zhuo, H. H., S. Kambhampati and T. Nguyen, "Model-lite case-based planning", in "Proceedings of the 27th AAAI Conference on Artificial Intelligence", (2013a).

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

Zitzler, E., J. Knowles and L. Thiele, "Quality assessment of pareto set approximations", Multiobjective Optimization pp. 373–404 (2008).