An Investigation of Topics in

Model-Lite Planning and Multi-Agent Planning

by

Sarath Sreedharan

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2016 by the
Graduate Supervisory Committee:

Subbarao Kambhampati, Co-Chair
Yu Zhang, Co-Chair
Heni Ben Amor

ARIZONA STATE UNIVERSITY

August 2016

ABSTRACT

Automated planning addresses the problem of generating a sequence of actions that enable a set of agents to achieve their goals.This work investigates two important topics from the field of automated planning, namely model-lite planning and multi-agent planning. For model-lite planning, I focus on a prominent model named Annotated PDDL and it's related application of robust planning. For this model, I try to identify a method of leveraging additional domain information (available in the form of successful plan traces). I use this information to refine the set of possible domains to generate more robust plans (as compared to the original planner) for any given problem. This method also provides us a way of overcoming one of the major drawbacks of the original approach, namely the need for a domain writer to explicitly identify the annotations.

For the second topic, the central question I ask is "*under what conditions are multiple agents actually needed to solve a given planning problem?*". To answer this question, the multi-agent planning (MAP) problem is classified into several sub-classes and I identify the conditions in each of these sub-classes that can lead to required cooperation (RC). I also identify certain sub-classes of multi-agent planning problems (named DVC-RC problems), where the problems can be simplified using a single virtual agent. This insight is later used to propose a new planner designed to solve problems from these subclasses. Evaluation of this new planner on all the current multi-agent planning benchmarks reveals that most current multi-agent planning benchmarks only belong to a small subset of possible classes of multi-agent planning problems.

DEDICATION

This work is dedicated to my family, my mentors, my friends and to coffee

ACKNOWLEDGMENTS

I would like to start by thanking both the Co-chairs of my thesis committee. I want to thank Dr. Subbarao Kambhampati for giving me the chance to work on these problems, for his guidance and for all the discussions we had, not just about the technical aspect of the work, but on how I can improve my writing and my presentation skills. I truly believe that I am a better researcher, thanks to those discussions. I would like to thank Dr. Yu Zhang for his immense patience, for the enormous amount of time he had spent discussing these works with me and for letting me be part of many of his exciting projects. I would also like to the thank Dr. Heni Ben Amor for agreeing to be part of my thesis committee, his valuable insights and for being so generous with his time to discuss various ideas with me. I would also like to thank Dr. Tuan Nguyen for taking time out of his busy schedule to help me whenever I had questions about PISA. Next, I would like to thank all the other members of Yochan, namely Lydia Manikonda, Tathagata Chakraborty, Sailik Sengupta , Anagha Kulkarni, Yantian Zha and Daniel D'Souza for all their support and all the enlightening discussions. I am also heavily indebted to my family and all my friends for their unconditional support without which I could have never finished this work. Finally, I would like to thank god for helping me finish this work and for creating coffee beans without which I could have never survived those long nights.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

OVERVIEW

Automated planning is quickly growing to become one of the most important branches of artificial intelligence. Automated planning as a field focuses on creating algorithms that are capable of identifying a series of steps or actions (usually called a plan) that can allow an agent to realize a pre-specified goal. Automated planning has shown great success especially in creating faster and more scalable planning algorithms, and have also been used in many real life applications like Knight et al. 2001. Two topics that have recently been getting a lot of attention within the automated planning community are the areas of multi-agent planning and model-lite planning. In this document, we look at each of these topics and solve an important problem related to each.

The first part of the document is related to **Model-lite planning**. In this section, we start by looking at model-lite planning models in general and why they are important. Then we focus on one of the most prominent model-lite planning model, namely Annotated PDDL (APDDL) model. For this model, we identify ways of incorporating model information from successful plan traces. We also see how we can use this method to eliminate one of the major drawbacks of the current approaches related to APDDL, namely the domain writer's responsibility to identify the annotations for the model.

The second part of the document focuses on **Multi-agent planning**. In this section, we identify problem characteristics which can lead to a given problem to require multiple agents to solve it. This section presents work that has already been

presented in Zhang, Sreedharan, and Kambhampati 2016, for which I was one of the co-authors. In this section, we will also look at a subset of multi-agent planning problems that can be simplified by compiling it to a single agent one. We also propose a planner that makes use of this compilation, and the evaluation of this planner on current multi-agent planning benchmarks shows that most of the current benchmarks in fact only belongs to a small subset of possible multi-agent planning problems.

Part I

Model-lite planning

Chapter 2

INTRODUCTION

In recent years, automated planning has seen major improvements in scalability and efficiency. Since the original IPC (International Planning Competition) held at 1998 the speed and efficiency of the planners have increased by leaps and bounds. Even with these great advances in planning technology, we are yet to see a wide usage of planning techniques in real world applications. One of the major hurdles to the wide adoption of planning technologies is their reliance on complete and correct planning models. A planning model is expected to capture both the agent's capabilities and the environment dynamics. It provides the information on what actions can be performed and what the result of each action execution would be. Most planners require and expect a complete and correct planning domain to be provided, for them to solve a problem in that domain. Even stochastic planners expect a complete list of all possible transitions and an accurate distribution over it is required for each action.

In most cases these models are created manually by a domain expert, who is well versed with the domain dynamics and is able to identify both required action and the factored representation of problem states. Even when such experts are available, creating a planning model for a new problem domain remains a time consuming and effort intensive process. Even when a pre-existing planning model is available for an action, a small change in the problem setting can render the model unusable. For example, when creating domains for Robotics application, a planning domain made for one robot may not be applicable for another even when solving the same problem.

Model-lite planning is a relatively new area within the automated planning com-

munity, which aims at developing novel planning and planning-related algorithms that do not depend on the availability of a complete and correct planning domain. Though there exist a number of earlier works with similar motivation, the first work to clearly motivate this as an important research direction was Kambhampati 2007. Kambhampati 2007 presents many areas including web service composition that do not require the availability of a complete domain. The work also analyzes various classes of models that can be designed to use varying degrees of incompleteness and identifies ways these models can be utilized for planning and plan related activities. Since the codification of this approach to planning, there have been many related works. Some of the prominent ones include PISA (Nguyen and Kambhampati 2014),CAP (Zhang, Sreedharan, and Kambhampati 2015), ML-CBP (Zhuo, Kambhampati, and Nguyen 2012) etc..

Even though there exist a plethora of model-lite planning models, in this work we will be focusing on a single one, namely annotated PDDL model (Nguyen and Kambhampati 2014 and Nguyen, Kambhampati, and Do 2013). In this thesis, we will focus on extending the current APDDL approaches by providing them the ability to incorporate additional domain information available through plan execution traces. This new extension not only provides a way of refining the model but also circumvent one of the main drawbacks of the system, namely the need for the domain expert to annotate the models. In an Annotated PDDL model, one expects a domain writer to come up with a concise and correct list of annotations to be used by the algorithms. This means that the domain writer should be capable of identifying a list of annotation that is guaranteed to contain the correct missing predicates and is concise enough for the algorithms to work with them efficiently. This could be an extremely hard task for most domain writers and by providing a method of

eliminating incorrect domains, we are able to relax this requirement by making use of all possible predicates as annotations. Our tests on the new extensions show that our new approach outperforms the original PISA method for a given annotated model, and even when compared to other model refinement methods like RIM Zhuo, Nguyen, and Kambhampati 2013. Our method is able to produce plans with higher robustness when less number of cases are available. w The rest of this part of the document is structured as follows, in chapter 2, we take a quick look at some of the background information required to understand this work. Next, we discuss the original annotated pddl model and in chapter 5 we discuss the extensions made to it as part of this thesis. Finally, we conclude this document with a discussion on possible applications of this approach and directions for future works.

Chapter 3

BACKGROUND INFORMATION

Automated planning refers to a model-based approach to identify a set of actions that can allow an agent to reach a goal state from the given initial state (Geffner and Bonet 2013). By definition planning depends on a predefined model to make decisions regarding the actions it needs to perform. The model is expected to capture both the agent capabilities and the environment dynamics. Geffner and Bonet 2013 defines a basic state model using the tuple $\langle S, I, G, A, f, c \rangle$, where the various components are

- S - Set of states
- I - Initial state
- G - Goal states
- A - Set of actions
- $f(a, s)$ - Deterministic transition function, which defines the effects of each action
- c(a|s) - Positive action cost

There are a number of variations on this basic model to capture additional properties and other constraints. In this work, we will be mainly focusing on *Classical planning* models, which makes use of this basic state model with the additional assumption that all action costs are independent of states (Geffner and Bonet 2013).

Given a model, the model needs to be captured by some predefined language, so these model definitions can be read by a planner, these languages are usually standardized to allow for compatibility of input among the planners. There exists a number of languages that has been proposed to capture classical planning models like

STRIPS, ADL etc.. But one of the most widely used planning languages for classical planning is called Planning Domain Definition Language or PDDL(Mcdermott et al. 1998).

Introduced in the late 90s, PDDL was created as part of the first International Planning Competition. It was conceived with a goal of creating a standard language to represent classical planning domains and problems. The original language was strongly influenced by many of the then popular planning languages like STRIPS and ADL(Wikipedia 2016) and even the latest versions of pddl still retain many important aspects of these earlier languages.

A planning task for pddl is usually captured using separate domain and problem file. While the domain file contains the various action descriptions, the problem files usually provides the specific problem details. A standard action description in pddl includes the precondition and the effects of the action, each represented by a set of first-order predicates. The precondition represents the set of conditions that needs to be satisfied for an action to be executable, while the effects represent the changes in the world the action would bring about. During planning, we aim to produce a sequence of actions that satisfy the given goal (i.e the combined effects of the plan leads to the goal), such that the preconditions of a given action is satisfied by the state generated by the previous action. Figure 1 gives us a sample action description from the IPC blocksworld pddl domain.

3.1 Learning Domain Models

In most cases, the domain file is specifically written by a domain writer in a pre-specified domain description language. But as we have discussed before the process of

```
(:action pick-up
 :parameters (?x - block)
 :precondition (and (holding ?x))
 :effect
 (and (not (holding ?x))
 (clear ?x)
 (handempty)
 (ontable ?x)))
```

Figure 1. The putdown action as presented in ipc blockworld domain

writing such a domain can be quite an effort intensive process for the domain writer. Most of the times, it may not even be possible to find domain writers with both domain knowledge and planning knowledge required to create the correct domain model.

One of the most obvious ways of getting around the requirement for the availability of pre-existing planning models would be to try to learn them. It has been shown that a complete planning domain can be learned from scratch by using information from various sources. One of the most commonly available sources of planning domain knowledge is plan execution traces. ARMS (Yang, Wu, and Jiang 2007) is a very popular domain learner that learns a domain from the given set of cases (a set of successful plans and their corresponding start and goal states). Though it's interesting to note that in such cases the required actions in the domain and the symbols (and sometimes even the actions) needed to represent the domain are already available to the system, and most of these systems only need to learn the precondition/effects, HTN methods etc... There are newer works like (Konidaris, Kaelbling, and Lozano-Perez 2015) and (Konidaris, Kaelbling, and Lozano-Perez 2014) that are trying to further relax this requirement, by assuming knowledge of only the underlying continuous low-level action model and no other high-level symbolic information. These systems

need to learn not only the action precondition and effects but also need to figure out the set of symbols to use to represent the planning states. But most of the times, these systems ends up learning symbols that lack any apparent semantic meaning, this would mean that people may have a hard time creating new problems in these learned domains.

One common attribute among all the previously listed works is the assumption that the system will have access to enough information to learn a complete and correct model. In most cases this is a very hard requirement to meet. Consider a system like ARMS that relies on planning traces to learn a domain, if the system is trying to learn a model for a relatively new problem, it is quite unlikely that we can provide the system a large set of successful cases.

This brings us to the topic of model-lite planning models and why they are important.

3.2 Model-lite Planning Models

As discussed above, it may not be possible (or in some cases even necessary) to learn a complete planning model. Model-lite planning models were proposed as an alternate strategy to deal with the lack of complete models. The topic of model-lite planning was first introduced to the AI research community through the paper (Kambhampati 2007), and a central question the paper raises is whether we really need a complete planning model to perform planning (provided we can accept some decrease in accuracy)?. The paper introduces a possible spectrum of planning models (illustrated in figure 2 as presented in the Tian, Zhuo, and Kambhampati 2015) and the possible planning related tasks they can be used for. The paper identifies

and investigates two classes of planning models at the two ends of this spectrum incompleteness, namely *Shallow* and *Approximate* Models and discusses the various ways these models can be utilized.

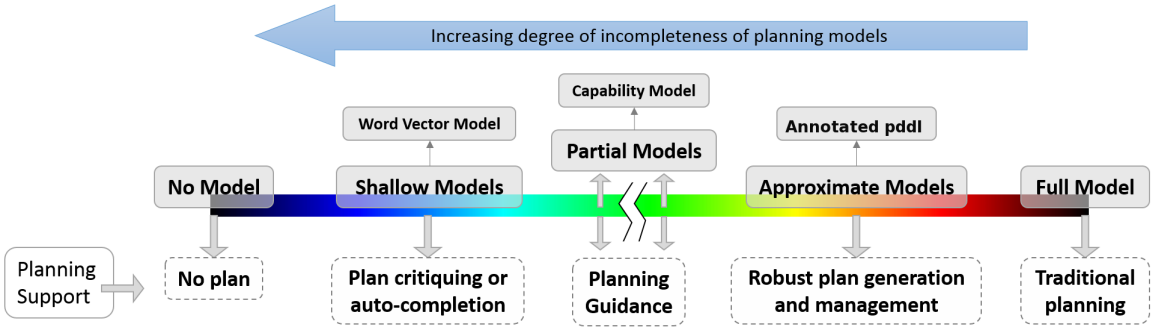

Figure 2. Models arranged on their increasing completeness

3.2.1 Shallow Models

Shallow models represent the more incomplete models within the spectrum. Shallow models do not contain enough information about the domain to perform planning, but can be successfully used for tasks like plan critique. Examples of shallow planning models include models that can only capture action affinity (the probability that certain actions can appear together) like word vector based models presented in Tian, Zhuo, and Kambhampati 2015.

3.2.2 Approximate Models

These planning models exist on the other end of the spectrum and are close to being complete. These models can support both plan creation as well as plan critiquing. These includes models that are incomplete in terms of some missing preconditions

and effects and models like annotated PDDL (which we will discuss in detail in the next chapter)

Another class of incomplete models that has been getting some recent attention are the partial models which lie closer to the center of the spectrum. while more incomplete than an approximate model, these types of planning models are still helpful in plan guidance and can be used for tasks like goal recognition and state prediction.

In the rest of this section of the thesis, we will mainly focus on Annotated PDDL and how we can use these models to generate more robust plans.

Chapter 4

ANNOTATED PDDL MODELS

In the previous chapter, we were introduced to the concept of incomplete model and why it is important to planning research. In this chapter, we take a closer look at annotated pddl model (Nguyen and Kambhampati 2014 and Nguyen, Kambhampati, and Do 2013), one of the earliest incomplete models to be proposed and see how we can improve it. Within the spectrum of incomplete models, we saw in the last chapter, annotated pddl models falls under Approximate models which are close to being complete. Annotated PDDL(APDDL) models were designed to handle cases where the incompleteness of the model arises from known unknowns, i.e cases where we have actions with missing preconditions and effects and are also aware of possible candidates to these missing predicates. In annotated pddl model, this information is represented in form annotations within action definitions. This means in APDDL the action definition also includes possible preconditions and possible effects in addition to the normal preconditions and effects, where each predicate could possibly be part of the real action definition (the unknown ground truth), Figure 3 presents an action definition from a sample APDDL model. One could see an APDDL model as a concise representation of a set of possible domains. Where each subset of the annotations represents a different domain and the true PDDL domain is present in this set of possible domains.

To see why such a model will be helpful, let us consider an example domain. Consider a simple gripper based domain where a robotic gripper needs to pick up boxes from an assembly line and place them on a pallet. A domain writer tasked

```
(:action pick-up
 :parameters (?x - block)
 :precondition (and )
 :poss-precondition (and (holding ?x))
 :effect
 (and (not (holding ?x))
 (ontable ?x))
:poss-effect
 (and
 (clear ?x)
 (handempty)
  )
)
```

Figure 3. The putdown action as presented in ipc blockworld domain

with the responsibility of writing an action model for this robotic gripper, may not be aware of all the details of the specific robotic arm to be used. The writer might believe that that arm could possibly have a limitation on the amount of weight it can lift. The fact whether this assumption is true or not may actually depend on the specific robot being used and other factors that may not be apparent to the domain writer. So it would make sense for the domain writer to keep this precondition as a possible precondition for the lift action of the robot.

4.1 Problem Formulation

We define an APDDL domain by the tuple $\widetilde{\mathcal{M}} = \langle \mathcal{R}, \mathcal{O} \rangle$ where $\mathcal{R}$ denotes the lifted predicates in the domain and $\mathcal{O}$ provides the list of operators. Each operator $o_i \in \mathcal{O}$, contains a list of known preconditions $pre(o_i)$, known add effects $add(o_i)$, known delete effects $del(o_i)$, possible preconditions $\widetilde{pre(o_i)}$, possible add effects $\widetilde{add(o_i)}$ and possible delete effects $\widetilde{del(o_i)}$. Each annotation r in the possible predicate list

may also be associate with a weight ($w_o^{pre}(r)$ or $w_o^{add}(r)$ or $w_o^{del}(r)$). But for the rest of this chapter we assume that all possible predicates have equal weights.

As discussed before an APDDL domain concisely represents a set of possible domains, where each unique "realization" of the annotations corresponds to a unique domain. A unique realization of the annotated model refers to a complete PDDL domain created by moving a certain subset of annotation to the actual precondition or effects list. We refer to the list of all possible realizations as completions list and is denoted as $\langle\langle\widetilde{\mathcal{M}}\rangle\rangle$. The size of completions list will be equal to 2^k, where k is the total number of all the annotations in the original APDDL model $\widetilde{\mathcal{M}}$. If we assume that the actual ground truth (the true unknown domain) is $\mathcal{M}^*$, then we know that $M^* \in \langle\langle\widetilde{\mathcal{M}}\rangle\rangle$.

For a given plan $\pi = \langle a_1, .., a_n\rangle$ and initial state s_0 and goal g, the validity of the plan for the model $\widetilde{\mathcal{M}}$ (if the plan can achieve the goal) is defined with respect to the completion list. For a given complete model $\mathcal{M}_i \in \langle\langle\widetilde{\mathcal{M}}\rangle\rangle$, the plan is said to be valid for the model $\mathcal{M}_i$ if

$$g \subseteq \gamma_{\mathcal{M}_i}(\pi, s_0) \tag{4.1}$$

Where $\gamma_{\mathcal{M}_i}$ provides the transition function for the model $\mathcal{M}_i$ and $\gamma_{\mathcal{M}_i}(\pi, s_0)$ is equal to $\gamma_{\mathcal{M}_i}(a_1, .., \gamma_{\mathcal{M}_i}(a_{n-1}, \gamma_{\mathcal{M}_i}(a_n, s_0)))$, where the application of transition function for a single action on a state s_i is given as

$$\gamma_{\mathcal{M}_i}(a_i, s_{i-1}) = \begin{cases} \phi, if pre(a_i) \not\subseteq s_{i-1} \\ (s_{i-1} \setminus del(a_i)) \cup add(a_i) \end{cases} \tag{4.2}$$

So if at any point of the plan, the preconditions of the action does not match the corresponding execution state, the plan would be considered invalid. This definition of plan validity, where the failure of a single action can lead to complete plan failure is generally referred to as STRIPS execution semantics in automated planning literature.

Another popular execution semantics that is considered by many planning algorithms is generous execution semantics. But for the remainder of this document we assume that the plans follow STRIPS execution semantics.

4.2 Robustness of the Plan

A given plan may only be executable in a fraction of the total number of domains in the completion list. Given the fact that the ground truth is completely unknown, higher the number of domains the plan is executable, more likely the plan is executable in the ground truth. Thus, assuming each possible complete domain is equally likely, the ratio of executable domains for a plan becomes a measure of *goodness* of the plan and is usually referred to as it's robustness. For a given plan π, if $\langle\langle\widetilde{\mathcal{M}}\rangle\rangle$ is the set of all possible domains for an APDDL model $\widetilde{\mathcal{M}}$ and $\langle\langle\widetilde{\mathcal{M}'}\rangle\rangle$ be the set of domains where *pi* is executable, then the robustness of the plan π is defined as

$$R(\pi) = \frac{|\langle\langle\widetilde{\mathcal{M}'}\rangle\rangle|}{|\langle\langle\widetilde{\mathcal{M}}\rangle\rangle|} \tag{4.3}$$

In general, the robustness of a plan can be defined as the cumulative probability of each possible domain where the plan is executable.

$$R(\pi) = \sum_{\mathcal{M}_i, \text{ where } G \subseteq \gamma_{\mathcal{M}_i}(\pi, I)} Pr(\mathcal{M}_i) \tag{4.4}$$

Thus, one of the main goals of using APDDL model becomes to generate the most robust plan for a given planning problem. In addition to generating robust plan, other important tasks related to APDDL (as listed in Nguyen, Kambhampati, and Do 2013) and robustness measures can be.

- Improving the robustness of a given plan

- Generating plans with the desired level of robustness
- Robust plan generation with fixed plan costs
- Assessing the robustness of a given plan

4.3 Assessing Robustness as a WMC

Most of the tasks listed above require assessing the plans, the most obvious way of measuring the robustness would be to iterate through all possible complete domains that can support the execution of the given plan. This could be an extremely time-consuming process, Nguyen and Kambhampati 2014 proposes a different approach, in which a set of SAT constraints for annotation required to support the execution of each plan is generated. Next, we use a weighted model counting solver to identify the realization of annotations that can support these constraints. A WMC solver should return the set of possible instantiations for the variables for which the constraints are satisfied. Since the variables in the SAT constraints corresponds to the annotations in the domain, each solution represents a possible domain in the completions list and the total number of solutions found by WMC corresponds to the number of domains where the plan is executable.

For a given plan $\pi =< a_1, .., a_n >$ with initial state s_I and goal g and a domain $\mathcal{M}$ that has k annotations. To create a set of WMC constraints to capture the robustness of the plan, we will need k boolean variables to capture every annotation in the model. For example, consider an annotation r in an operator $o_i \in \mathcal{O}$, let $r \in \widetilde{pre(o_i)}$, then we need to consider an equivalent boolean variable $r_{o_i}^{pre}$ and for every WMC solution with the value $r_{o_i}^{pre} = T$, the corresponding domain model will have $r \in pre(o_i)$.

Next, we need to create a set of constraints using these variables to capture the

validity of the given plan, for ease of creating the constraints we consider an updated plan $\pi' =< a_0, a_1, .., a_n, a_{n+1} >$, where Nguyen and Kambhampati 2014 presents two possible constraints that can capture the executability of a given plan, the constraints are namely

- Precondition establishment and protection
- Possible precondition establishment and protection

The first constraint deals with the fact that given a known precondition of an action, there should exist a previous action in the plan that establishes the required predicate, if it is not already true and this action should appear after all actions that could possibly delete this predicate. This means if the an action a_i at position i in the plan has a precondition $p_{a_i}^{pre}$, the constraint will be

$$\bigvee_{C_p^i \leq k < i, p \in \widetilde{add}(a_k)} p_{a_k}^{add} \tag{4.5}$$

Where C_p^i is the last level before i where the value p was confirmed. If there is any actions that can delete the predicate p at some level $m > C_p^i$

$$p_{a_m}^{del} \implies \bigvee_{m \leq k < i, p \in \widetilde{add}(a_k)} p_{a_k}^{add} \tag{4.6}$$

Similarly, the second set of constraints is used to protect the values of possible preconditions that may be realized.

4.4 Generating Robust Plans

The above section talked about how we can assess a given plan to identify the robustness. Now a natural question is to ask if it is possible to generate the most robust

plan for a given problem. Nguyen and Kambhampati 2014 introduces a stochastic search based method to perform robust plan generation. The search proceeds by trying to extract a plan with robustness value greater than the robustness threshold of that specific iteration (initially set to zero), at the end of each iteration the threshold is updated to the robustness of the last extracted plan. This means that the plan extraction process involves multiple evaluations of plan robustness and since WMC calculations can be expensive Nguyen and Kambhampati 2014 also proposes an approximation for exact WMC value.

- Lower limit on the Robustness value - If we are given a set of monotone clauses $\Sigma = \{c_1, .., c_k\}$, then $WMC \geq \Pi_{i=1..K} Pr(c_i)$. Where the probability is given by the weights of variables in the clauses (for eg: if $c_i = \vee_i x_i$, then $Pr(c_i) = 1 - \Pi_i(1 - w_i)$). The only catch being that the clauses we calculate need not be monotone, we get around this by calculating the inverted constraints for any negative constraints, for eg: if there is a constraint p_o^{del} for elimination a delete effect of an action o_i with a weight $w_o^{del}(p)$, then we can use a constraint p_o^{del} and weight $1 - w_o^{del}(p)$
- Upper limit on the Robustness value - We can use the minimum probability within the given set of monotone clauses as an upper bound

4.5 Current Drawbacks of Robust planning

In this chapters, we described a way of performing robust planning given an annotated pddl model. This method, unfortunately, is not without its drawbacks. One of the major drawbacks being, the need for the domain writer to come up with appropriate annotations for each incomplete action. In addition to the requirement

that the annotations should contain the missing predicates, the fact that the original PISA do not provide a way of refining a given domain model means that if the domain writer chooses too many annotations, it can conversely affect the planner's ability to create plans with high robustness value. In fact, one might argue that there may be cases when the domain writer is not even aware of the fact that a given action definition is incomplete.

Chapter 5

CASE-BASED PISA

In the previous chapter, we looked at APDDL an incomplete planning model which relied on annotations provided by the domain writer to generate robust plans. The approach as presented in the last chapter does not suggest any ways of refining an already defined domain model. In this chapter, we introduce an extension of the existing PISA planning framework to incorporate information from an additional source, namely a set of past successful plans. We assume a plan trace includes the initial state, goal state and the set of action sequences in the plan, there exist a plethora of works (like Yang, Wu, and Jiang 2007, Zhuo, Kambhampati, and Nguyen 2012, Zhuo, Nguyen, and Kambhampati 2013 and Zhang, Sreedharan, and Kambhampati 2015) that make use of successful plan traces to not only refine incomplete models but also learn complete models from scratch. The choice of successful plan traces as the additional source of information was also motivated by the fact that one could easily assume that these successful plans were in fact previously generated robust plans. In the rest of this document, we will refer to this approach as Case-Based PISA or C-PISA.

To see how the use of successful plan traces would help to improve robustness, consider an incomplete model $\widetilde{\mathcal{M}}$ with p number of possible preconditions and q number of possible effects. Let the completion list of the incomplete model be $\langle\langle\widetilde{\mathcal{M}}\rangle\rangle$, where $|\langle\langle\widetilde{\mathcal{M}}\rangle\rangle| = 2^{p+q}$. Let $\mathcal{M}^*$ be the ground truth with $\mathcal{M}^* \in \langle\langle\widetilde{\mathcal{M}}\rangle\rangle$. Now let us assume we are given a set of successful plan traces $\mathcal{T}$ that succeeded in $\mathcal{M}^*$ (we assume plan traces are noise free). Our goal would be to try and build a set of models $\langle\langle\widetilde{\mathcal{M}}_1\rangle\rangle$

such that $\langle\langle\widetilde{\mathcal{M}}_1\rangle\rangle \subset \langle\langle\widetilde{\mathcal{M}}\rangle\rangle$ and $\mathcal{M}^* \in \langle\langle\widetilde{\mathcal{M}}_1\rangle\rangle$, which can then be used to perform robust planning via PISA. $\langle\langle\widetilde{\mathcal{M}}_1\rangle\rangle$ is formed in effect by eliminating the domains where the given cases are not executable. Since we already know that all plans in $\mathcal{T}$ are executable in $\mathcal{M}^*$ and $\langle\langle\widetilde{\mathcal{M}}_1\rangle\rangle$ contains all domains where $\mathcal{T}$ is executable, we have $\mathcal{M}^* \in \langle\langle\widetilde{\mathcal{M}}_1\rangle\rangle$. By eliminating the known incorrect domains, the new robustness measure better approximates the plan's chance of success or failure. Following this logic, if we are given enough cases we can reduce the size of $\langle\langle\tilde{\mathcal{M}}_1\rangle\rangle$ to one and thereby obtaining the ground truth.[1]

5.1 Revised robustness measure

PISA defines the robustness of the plan, as the fraction of the domains where a given plan is executable given each possible domain is equally likely. More generally, the robustness value can be defined as the cumulative probability of each possible domain, where the robustness for a given plan π would be defined as

$$R(\pi) = \sum_{\mathcal{M}_i, \text{ where } G \subseteq \gamma_{\mathcal{M}_i}(\pi, I)} Pr(\mathcal{M}_i) \tag{5.1}$$

For C-PISA, we need to measure the robustness given the set of successful plan traces $\mathcal{T}$.

$$R(\pi|\mathcal{T}) = \sum_{\mathcal{M}_i, \text{ where both } \pi \text{ and } \mathcal{T} \text{ are valid}} Pr(\mathcal{M}_1|\mathcal{T}) \tag{5.2}$$

Now let the set of constraints for $\mathcal{T}$ be $\Sigma_{\mathcal{T}}$ and the constraints for the plan be Σ_π. Then the above equation becomes

$$R(\pi|\mathcal{T}) = \frac{Pr(\pi \wedge \mathcal{T})}{Pr(\mathcal{T})} = \frac{WMC(\Sigma_\pi \wedge \Sigma_{\mathcal{T}})}{WMC(\Sigma_{\mathcal{T}})} \tag{5.3}$$

[1]There are certain possible domains that the cases can never eliminate. For example, if one of the possible precondition p of an action a is an actual precondition in the D*. Then any given plan trace executable in D* is also executable in a domain where p is not the precondition of a

So if a plan has a robustness value of 1, this means that the plan is valid on all domains where the plan traces $\mathcal{T}$ is valid.

5.2 Comparing C-PISA against PISA

In order to compare this new planner for APDDL, we compared our system against PISA for two separate domains, namely zenotravel and rover to demonstrate it's utility. Both of these domains were used in evaluating the original approach. In our original test, the incomplete domains were created by moving random predicates to possible predicate section and introducing few new predicates. But in our new test we created incomplete domain files by adding more incorrect random predicates (i.e predicates that were not originally part of an action) to possible predicates section than in our previous tests. This ensures that we have more incorrect domains, which in turn can be eliminated with the help of cases. While adding these new predicates, we made sure that the argument list of each new predicate was compatible with the signature of the action. Next, we created a set of problems of varying sizes for each domain, which we then solved with respect to the actual domain using an off the shelf planner. The solutions to these problems formed our case library.

Some of these cases might implicitly eliminate some domains from consideration. For example, in the case of zenotravel, Let us assume we have an action board with the following action definition

```
(:action board
    (?p - person ?a - aircraft ?c - city)
    :precondition
```

```
        (and
            (at ?a ?c)
            (at ?p ?c)
        )
    :poss-precondition
        (and
        )
    :effect
        (and
            (in ?p ?a)
        )
    :poss-effect
        (and
            (not (at ?a ?c))
        )
)
```

As shown in the action definition, the action **board** has a possible effect $(not(at?a?c))$, which means that after boarding the plane will no longer be in that city.

Next let us consider a plan trace containing the following actions

(board person1 plane3 city4)

(fly plane3 city4 city5 fl4 fl3)

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

(not (at ?a ?c))

In the case of zeno travel domain, we created an incomplete domain by introducing five possible preconditions, five possible add effects and nine possible delete effects. Out of these possible predicates, nine of them are part of the actual base domain. While 10 predicates were incorrect predicates added randomly. We created 15 case files by solving random problems using the base domain. The 19 possible predicates represents 524288 possible number of domains, of which only 1024 domains satisfy all the provided cases.

We chose to test this domain on 15 problems of varying sizes and we ran the same problems using the original PISA system and with differing number of cases. Each system was run with a 15 minutes time limit per problem. Once we had the resulting plans for all the problems, we collected the best plan for each problem as produced by the respective approaches. To make the comparison more accurate, the robustness of plans generated by the original approach was re-evaluated in the smallest known domain space (in the case of zenotravel domain it corresponds to running with all 15

cases). Since both approaches rely on stochastic search, each planning instance was run multiple times to ensure that the search was not getting stuck on undesirable search paths. During each search instance, we used upper bound for the robustness approximation. The result of the comparison is given in Figures 4 and 5.

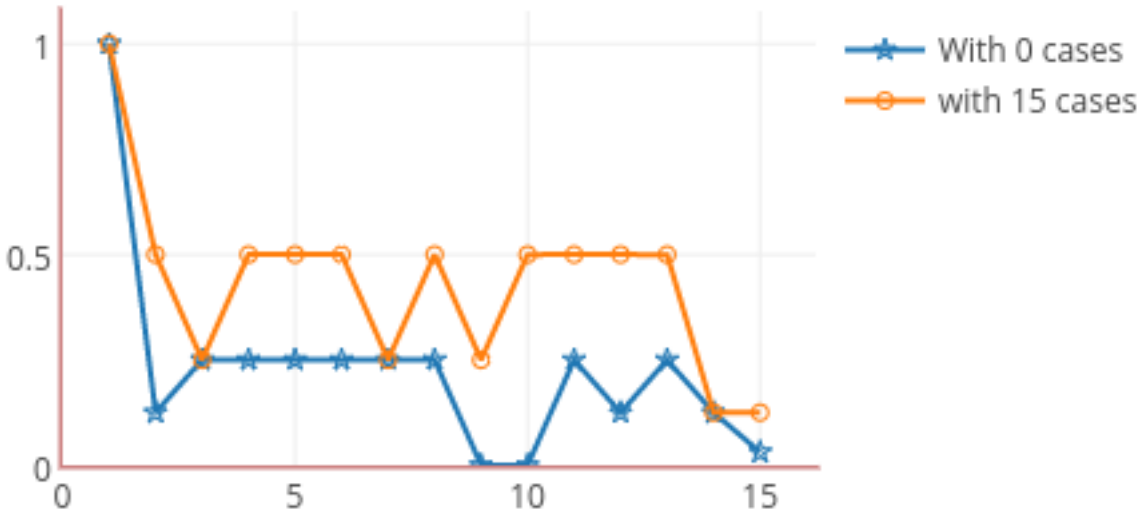

Figure 4. Comparing the plans produced with 10 cases against 15 cases

The results in Figure 4 and 5 illustrate that the plans produced in the presence of full case library are consistently more robust. In Figure 4 we see that C-PISA running with full case library was able to outperform the original approach (which

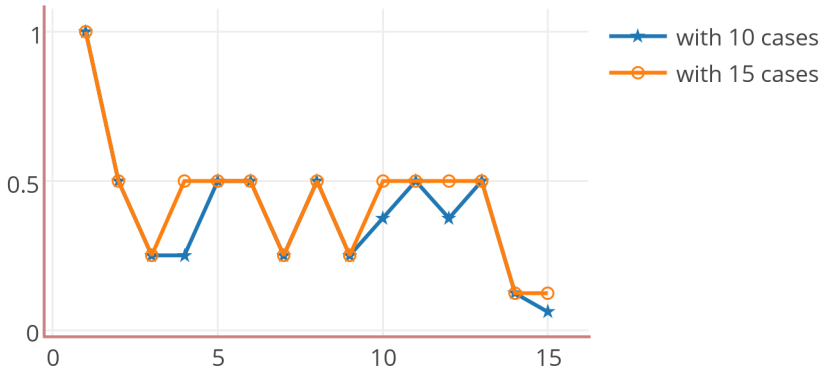

Figure 5. Comparing the plans produced with 10 cases against 15 cases

is same as running C-PISA in the absence of any case files) in 9/15 problems. It is interesting to note that PISA in fact twice chose plans which ended up having zero robustness in the reduced domain space. In Figure 5, we compare the outputs of C-PISA running with full case library against C-PISA running on a subset of the total cases. Again we see that even though the full set only had five additional cases, it was still able to produce better plans in many problems.

Next for the Rover domain, we created an incomplete domain similar to zenotravel. Our incomplete domain had 21 possible effect predicates and four possible precondition predicates. This includes a combination of existing predicates moved to possible section and new predicates. In total, this represents 33554432 possible domains. Next, we created four cases for the base domain, such that the four cases reduced the possible number of domains down to 65535.

We tested the domain using ten problems of different sizes. Similar to zenotravel,

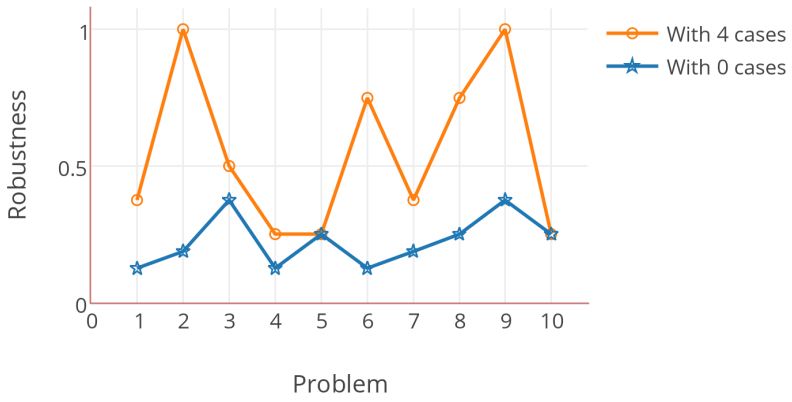

Figure 6. Comparing the plans produced with 0 cases against 4 cases

each rover problem was tested with the original approach and then tested with our extension for varying sizes of case library. The best plan produced by each variation of the approach is evaluated in the reduced domain space to obtain the best possible approximation of the robustness. Similar to zenotravel each problem was run multiple times. The result of the experiment is shown in Figure 6 and Figure 7.

Again we see that with more cases, we are able to produce more robust plans. An interesting point to note here is the robustness of the best plan produced for the third problem. Here the robustness of the best plan produced by the original approach is better than the robustness of the plan produced with two cases, as evaluated in the smallest domain space. This is because the effect of cases is not uniform across every plan. As seen here, the addition of two new cases (bringing the total number of cases to four) affected one plan (the plan selected as the best plan with two cases) more drastically than the other.

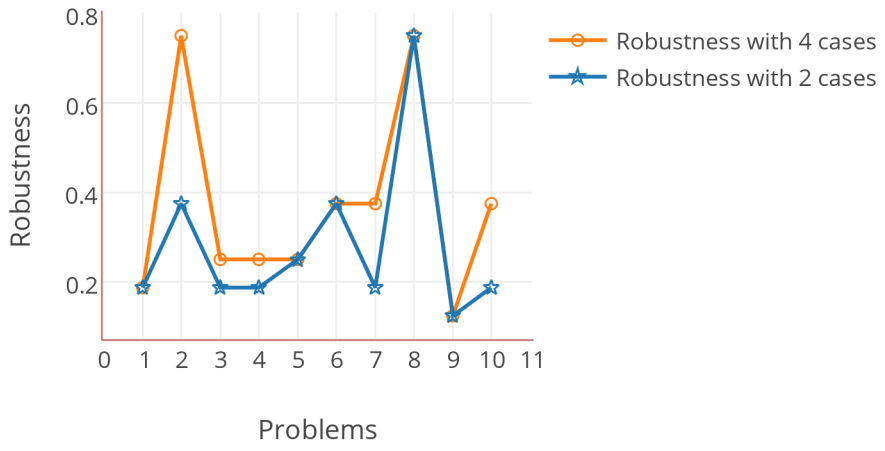

Figure 7. Comparing the plans produced with 2 cases against 4 cases

As is evident from the above results, the new case-based approach was able to produce plans with better robustness in the majority of the problems. One interesting point we noticed was that the plans that appeared to have better robustness in larger domain space, need not be the better choice when we reduce the domain space. In many cases, the original approach chose plans with worse actual robustness over those with better actual robustness values[2], as they seemed to have higher robustness in the larger domain space.

[2]please note we use the term actual robustness value to represent the robustness value of a plan in the smallest known domain space. In our case, this usually represents all domains that satisfy the cases

5.3 C-PISA as a way to further reduce domain-modeling burden

One of the major drawbacks of APDDL that we pointed out in the last chapter was the difficulty of creating an APDDL domain. A correct APDDL domain, expects the domain writer to be not only aware of the fact that the action definition created may be incomplete, but is also tasked with coming up with a full list of possible predicates. One of the strong assumptions made by the approach is the fact that any of the missing predicates are present in the list of possible predicates. C-PISA provides a way of simplifying this process by assuming each action is incomplete and by considering all predicates that are not already part of the action definition. For example, consider an action a with c_p known preconditions, c_a known add effects and c_d known delete effects. If the total number of predicates be N, then the number of possible preconditions would be equal to $N - c_p$, possible add effects will be $N - c_a$ and possible deletes will be $N - c_d$. Clearly the number of possible predicates will be quite high, but we can leverage the successful plan traces to remove the incorrect annotations.

In order to support all possible predicates as the annotations, we can add some extra constraints to simplify the domain. For example, whenever we confirm an add effect, the equivalent delete effect for the same predicate be removed from the action

$$p_{a_i}^{\tilde{add}} \implies \neg p_{a_i}^{\tilde{del}} \tag{5.4}$$

Similarly if we are adding a predicate to the add effect, we can remove it from the possible precondition list

$$p_{a_i}^{\tilde{add}} \implies \neg p_{a_i}^{\tilde{pre}} \tag{5.5}$$

5.3.1 C-PISA against RIM

Now in order to evaluate the performance of C-PISA on incomplete domains without any annotations, we need to compare against a method that is designed to work with such domains. For this we choose to compare our system against RIM[] a MAX-SAT based approach that has been proposed to refine incomplete models based on information available from successful plan traces. For each plan trace, the RIM system extracts a set of soft and hard constraints to capture the executability of each plan. The constraints are then solved using a MAX-SAT solver to identify a domain that can support the most number of the given constraints. One interesting aspect of this approach that differentiates it from PISA is the fact that the approach chooses to identify a single domain that satisfies the most number of constraints. This makes the planning process easier, but if the model that has been identified was incorrect, this can cause all the plans produced by the domain to fail. C-PISA, on the other hand, takes a safer approach by trying to create plans that can satisfy the largest number of domains. One way RIM tries to prevent creating incorrect plans is by making use of macro operators (a sequence of actions), learned from the traces. By using a sequence of actions already present in the traces, there is a higher chance that the actions in the sequences are executable.

5.3.2 Evaluation

In order to evaluate the performance of C-PISA in relation with RIM, we compare the two systems for the domain zenotravel. For the domain random preconditions and effects are removed from the domain file. This incomplete domain will be used

directly by the RIM system, while C-PISA uses an equivalent annotated model. The equivalent annotated model is created by adding all possible predicates not already part of the precondition and effects of each action. Now the plans produced by each of the systems are evaluated based on the robustness of the plans being produced. It is important to note that RIM system itself has no concept of robustness and by default, there is no way of evaluating the validity of the plans in the absence of the ground truth.

In zenotravel, with ten missing predicates from the ground truth (with 6 missing preconditions and 4 missing effects). This is converted to an annotated pddl model with 56 annotations (20 possible preconditions and 36 possible effects), which is equivalent to 72057594037927936 domains. We take into account eight possible cases that were randomly generated, next we try to solve 18 randomly selected plans of various sizes with varying subsets of total cases(a single plan trace, two cases, four cases and the complete set). Each plan being generated by both RIM and C-PISA are then evaluated using the entire case set.

As apparent from Figures 8, 9 and 10 C-PISA clearly does better than RIM for smaller number of cases. This is because even in the absence of cases the ground truth is still part of the completion list of the domain. While for RIM, the domain selected would most likely won't be the ground truth. Especially for the first problem C-PISA was able to produce plans with robustness one, this means that those plans were guaranteed to be executed, while RIM was not even able to produce plans for these problems. While in Figure 11 we see RIM starts doing better, this would be true in general for RIM, as RIM is able to learn more and more macro operators. As

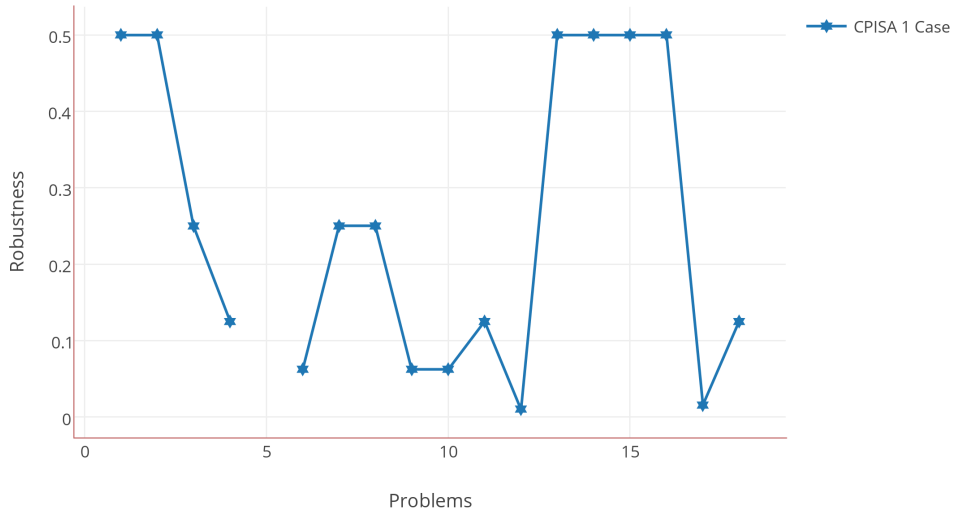

Figure 8. Comparing plans produced by C-PISA vs RIM provided a single case

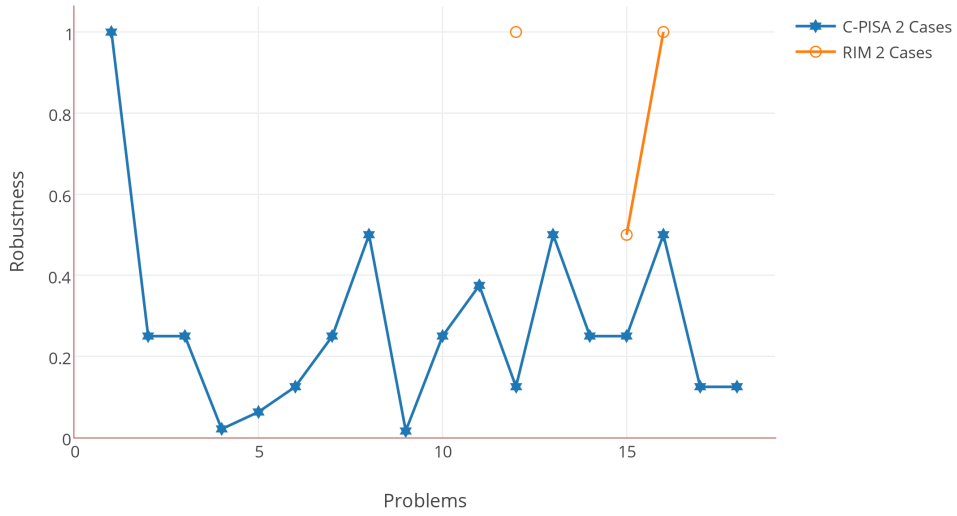

Figure 9. Comparing plans produced by C-PISA vs RIM provided two single cases

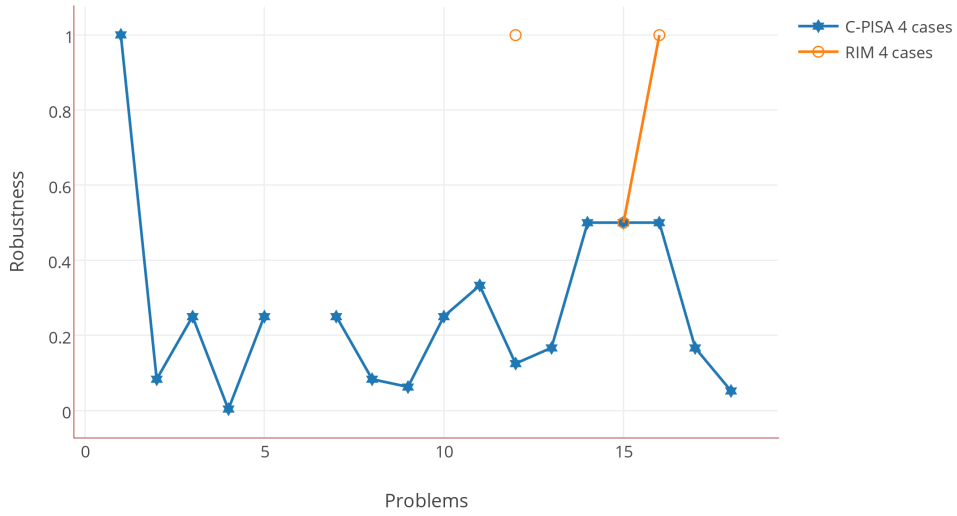

Figure 10. Comparing plans produced by C-PISA vs RIM provided with four cases

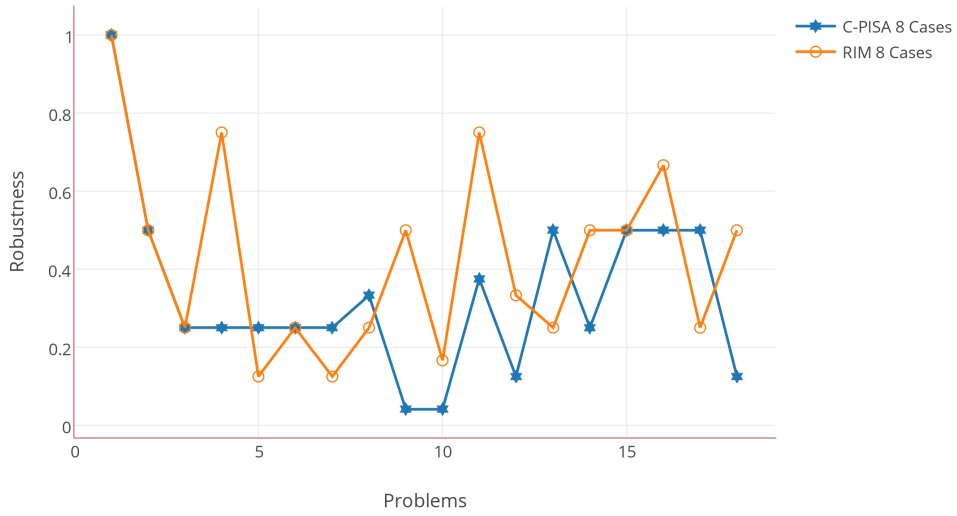

Figure 11. Comparing plans produced by C-PISA vs RIM provided with eight cases

macro operators are action sequences that were already seen in the plan traces, using these sequences will not add any new constraints and will not reduce the robustness of a plan.

Chapter 6

FUTURE WORKS

One of the main reasons for using PISA or C-PISA over other model-lite planning models would be because of the guarantees it can provide regarding the plans it generated. For example, if a plan generated by PISA has a robustness value of one, we can be sure that the plan can be executed on the ground truth. This property can be extremely valuable when we are dealing with scenarios where security and safety are a major concern. In such cases, by limiting execution to plans with higher robustness, we have a greater chance of success. This is a property that is sorely missing in other approaches like RIM, which chooses to use the most likely domain and does not provide any guarantees about the plans it produces. Another way we could possibly utilize APDDL models for safety would be to adapt robustness measure to also reflect models to also consider safety constraints. So now the robustness measure would be defined as

$$R_{safe}(\pi) = \frac{WMC(\Sigma_{\mathcal{T}} \vee \Sigma_{safe} \vee \Sigma_{\pi})}{WMC(\Sigma_{\mathcal{T}} \vee \Sigma_{safe})} \tag{6.1}$$

Where Σ_{safe} are a set of constraints related to domain safety. Now the robustness measure becomes equal to the ratio between the number of domains where both plan constraints and safety constraints are met. The safety constraints can either be specified (for example we could have constraints similar to the one listed in Weld and Etzioni 1994) or can be learned from safe and even possibly unsafe plans (we would have to use the negation of the constraints learned from possibly unsafe plans). In future, I hope to further investigate such connections between APDDL and safety.

Another possible future direction would be to investigate other APDDL related

tasks. One of the more useful tasks would be to increase the robustness of a given plan, which has a lot of possible applications. One way we can perform this would be by replacing part of the plan with the lowest robustness value. If we are given a plan $\pi = \{a_1, a_2, .., a_n\}$, then we need to find the robustness value of each possible plan prefix π_i, where $\pi_i = \{a_1, a_2, .., a_i\}$. Next, we identify plan prefix with the greatest reduction in robustness value compared to the previous step (i.e the plan prefix with the smallest value for the ratio $\frac{R(\pi_i)}{R(\pi_{i-1})}$ with $R(\pi_0) = 1$). Now we rerun the PISA/C-PISA search with π_{i-1} set as the initial plan prefix (instead of an empty plan prefix) and set the robustness of the original plan as the initial robustness threshold of the search. We can repeat this till the robustness values converge.

Another possible extension to C-PISA we could consider to produce higher robustness plan would be to consider macro operators similar to RIM. By reusing sequences seen before we can ensure that the actions do create any new additional constraints.

Part II

Multi-agent planning

Note: The following section's content were taken from the paper Zhang, Sreedharan, and Kambhampati 2016, which was presented at ICAPS 2016, and for which I was a second author.

Chapter 7

MOTIVATION AND RELATED WORKS

Despite the increased interest in multi-agent planning, one question has remained largely unaddressed: "*under what conditions are multiple agents actually needed to solve a planning problem?*" This question is of fundamental importance as it clearly demarcates two distinct uses of multiple agents in a given planning problem: (i) the situations where multiple agents are used because there is no single agent plan for the problem, and (ii) those situations where multiple agents are used to improve execution efficiency, even though a single agent plan is in fact feasible. This latter class of problems can arguably be viewed as an easier form of multi-agent planning problems, in as much as they can be solved by first generating a single-agent plan, and then using a post-processing step to optimize the execution cost by deploying multiple agents. Without keeping such demarcation in mind when designing benchmark domains, it can be misleading to compare and evaluate multi-agent planners – in the extreme case, very fast planners can be designed to solve only problems that do not require cooperation. Unfortunately, we shall see that most of the domains used in CoDMAP, a proto multi-agent planning competition at ICAPS 2015, need multiple agents only for plan efficiency rather than feasibility.

The aim of this section is precisely to shed light on this central question of when multiple agents need to be involved in the plan to solve a problem.[3] In particular, we hope to address the following: $Q1$) Given a multi-agent planning (MAP) problem to

[3]Our focus in this work is on the number of agents involved in the plan, rather than the planning process by which the plan is made; see related work for details.

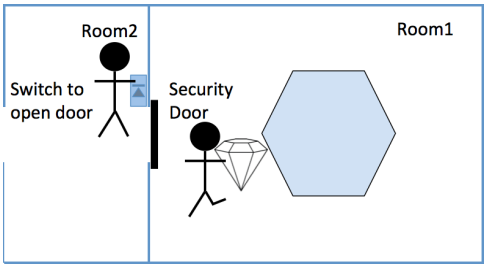

Figure 12. **Burglary problem** – The goal of this problem is to steal a diamond ($diamond1$) from $room1$, in which the diamond is secured, and place it in $room2$. The diamond is protected by a security system. If the diamond is taken, the system locks the door ($door1$) of $room1$, so that the insiders cannot exit. There is a switch ($switch1$) to manually open $door1$ but it is located in $room2$

solve, what are the conditions that make cooperation between multiple agents *required*; $Q2$) How do these conditions affect planning for MAP problems; $Q3$) Can we determine the minimum number of agents required for a MAP problem. The answer to the first question separates multi-agent planning problems from single-agent planning (SAP) problems in a fundamental way. The answer to the second question can inform the design of future multi-agent planning competitions (e.g., CoDMAP in ICAPS 2015) so that MAP problems that require cooperation can be deliberately introduced to obtain a clearer understanding of the capabilities of the competing planners. The answer to the third question has real-world applications, e.g., determining the minimum number of agents to be assigned.

However, as we shall see shortly, determining whether a MAP problem requires cooperation is in general no easier than finding a plan for the problem itself, which is PSPACE-complete. Hence, instead of providing exact answers to questions $Q1 - Q3$ above, we provide answers in restrictive settings (i.e., for subsets of MAP problems). First of all, given that the most obvious reason for a MAP problem to require cooperation is when capabilities of different agents are needed, we divide MAP problems into two classes: the class of problems where agents have the same capabilities

Initial State:

$location(agent1)\ room1$
$location(agent2)\ room1$
$location(diamond1)\ room1$
$doorLocked(room1)\ false$
$location(switch1)\ room2$

Goal State:

$location(diamond1)\ room2$

Operators:

$WalkThrough(agent, door, fromRoom, toRoom)$:

prv: $doorLocked(door)\ false$
pre: $location(agent)\ fromRoom$
$post$: $location(agent)\ toRoom$

$Steal(agent, diamond, room, door)$:

prv: $location(agent)\ room$
pre: $location(diamond)\ room$
$post$: $doorLocked(door)\ true$
$post$: $location(diamond)\ agent$

$Switch(agent, switch, room, door)$:

prv: $location(switch)\ room$
prv: $location(agent)\ room$
$post$: $doorLocked(door)\ false$

$Place(agent, diamond, room)$:

prv: $location(agent)\ room$
pre: $location(diamond)\ agent$
$post$: $location(diamond)\ room$

Figure 13. The problem and domain descriptions of the Burglary problem using SAS$^+$ in which the value is immediately specified after each variable.

(i.e., homogeneous agents) and the class where agents have different capabilities (i.e., heterogeneous agents).

For the first and more restrictive class with homogeneous agents, it may appear that cooperation is only required when joint actions are present (which are actions that must be executed by multiple agents at the same time). This intuition, however, is falsified by a simple problem with only sequential actions as shown in Fig. 12, which is referred to as the **Burglary problem** in the later part of this section. We show that, in this class of problems, RC can be caused by the "traversability" or the "causal loops" in the causal graph of the agent state variables. Our main theorem shows that these two causes are exhaustive when agents are homogeneous: if the causal graph of

agents is traversable and contains no causal loops, a single agent alone is sufficient for the problem ($Q1$). However, none of them individually represent a sufficient or necessary condition for RC. When these two causes are present in a problem, we show that upper bounds of the number of agents required can be provided ($Q3$).

For the second class where agents can have different capabilities, the analysis becomes more complex. Hence, we further divide the problems in the second class into two subclasses: the subclass of problems where the causes of RC in the first class do not appear, and the subclass of the remaining problems. For the first subclass, one observation is that the number of agents that are required can often be significantly reduced if agents are simply made to be "transformable" ($Q3$). Although this does not necessarily lead to the reduction of agents in the final plan for these problems, we show that the planning performance can be significantly improved ($Q2$). The second subclass corresponds to the most difficult setting and an analysis needs to be provided in future work. Finally, as a practical contribution of our investigation we develop a planner called RCPLAN, based on the idea of transformable agents above to efficiently solve MAP problems. We show that RCPLAN outperforms one of the best performers in the IPC CoDMAP competition.

7.1 Related Work

The term "multi-agent planning" has traditionally been quite loosely defined as "planning in the presence of multiple agents" (c.f. Jonsson and Rovatsos 2011).

This definition blurs multi-agent plans and distributed planning by confounding two distinct types of agents: agents that are involved in the planning process ("planning agents") and agents involved in plan execution ("execution agents").

A multi-agent plan is one that involves multiple execution agents; whereas distributed planning involves using multiple *planning agents*, which, normally, also happen to be the execution agents. However, these two types of agents have orthogonal properties: it is possible to have multiple planning agents working together make a plan involving a single execution agent, just as it is possible to have a single planning agent make a plan involving multiple execution agents (aka "centralized multi-agent planning").

When we refer to "multi-agent planning" problems in this section of the document, our focus is on the plans of these problems for execution agents, regardless of how many planning agents were involved in the planning process. Such plans may be the result of centralized Kvarnstrom 2011; Muise, Lipovetzky, and Ramirez 2015 or distributed planning

Our analysis of required cooperation in multi-agent plans is similar in spirit to Cushing et. al.'s analysis of temporal planning Cushing, Kambhampati, et al. 2007; Cushing, Weld, et al. 2007. Just as concurrency is sometimes seen as a way to improve the running time of the plan, execution agents are someimes viewed as "resources" that can be added to the plan to improve its efficiency Pape 1990. While Cushing et. al. focus on characterizing conditions where concurrency is required to solve a planning problem, we focus on conditions where cooperation between multiple execution agents is required to solve a planning problem.

Our analysis of required cooperation uses SAS^+ formalism Backstrom and Nebel 1996 with *causal graphs* Knoblock 1994; Helmert 2006, which are often discussed in the context of factored planning Bacchus and Yang 1993; Amir and Engelhardt 2003; Brafman 2006; Brafman and Domshlak 2013. A causal graph captures the interactions

between state variables; intuitively, it can also capture the interactions between agents since they affect each other through these variables Brafman and Domshlak 2013.

In the next section, we start by looking at the definition of required cooperation and the various subclasses of multi-agent problems, we will be looking at. Next, we look at each of these sub-classes separately and try to identify the conditions in each subclass which can lead to required cooperation. We also propose a new planner that makes use of the ideas discussed in the previous section to solve a sub-set of multi-agent problems more efficiently.

Chapter 8

REQUIRED CO-OPERATION

For this work, we focus on required cooperation (RC) in scenarios with instantaneous actions and sequential execution. The possibility of RC can only increase when we extend the model to the temporal domains in which concurrent or synchronous actions must be considered. We develop our analysis of RC based on SAS$^+$ Backstrom and Nebel 1996.

Definition 1 *A SAS$^+$ problem is given by a tuple $P = \langle V, A, I, G \rangle$, where:*

- *$V = \{v\}$ is a set of state variables. Each variable v is associated with its domain $D(v)$.*
- *$A = \{a\}$ is a set of actions (i.e., ground operators). Each action a is a tuple $\langle pre(a), post(a), prv(a) \rangle$, in which $prv(a)$ denotes prevail conditions which are preconditions that persist through a.*
- *I and G denote initial and goal state, respectively.*

A plan in SAS$^+$ is often defined to be a total-order plan, which is a sequence of actions. For details of SAS$^+$, see Backstrom and Nebel 1996. To extend the SAS$^+$ formalism to MAP, we minimally modify the definitions.

Definition 2 *A SAS$^+$ MAP problem is given by a tuple $P = \langle V, \Phi, I, G \rangle$ ($|\Phi| > 1$), where $\Phi = \{\phi\}$ is the set of agents; each agent ϕ is associated with a set of actions $A(\phi)$.*

Definition 3 *A plan π_{MAP} in MAP is a sequence of agent-action pairs $\pi_{MAP} = \langle (a_1, \phi(a_1)), ..., (a_L, \phi(a_L)) \rangle$, in which $\phi(a_i)$ represents the agent executing the action a_i and L is the length of the plan.*

We assume that the reference of the executing agent is encoded and appears as the first argument in an action, similar to the operators (ungrounded actions) in Fig. 13.

8.1 Required Cooperation for MAP Problems

Next, we formally define the notion of *required cooperation* and a few other terms used. We assume throughout this section that more than one agent is considered (i.e., $|\Phi| > 1$).

Definition 4 (k-agent Solvable) *Given a MAP problem $P = \langle V, \Phi, I, G \rangle$ ($|\Phi| \geq k$), the problem is k-agent solvable if $\exists \Phi_k \subseteq \Phi$ ($|\Phi_k| = k$), such that $\langle V, \Phi_k, I, G \rangle$ is solvable.*

Definition 5 (Required Cooperation (RC)) *Given a MAP problem $P = \langle V, \Phi, I, G \rangle$, there is required cooperation if P is solvable but not 1-agent solvable.*

In other words, given a MAP problem that satisfies RC, any plan must involve more than one agent. Note also that to satisfy RC, a MAP problem must first be solvable.

Lemma 1 *Given a solvable MAP problem $P = \langle V, \Phi, I, G \rangle$, determining whether it satisfies RC is PSPACE-complete.*

First, it is not difficult to show that the RC decision problem belongs to PSPACE, since we only need to verify that $P = \langle V, \phi, I, G \rangle$ is unsolvable for all $\phi \in \Phi$, given that

the initial problem is known to be solvable. Then, we complete the proof by reducing from the PLANSAT problem, which is PSPACE-complete in general Bylander 1991. Given a PLANSAT problem (with a single agent), the idea is that we can introduce a second agent with only one action. This action directly achieves the goal but requires an action of the initial agent (with all preconditions satisfied in the initial state) to provide a precondition that is not initially satisfied. We know that this constructed MAP problem is solvable. If the algorithm for the RC decision problem returns that cooperation is required for this MAP problem, we know that the original PLANSAT problem is unsolvable; otherwise, it is solvable.

Definition 6 (Minimally k-agent Solvable) *Given a MAP problem $P = \langle V, \Phi, I, G \rangle$ ($|\Phi| \geq k$), P is minimally k-agent solvable if it is k-agent solvable, and not $(k-1)$-agent solvable.*

Corollary 1 *Given a solvable MAP problem $P = \langle V, \Phi, I, G \rangle$, determining the minimally solvable k ($k \leq |\Phi|$) is PSPACE-complete.*

Hence, directly querying for RC is intractable even when the problem is known to be solvable. Instead, we aim to identify conditions that can potentially cause RC. First, note that although actions (or ground operators) are unique for each agent, they may be identical except for the executing agent.

Definition 7 (Action Signature (AS)) *An action signature is an action with the reference of the executing agent replaced by a global AG_{EX} symbol.*

For example, an action signature in the IPC logistics domain is $drive(AG_{EX}, pgh\text{-}poT, pgh\text{-}airport)$. As a result, different agents can share the same action signatures. We denote the set of action signatures for any $\phi \in \Phi$ as $AS(\phi)$.

Definition 8 (Agent Variable (Agent Fluent)) *A variable (fluent) is an agent variable (fluent) if it is associated with the reference of an agent.*

Agent variables are used to specify agent state. For example, $location(truck\text{-}pgh)$ is an agent variable since it is associated with an agent *truck-pgh*. We use $V_\phi \subseteq V$ to denote the set of agent variables that are associated with an agent ϕ (i.e., variables that are present in the initial state or actions of ϕ), and V_o to denote the set of non-agent variables. Furthermore, we assume that agents can only interact with each other through non-agent variables (i.e., V_o).[4] This assumption implies that agent variables are associated with one and only one reference of an agent. Thus, we have $V_\phi \cap V_{\phi'} \equiv \emptyset$ $(\phi \neq \phi')$.

Definition 9 (Variable (Fluent) Signature (VS)) *Given an agent variable (fluent), its signature is the variable (fluent) with the reference of agent replaced by* AG_{EX}.

For example, $location(truck\text{-}pgh)$ is an agent variable for *truck-pgh* and its signature is $location(AG_{EX})$. We denote the set of variable signatures for V_ϕ as $VS(\phi)$, and use VS as an operator such that $VS(v)$ returns the signature of a variable v (it returns any non-agent variable unchanged).

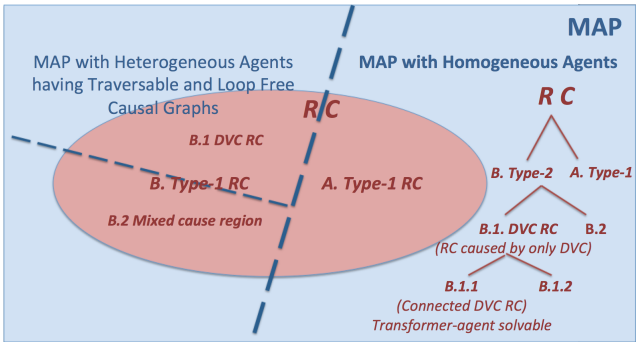

Figure 14. Division of MAP problems into MAP with heterogeneous and homogeneous agents. Consequently, RC problems are also divided into two classes: Type-1 RC involves problems with homogeneous agents (A) and Type-2 RC involves problems with heterogeneous agents (B). Type-1 RC is only caused when the causal graph is non-traversable or contains loops. Type-2 RC problems are further divided into DVC RC problems ($B.1$) where RC is caused only by the heterogeneity of agents, and RC problems with mixed causes ($B.2$). $B.1.1$ and $B.1.2$ represent DVC RC problems with and without connected state spaces, respectively.

8.2 Classes of Required Cooperation (RC)

In this work, we focus on MAP problems with goals that do not involve agent variables (i.e., $G \cap V_\phi = \emptyset$) (because having agent variables in goals forces RC in a trivial way). We divide MAP problems into two classes to facilitate the analysis of RC. The division of MAP problems (the rectangle shaped region) as shown in Fig. 14 correspondingly also divides RC problems (the oval shaped region) into two classes based on the heterogeneity of the agents:

Agent Heterogeneity: Given a MAP problem $P = \langle V, \Phi, I, G \rangle$, the heterogeneity of the agents can be characterized by these conditions: 1) *Domain Heterogeneity*

[4] It is possible to compile away exceptions by breaking an agent variable (with more than one reference of agent) into multiple agent variables and introducing non-agent variables to correlate them. Given that this compilation only increases the problem size linearly (in the number of agents) for each such agent variable, it does not influence our later discussions.

(DH): $\exists v \in V_\phi$ and $D(V') \setminus D(v) \neq \emptyset$, in which $V' = \{v' | v' \in V_{\phi'} (\phi' \neq \phi)$ and $VS(v) = VS(v')\}$. 2) *Variable Heterogeneity (VH)*: $\exists \phi \in \Phi, VS(\Phi \setminus \phi) \setminus VS(\phi) \neq \emptyset$. 3) *Capability Heterogeneity (CH)*: $\exists \phi \in \Phi, AS(\Phi \setminus \phi) \setminus AS(\phi) \neq \emptyset$. $D(V)$ above denotes the joint domain of all $v \in V$.

We define heterogeneous agents as a set of agents in which DH, VH or CH is satisfied for any agent. This condition is also referred to as the heterogeneity condition. In contrast, we define homogeneous agents as a set of agents in which neither DH, VH nor CH is satisfied for any agent. This allows us to divide RC problems into:

Definition 10 (Type-1 (Homogeneous) RC) *An RC problem belongs to type-1 RC if the heterogeneity condition is not satisfied (i.e., agents are homogeneous).*

Definition 11 (Type-2 (Heterogeneous) RC) *An RC problem belongs to type-2 RC if $\exists \phi \in \Phi$, such that DH or VH or CH is satisfied.*

It is worth noting that when considering certain entities (e.g., truck and plane in the logistics domain) as agents rather than as resources (such as in CoDMAP competition Stolba, Komenda, and Kovacs 2015), many problems in the IPC domains have RC and belong to type-2.

Chapter 9

ANALYSIS OF TYPE-1 (HOMOGENEOUS) RC

We start with type-1 RC which represents a class of more restrictive problems and hence are easier to analyze.

Type-1 RC Caused by Traversability: One condition that can cause RC in type-1 RC problems is the traversability of the state space. One obvious example is related to non-restorable resources such as energy. For example, a robot may have all the capabilities to achieve the goal but insufficient amount of battery power. Since traversability is associated with the evolution of variables and their values, we analyze it using causal graphs.

Definition 12 (Causal Graph) *Given a MAP problem $P = \langle V, \Phi, I, G \rangle$, the causal graph G is a graph with directed and undirected edges over the nodes V. For two nodes v and v' ($v \neq v'$), a directed edge $v \rightarrow v'$ is introduced if there exists an action that updates v' while having a prevail condition associated with v. An undirected edge $v - v'$ is introduced if there exists an action that updates both.*

An example of a causal graph for an agent is in Fig. 15. When we use agent

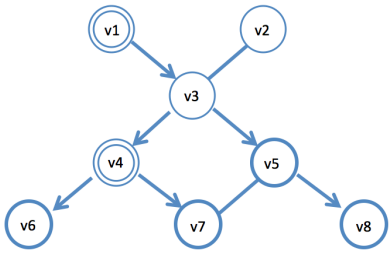

Figure 15. Example of a causal graph.

variable signatures to replace agent variables in the graph, the causal graphs for all agents in type-1 problems are the same. We refer to any of these graphs as an *individual causal graph signature* (ICGS). Next, we define the notions of *closures* and *locally traversable state space*.

Definition 13 (IC and its OC) *Given a causal graph, an inner closure (IC) is any set of variables for which no other variables are connected to them with undirected edges; an outer closure (OC) of an IC is the set of nodes that have directed edges going into nodes in the IC.*

In Fig. 15, $\{v_2, v_3\}$ and $\{v_4\}$ are examples of ICs. The OC of $\{v_2, v_3\}$ is $\{v_1\}$ and the OC of $\{v_4\}$ is $\{v_3\}$.

Definition 14 (Locally Traversable State Space) *An inner closure (IC) in a causal graph has a locally traversable state space if and only if: given any two states of this IC, denoted by s and s', there exists a plan that connects them, assuming that the state of the outer closure (OC) of this IC can be changed freely within its state space.*

In other words, an IC has a locally traversable state space if its traversal is only dependent on the variables in its OC. This also means that when the OC of an IC is empty, the state of the IC can change freely. In the case of non-restorable resources, the ICs that include variables for these resources would not satisfy this requirement (hence non-restorable). When all the ICs in the causal graph of an agent satisfy this, the causal graph is referred to as being *traversable*.

Type-1 RC Caused by Causal Loops: A problem with a traversable causal graph, however, may still require RC. Let us revisit the Burglary problem in Fig. 12. We construct the *individual causal graph signature* (ICGS) for this type-1 RC example in Fig. 16. It is not difficult to verify that this ICGS is traversable, given that

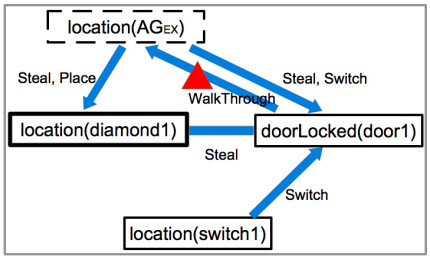

Figure 16. ICGS for the Burglary problem to illustrate causal loops that cause RC in type-1 RC problems. Actions (without arguments) are labeled along with their corresponding edges. Variables in G are shown as bold-box nodes and agent variable signatures are shown as dashed-box nodes.

$location(diamond1)$ and $doorLocked(door1)$ form an IC with the other two variables as its OC. More specifically, when assuming that the agent location can change freely, we can also update the location of the diamond as well as the status of the door to arbitrary values. One key observation is that a single agent cannot address this problem due to the fact that $WalkThrough$ with the diamond to $room2$ requires $doorLocked(door1) = false$, which is violated by the $Steal$ action to obtain the diamond in the first place. This is clearly related to the causal loop in Fig. 16:

Definition 15 (Causal Loop) *A causal loop in a causal graph is a directed loop that contains at least one directed edge (undirected edges are considered as edges in both directions when checking for loops).*

9.0.1 When Cooperation Is Not Required

The following theorem establishes that the two causes discussed above are exhaustive – when none of them are present in a solvable MAP problem with homogeneous agents, it can be solved by a single agent.

Theorem 1 *Given a solvable MAP problem with homogeneous agents, and for which the individual causal graph signatures (ICGSs) are traversable and contain no causal loops, any single agent can also achieve the goal.*

Given no causal loops, the directed edges in the ICGS divide the variables into stratified levels, in which: 1) variables at each level do not appear in other levels; 2) higher level variables are connected to lower level variables with only directed edges going from higher levels to lower levels; 3) variables within each level are either not connected, or connected with undirected edges. For example, the variables in Fig. 15 are divided into the following levels (from high to low): $\{v_1\}$, $\{v_2, v_3\}$, $\{v_4\}$, $\{v_5, v_7\}$, $\{v_6, v_8\}$. Note that this division is not unique.

The intuition is to show that there exists a single agent plan that can lead to any goal state given an initial state. We prove the result by induction on the level. Suppose that the ICGS has k levels and the following holds: given any trajectory of states for all variables, there exists a plan whose execution traces of states include this trajectory in the correct order.

When the ICGS has $k+1$ levels: given any state s for all variables from level 1 to $k+1$, we know from the assumption that the ICGS is traversable that there exists a plan that can update the variables at the $k+1$ level from their current states to the corresponding states in s. This plan, denoted by π, requires the freedom to change the states of variables from level 1 to k. Given the induction assumption, we know that we can update these variables to their required states in the correct order to satisfy π; furthermore, these updates (at level k and above) do not influence the variables at the $k+1$ level (hence do not influence π). Once the states of the variables at the $k+1$ level are updated to match those in s, we can then update variables at level 1 to k to match their states in s accordingly. Using this process, we can incrementally

build a plan whose execution traces of states contain any trajectory of states for all the variables in the correct order.

Furthermore, the induction holds when there is only one level given that ICGS is traversable. Hence, the induction conclusion holds. The main conclusion directly follows.

Note that Theorem 1 provides an answer for the inverse of the first question (Q1) in the introduction: Theorem 1 is used to determine when cooperation is not required instead of when it is. More specifically, the conjunction of the conditions (traversable ICGS and no causal loop) is a sufficient condition for a MAP problem with homogeneous agents to be single-agent solvable (i.e., no RC). However, the absence of any of these conditions does not necessarily lead to RC. Theorem 1 provides an insight into the separation of SAP and MAP for problems with homogeneous agents.

9.0.2 Towards an Upper Bound for Type-1 RC

When the causal graph is not traversable or there are causal loops in type-1 RC problems, we find that upper bounds on the k in Def. 6 can be provided. We first investigate when causal loops are present and show that the upper bound on k is associated with how the causal loops containing agent variable signatures (agent VSs) can be broken in the individual causal graph signature (i.e., ICGS). The observation is that certain edges in these loops can be removed when there is no need to update the associated agent VSs. In our Burglary problem, when there are two agents in $room1$ and $room2$, respectively, there is no need to $WalkThrough$ to change locations (to access the switch after stealing the diamond). Hence, the associated edges can be removed to break the loops as shown in Fig. 17.

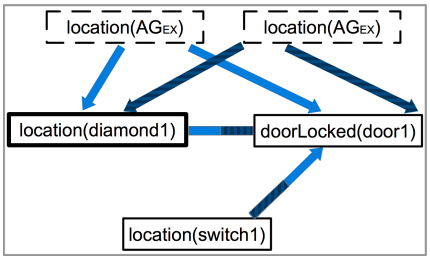

Figure 17. Causal loop breaking for the Burglary problem, in which the loop is broken by removing the edge marked with a triangle in previous figure. Two agent variable signatures (VSs) are introduced to replace the original agent VS.

Lemma 2 *Given a solvable MAP problem with homogeneous agents having traversable ICGSs, if all causal loops contain agent VSs and all the edges going in and out of agent VSs are directed, the minimum number of agents required is upper bounded by* $\Pi_{v \in CR(\Phi)} |D(v)|$ *(*$|D(v)|$ *denotes the size of the domain for variable* v*) when assuming that the agents can choose their initial states.*

$CR(\Phi)$ is created by: 1) adding the set of agent VSs in the causal loops to $CR(\Phi)$; 2) adding in any agent VS to $CR(\Phi)$ if there is a directed edge going into it from any variable in $CR(\Phi)$; 3) iterating 2 until no agent VSs can be added.

For each variable in $CR(\Phi)$, denoted by v, we introduce a set of variables $N = \{v_1, v_2, ..., v_{|D(v)|}\}$ to replace v. Any edges connecting v with other variables are duplicated on all variables in N, except for the directed edges that go into v. Each variable $v_i \in N$ has a domain with a single value; this value for each variable in N is different and chosen from $D(v)$. These new variables do not affect the traversability of the ICGS and all loops are broken.

From Theorem 1, we know that a virtual agent ϕ^+ that can assume the joint state specified by $CR(\Phi)$ can achieve the goal. We can simulate ϕ^+ using $\Pi_{v \in CR(\Phi)} |D(v)|$ agents as follows. We choose the agent initial states according to the permutations of states for $CR(\Phi)$, while choosing the same states for all the other agent VSs according

to ϕ^+. Given a plan for ϕ^+, we start from the first action. Given that any permutations of states for $CR(\Phi)$ is assumed by an agent, we can find an agent, denoted by ϕ, that can execute this action with the following three cases (we show that all three cases can be simulated):

1) If this action updates an agent VS in $CR(\Phi)$, we do not need to execute this action based on the following reasoning. Given that all edges going in and out of agent VSs are directed, we know that this action does not update V_o. (Otherwise, there must be an undirected edge connecting a variable in V_o to this agent VS. Similarly, we also know that this action does not update more than one agent VS.). As a result, it does not influence the execution of the next action.

2) If this action updates an agent VS that is not in $CR(\Phi)$, we know that this action cannot have variables in $CR(\Phi)$ as preconditions or prevail conditions, since otherwise this agent VS would be included in $CR(\Phi)$ given its construction process. Hence, all agents can execute the action to update this agent VS, given that all the agent VSs outside of $CR(\Phi)$ are always kept synchronized in the entire process (in order to simulate ϕ^+).

3) Otherwise, this action must be updating only V_o and we can execute the action on ϕ.

Following the above process for all the actions in ϕ^+'s plan to achieve the goal. Hence, the conclusion holds. The requirement on the traversability of ICGS in Lem. 2 is further relaxed below:

Corollary 2 *Given a solvable MAP problem with homogeneous agents, if all the edges going in and out of agent VSs are directed in the causal graphs, the minimum number of agents required is upper bounded by* $\Pi_{v \in VS(\Phi)}|D(v)|$*, assuming that the agents can choose their initial states.*

Given a valid plan π_{MAP} for the problem, we can solve the problem using $\Pi_{v \in VS(\Phi)}|D(v)|$ agents as follows: first, we choose the agent initial states according to the permutations of state for $VS(\Phi)$.

The process is similar to that in Lemma 2. We start from the first action. Given that all permutations of $VS(\Phi)$ are assumed by an agent, we can find an agent, denoted by ϕ, that can execute this action: if this action updates some agent VSs in $VS(\Phi)$, we do not need to execute this action; otherwise, the action must be updating only V_o and we can execute the action on ϕ.

Following the above process for all the actions in π_{MAP} to achieve the goal. Hence, the conclusion holds.

The bounds above are upper bounds. Nevertheless, for our Burglary problem, the assumptions for both are satisfied and the 2 is returned for both, which happens to be exactly the k for which the problem is minimally k-agent solvable. In future work, we plan to establish the tightness of these bounds.

Chapter 10

ANALYSIS OF TYPE-2 (HETEROGENEOUS) RC

For the class of problems with heterogeneous agents, the most obvious cause for required cooperation (RC) in type-2 RC problems is the requirement of capabilities from different agents (due to domain, variable and capability heterogeneity). In the logistics domain, for example, the domain of the location variable for a truck agent can be used to force the agent from visiting certain locations in a city (domain heterogeneity – DH). When there are packages that must be transferred between different locations within a city, at least one truck agent that can access each location is required (hence RC). In the rover domain, a rover that is equipped with a camera sensor would be associated with the agent variable $equipped\_for\_imaging(rover)$. When we need both $equipped\_for\_imaging(rover)$ and $equipped\_for\_rock\_analysis(rover)$, and no rovers are equipped with both sensors (variable heterogeneity – VH), we have RC. In the logistics domain, given that the truck cannot fly (capability heterogeneity – CH), when a package must be delivered from a city to a non-airport location of another city, at least a truck and a plane are required.

We note that 1) the presence of DH, VH or CH (i.e., the heterogeneity condition) in a solvable MAP problem does not always cause RC. In other words, a solvable MAP problem that satisfies the heterogeneity condition may not have RC (e.g., when the problem does not need the different capabilities); 2) the presence of the heterogeneity condition in a type-2 RC problem is not always the *sole cause* of RC. For example, the conditions that cause RC in type-1 problems may also cause RC in type-2 problems (see the mixed cause region in Fig. 14).

Due to the complexities above, to continue the analysis, we further divide MAP problems with heterogeneous agents into two subsets as shown in Fig. 14, which in turn divides type-2 RC problems into two subclasses – problems in which RC can only be caused by the heterogeneity of the agents (termed DVC RC), and the remaining problems. In other words, no causes for type-1 (homogeneous) RC are present in DVC RC problems. For DVC RC, we can improve the planning performance using a simple compilation.

10.0.1 DVC RC in Type-2 RC

In particular, we show that the notion of *transformer agent* allows us to significantly reduce the number of agents to be considered in planning for DVC RC problems, which can be solved first with a single or a small set of transformer agents. The transformer-agent plans can then be expanded to use agents in the original problems.

Definition 16 (DVC RC) *A DVC RC problem is an RC problem in which all agents have traversable causal graphs with no causal loops.*

DVC RC problems can be solved by the construction of transformer agents (defined below), which are *virtual* agents that combine all the domain values, variable signatures and action signatures of the agents in the original MAP problem (i.e., Φ). To ensure that this combination is valid, we make the following assumption: agent variables for different agents are positively (i.e., no negations in preconditions or prevail conditions and non-exclusively defined. Exclusively defined variables can be compiled away. Two variables are exclusively defined when associating them with the same agent can introduce conflicts or lead to undefined states. For example, $using\_gas(AG_{EX})$ and $using\_kerosene(AG_{EX})$ can lead to undefined states, if an agent can use either

gas or kerosene but the state in which both are used is undefined (i.e., the agent cannot be flying and driving at the same time). This issue can be compiled away, e.g., $using(AG_{EX}) = \{gas, kerosene\}$, potentially with a few complications to handle the transition between the values. Given a MAP problem $P = \langle V, \Phi, I, G \rangle$,

Definition 17 (Transformer Agent) *A transformer agent is an agent ϕ^* that satisfies:* 1) $\forall v \in V_\Phi$, $\exists v^* \in V_{\phi^*}$, $D(v^*) = D(V)$, *in which* $V = \{v | v \in V_\Phi$ *and* $VS(v^*) = VS(v)\}$. 2) $VS(\phi^*) = VS(\Phi)$. 3) $AS(\phi^*) = AS(\Phi)$.

An intuitive way to think about a transformer agent is that it is a single agent that can "transform" into any agent in the original MAP problem. A transformer agent can use any other agent's capabilities (but not simultaneously). Before discussing how the initial states of these transformer agents can be specified, we introduce a subset of DVC RC problems that can be solved by efficiently.

Connected DVC RC: A subset of DVC RC problems, referred to as *Connected DVC RC* (B.1.1 in Fig. 14), can be solved by a single transformer agent. To define *Connected DVC RC*, we first define *state space connectivity* for agents.

Definition 18 (State Space Connectivity) *Given two agents ϕ and ϕ' that have traversable causal graphs with no causal loops, denote their state spaces as S_ϕ and S'_ϕ, respectively, S_ϕ and S'_ϕ are connected if $\exists s \in S_\phi, \exists s' \in S'_\phi$, $VS(s) \cap VS(s') \neq \emptyset \wedge \forall v \in s_\cap, VS(s)[v] = VS(s')[v]$, in which $s_\cap = VS(s) \cap VS(s')$ and $VS(s)[v]$ denotes the value of variable v in state s.*

Intuitively, when two agents have connected state spaces, the transformer agent is allowed to transform from one agent to the other agent and vice versa in the shared states (i.e., s and s' above). This is necessary to ensure that a single transformer agent can traverse the state spaces of both agents. For example, the prerequisite for a

truck-plane agent to be able to deliver a package that is at the airport of a city to a non-airport location in another city is that the truck and plane agents in the original problem must be able to meet at the destination airport to transfer the package.

Definition 19 (Connectivity Graph) *In a DVC RC (or DVC MAP) problem, a connectivity graph is an undirected graph in which the nodes are the agents and any two nodes are connected if they have connected state spaces.*

Definition 20 (Connected DVC RC) *A connected DVC RC problem is a DVC RC problem in which the connectivity graph is a single connected component .*

The result of Theorem 1 can be extended below:

Lemma 3 *Given a connected DVC RC problem, it is solvable by a single transformer agent for any specification of its initial state.*

Given that the causal graphs are traversable and contain no causal loops for all agents in DVC RC, the only condition that can cause RC is the heterogeneity condition (i.e., DH, VH or CH). Given that the state spaces of agents are connected, a transformer agent can traverse the state spaces of all agents. Hence, the problem is solvable by this transformer agent based on Theorem 1.

Corollary 3 *A DVC RC problem in which all the goal variables lie in a single connected component in the connectivity graph can be solved by a single transformer agent, given a proper specification of the initial state.*

The initial state of the transformer agent only needs to lie within the connected component. We refer to problems that can be solved by a single transformer agent as *transformer-agent solvable* ($B.1.1$) in Fig. 14. Many problems in the IPC domains

belong to Connected DVC RC (e.g., logistics and rover domains) and are transformer-agent solvable.

Lemma 4 *Given a DVC RC problem $P = \langle V, \Phi, I, G \rangle$ in which all the goal variables lie in a single connected component in the connectivity graph, the single transformer-agent plan that solves this problem can be expanded to a multi-agent plan using Φ.*

The proof is by construction. Given the initial state of the transformer agent, from previous discussions, we know that the transformer agent "is assuming the form" of an agent in Φ for which the transformer agent is executing an action (i.e., the first action in the single transformer-agent plan). Given that this agent has a traversable causal graph with no causal loops, we can plan it to reach the current state of the transformer agent while keeping the values of variables in V_o, and then let it execute the first action. The same process continues until the last action of the transformer agent is executed, and the goal is achieved.

For example, in the logistics domain, suppose that we have a package to be delivered to a non-airport location in city c. The package is initially at the airport in city b, the plane agent is at the airport in city a, and the truck agent is at a non-airport location in city c. We solve this problem with a *truck-plane* agent initially at the airport in city b. This transformer agent can fly to c with the package, "transform" to the truck agent, drive to the non-airport location to deliver. To create the plan for the original problem, we need to first expand the single transformer-agent plan by sending the plane agent from city a to b. We then follow the transformer-agent plan until the package arrives at the airport in city c. Next, we expand the plan again by sending the truck agent to the airport in c to pick it up. The "transformation" forces the plane

agent to unload and the truck agent to load the package at the airport. The plan then follows through.

Corollary 4 *Given a DVC RC problem in which the connectivity graph is separated into connected components, the number of transformer agents to solve the problem is upper bounded by the number of connected components (assuming proper specifications of the initial states); the plan can be expanded to use agents in the original problem.*

For DVC RC problems with more than one connected component in the connectivity graph (B.1.2 in Fig. 14), we can similarly expand the multiple transformer-agent plans into plans for Φ in the original problem.

Hence, given a MAP problem with heterogeneous agents, we can first construct the causal graphs for all agents and execute algorithms to determine whether the causal graphs are traversable and loop free. If the problem is not determined to belong to DVC MAP, we can use any single-agent planner to solve the problem by considering agents as resources. Otherwise, we first create the connectivity graph. We do not need to construct the exact graph; a partial graph (with a subset of the edges) only increases our estimation of the number of transformer agents needed. When more time is allowed, we can continue completing the graph. At any time during this process, if the graph is determined to be connected, we can stop immediately, in which case we know that we have a connected DVC RC problem. When the upper bound is estimated from the graph, we can create a set of transformer agents accordingly to solve the problem. If a plan is not found, we know that the MAP problem is unsolvable; otherwise, we can then expand the plan into a plan for the original problem. A similar process can be used to implement a planner with homogeneous agents in which case the results from Lemma 2 and Corollary 2 can also be utilized.

Chapter 11

USING THE TRANSFORMER AGENT COMPILATION

We now turn our attention to performance in practice. Specifically, we show how the transformer agent compilation can be used to improve the planning performance. We compare with one of the best performing centralized planners, MAP-LAPKT (planner entry is SIW+-then-BFS(f)) Stolba, Komenda, and Kovacs 2015; Muise, Lipovetzky, and Ramirez 2015, in CoDMAP, MAP-LAPKT also uses a compilation approach in which a MAP problem is converted into a single-agent planing problem; an off-the-shelf planner is then used to solve it. MAP-LAPKT's performance is very close to the best MA planner in CoDMAP (which is ADP). We could not compare against ADP since their approach does not use compilation and is incorporated as a heuristic in the planning process.

We implemented a planner called RCPLAN. First, we determine whether the problem belongs to connected DVC MAP (e.g., whether the causal graph is traversable and contains no causal loops, and whether the connectivity graph is a single connected component, which are often determined by the domain). If it is, we compile the problem into a problem with a single transformer agent based on Def. 17. We then solve this new problem with an existing planner (i.e., FastDownward). Finally, we use Metric FF to expand the tranformer-agent plan to a plan for the original problem as explained in Lemma 4.

2*Problem Type	Coverage		IPC Agile Score (Time Score)		IPC Sat Score (Quality Score)		2*# Domains	2*# Agents
	RCPLAN	MAP-LAPKT	RCPLAN	MAP-LAPKT	RCPLAN	MAP-LAPKT		
CoDMAP Problems	**219 (98.6%)**	214 (96.4%)	**214.37**	186.73	187.31	**204.08**	11	2 - 20
Large Problems	**51 (98.1%)**	41 (78.8%)	**44.54**	34.90	**49.38**	38.81	3	2- 50

Table 1. Performance Comparison between RCPLAN and MAP-LAPKT

To remove the influence of the underlying planner, we replace the internal planner in MAP-LAPKT with our FastDownward (without optimizations). For CoDMAP competition domains, we use 11 out of 12 domains, since the Wireless domain does not satisfy our compilation criteria (i.e., the goals contain agent variables). Due to the additional expansion process, we expect RCPLAN to improve performance particularly over large problems (in terms of number of agents). Hence, besides the CoDMAP domains, we also generate larger problems for three CoDMAP domains (i.e., Rover, Blocksworld and Zenotravel) using standard IPC generators.

The results are listed in Table 1. Both RCPLAN and MAP-LAPKT are given 30min to solve each problem. We use the IPC Agile (for time) and IPC Sat (for plan quality) scores. [5] The experiments were run on a 3.0 GHz quad-core linux machine with 7GB memory. We can see that RCPLAN performs better than MAP-LAPKT in time performance (i.e., IPC Agile) consistently, although slightly worse in IPC Sat for CoDMAP problems. This performance improvement is mainly due to the reduction of instantiated actions in the compiled transformer agent problem. For example, for one of the Zenotravel problem with 6 agents, RCPLAN instantiated 9152 actions while the number of agent actions used in MAP-LAPKT is 54912. Our planner achieves a higher IPC Sat score for the larger problems due to the coverage. We can always post-process plans to improve quality Nakhost and Müller 2010. Our results confirm that many IPC domains (also chosen in CoDMAP as MAP domains) are in fact (single) transformer-agent solvable!

[5] For each problem: the IPC Agile score is $\frac{1}{(1+\log_{10}(T/T^*))}$ where T is the time taken by a given planner and T^* is the time taken by the fastest planner for that problem; similarly, the formula for IPC Sat score is $\frac{Q}{Q^*}$ where Q is the plan quality produced by a given planner and Q^* is the highest quality produced for the problem. For most domains, we use the inverse of the plan length as the quality. The final scores are the sum of scores for all problems.

Chapter 12

THESIS CONCLUSION

In this document, I have investigated two topics from the field of automated planning, namely model-lite planning models and multi-agent planning. In the For model-lite planning models, I chose to investigate APDDL and the related application of robust planning. For this model, I proposed an extension (named C-PISA) to the existing robust planning approach (PISA) to allow the use of external information (available through cases) during planning. This extension also provides the method a way of overcoming one of its main drawbacks, namely the need for a domain writer to identify the annotations. Finally we also took a look at possible way we can utilize APDDL and robust planning for practical application and possible extensions to the existing approach.

In the second section of the document, we introduced the notion of required cooperation (RC). First, we showed that directly querying for RC is intractable. As a result, we started the analysis with a class of more restrictive problems where agents are homogeneous. We identified an exhaustive set of causes of RC for this class and provided bounds for the number of agents required in different problem settings.For the remaining problems where agents are heterogeneous, we showed that a subclass of problems in which RC is only caused by the heterogeneity of the agents (i.e., DH, VH or CH) can be solved with a smaller number of transformer agents than with the agents in the original problems. This RC analysis makes theoretical contributions to the understanding of multi-agent planning problems, informs the design of future multi-agent planning competitions, and presents practical applications,

e.g., determining how many agents to be assigned to each given task when agent resources are limited. We implemented a planner using our theoretical results and compared it with one of the best IPC CoDMAP planners. Results show that our planner improved performance on most IPC CoDMAP domains,which incidentally implies that only a subset of MAP problems were covered (i.e., Connected DVC RC) in this competition.

REFERENCES

Amir, Eyal, and Barbara Engelhardt. 2003. "Factored planning." In *IJCAI,* 929–935.

Bacchus, Fahiem, and Qiang Yang. 1993. "Downward Refinement and the Efficiency of Hierarchical Problem Solving." *AIJ* 71:43–100.

Backstrom, Christer, and Bernhard Nebel. 1996. "Complexity Results for SAS+ Planning." *Computational Intelligence* 11:625–655.

Brafman, Ronen I. 2006. "Factored Planning: How, When, and When Not." In *AAAI,* 809–814.

Brafman, Ronen I., and Carmel Domshlak. 2013. "On the complexity of planning for agent teams and its implications for single agent planning." *AIJ* 198:52–71.

Bylander, Tom. 1991. "Complexity Results for Planning." In *IJCAI,* 1:274–279. Sydney, New South Wales, Australia.

Cushing, William, Subbarao Kambhampati, Mausam, and Daniel S. Weld. 2007. "When is Temporal Planning Really Temporal?" In *IJCAI,* 1852–1859.

Cushing, William, Daniel S. Weld, Subbarao Kambhampati, Mausam, and Kartik Talamadupula. 2007. "Evaluating Temporal Planning Domains." In *ICAPS.*

Geffner, Hector, and Blai Bonet. 2013. "A concise introduction to models and methods for automated planning." *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8 (1): 1–141.

Helmert, Malte. 2006. "The Fast Downward Planning System." *JAIR* 26:191–246.

Jonsson, A., and M. Rovatsos. 2011. "Scaling Up Multiagent Planning: A Best-Response Approach." In *ICAPS,* 114–121. June.

Kambhampati, Subbarao. 2007. "Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models." In *Proceedings of the National Conference on Artificial Intelligence,* 22:1601. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Knight, S, Gregg Rabideau, Steve Chien, Barbara Engelhardt, and Rob Sherwood. 2001. "Casper: Space exploration through continuous planning." *IEEE Intelligent Systems* 16 (5): 70–75.

Knoblock, Craig. 1994. "Automatically Generating Abstractions for Planning." *AIJ* 68:243–302.

Konidaris, George, Leslie Pack Kaelbling, and Tomas Lozano-Perez. 2014. "Constructing symbolic representations for high-level planning."

———. 2015. "Symbol acquisition for probabilistic high-level planning." *Image (a1, Z0)* 1:Z0.

Kvarnstrom, Jonas. 2011. "Planning for Loosely Coupled Agents Using Partial Order Forward-Chaining." In *ICAPS.*

Mcdermott, D., M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. *PDDL - The Planning Domain Definition Language.* Technical report TR-98-003. Yale Center for Computational Vision and Control,

Muise, Christian, Nir Lipovetzky, and Miquel Ramirez. 2015. "MAP-LAPKT: Omnipotent Multi-Agent Planning via Compilation to Classical Planning." *(CoDMAP-15):* 14.

Nakhost, H., and M. Müller. 2010. "Action elimination and plan neighborhood graph search: Two algorithms for plan improvement." In *ICAPS,* 121–128. http://www.scopus.com/inward/record.url?eid=2-s2.0-78650595490&partnerID=40&md5=6f11c6eb5ff82702f878545bdd5fea08.

Nguyen, T, and Subbarao Kambhampati. 2014. "A heuristic approach to planning with incomplete strips action models." In *International Conference on Automated Planning and Scheduling.*

Nguyen, Tuan A, Subbarao Kambhampati, and Minh Do. 2013. "Synthesizing robust plans under incomplete domain models." In *Advances in Neural Information Processing Systems,* 2472–2480.

Pape, Claude Le. 1990. "A combination of centralized and distributed methods for multi-agent planning and scheduling." In *ICRA.*

Stolba, Michal, Antonin Komenda, and Daniel Laszlo Kovacs. 2015. *Competition of Distributed and Multiagent Planners (CoDMAP).* http://agents.fel.cvut.cz/codmap/results-summer.

Tian, Xin, Hankz Hankui Zhuo, and Subbarao Kambhampati. 2015. "Discovering Underlying Plans Based on Distributed Representations of Actions." *arXiv preprint arXiv:1511.05662.*

Weld, Daniel, and Oren Etzioni. 1994. “The first law of robotics (a call to arms).” In *AAAI,* 94:1042–1047. 1994.

Wikipedia. 2016. *Planning Domain Definition Language — Wikipedia, The Free Encyclopedia.* [Online; accessed 4-June-2016]. https://en.wikipedia.org/w/index.php?title=Planning_Domain_Definition_Language&oldid=707868255.

Yang, Qiang, Kangheng Wu, and Yunfei Jiang. 2007. “Learning action models from plan examples using weighted MAX-SAT.” *Artificial Intelligence* 171 (2): 107–143.

Zhang, Yu, Sarath Sreedharan, and Subbarao Kambhampati. 2015. “Capability Models and Their Applications in Planning.” In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems,* 1151–1159. International Foundation for Autonomous Agents and Multiagent Systems.

———. 2016. “A Formal Analysis of Required Cooperation in Multi-agent Planning.” In *International Conference on Automated Planning and Scheduling.*

Zhuo, Hankz Hankui, Subbarao Kambhampati, and Tuan Nguyen. 2012. “Model-lite case-based planning.” *arXiv preprint arXiv:1207.6713.*

Zhuo, Hankz Hankui, Tuan Nguyen, and Subbarao Kambhampati. 2013. “Refining incomplete planning domain models through plan traces.” In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence,* 2451–2457. AAAI Press.

APPENDIX A

NOTES ON PART B

Please note that the part B of this document contains parts taken from the paper Zhang, Sreedharan, and Kambhampati 2016, which was presented at ICAPS 2016. This paper was co-authored with Dr. Yu Zhang and Prof. Subbarao Kambhampati who were the Co-Chairs of my defense committee and approves of the inclusion of the above-stated work in this thesis.