Domain Concretization from Examples:

Addressing Missing Domain Knowledge via Robust Planning

by

Akshay Sharma

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

Approved November 2020 by the
Graduate Supervisory Committee:

Yu Zhang, Chair
Georgios Fainekos
Siddharth Srivastava

ARIZONA STATE UNIVERSITY

December 2020

ABSTRACT

Most planning agents assume complete knowledge of the domain, which may not be the case in scenarios where certain domain knowledge is missing. This problem could be due to design flaws or arise from domain ramifications or qualifications. In such cases, planning algorithms could produce highly undesirable behaviors. Planning with incomplete domain knowledge is more challenging than partial observability in the sense that the planning agent is unaware of the existence of such knowledge, in contrast to it being just unobservable or partially observable. That is the difference between known unknowns and unknown unknowns.

In this thesis, I introduce and formulate this as the problem of *domain concretization*, which is inverse to domain abstraction studied extensively before. Furthermore, I present a solution that starts from the incomplete domain model provided to the agent by the designer and uses teacher traces from human users to determine the candidate model set under a minimalistic model assumption. A robust plan is then generated for the maximum probability of success under the set of candidate models. In addition to a standard search formulation in the model-space, I propose a sample-based search method and also an online version of it to improve search time. The solution presented has been evaluated on various International Planning Competition domains where incompleteness was introduced by deleting certain predicates from the complete domain model. The solution is also tested in a robot simulation domain to illustrate its effectiveness in handling incomplete domain knowledge. The results show that the plan generated by the algorithm increases the plan success rate without impacting action cost too much.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor, Dr. Yu "Tony" Zhang, for his guidance and constant support throughout my journey at ASU. Talking to him always gave me clarity of thoughts and made me feel motivated during my hard times. I am thankful to him for giving me the opportunity to work with him and encouraging me to enter this beautiful world of research. I would like to thank my thesis committee members Dr. Georgios Fainekos and Dr. Siddharth Srivastava for their invaluable feedback and time. I would also like to thank the members of the Cooperative Robotic Systems (CRS) lab and my friends for their advice, feedback and support.

And most importantly, I would like to thank my parents, my grandparents, my brother and my entire family for their constant love, support, motivation and faith in me. Without them, this would never have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Most planning agents rely on complete knowledge of the domain, which could be catastrophic when certain knowledge is missing in the domain model. The fact that the agent is unaware of some domain knowledge makes this problem different and more challenging than partial observability where the agent knows what is missing. That is the difference between known unknowns and unknown unknowns. For example, missing certain state features means that the agent would perceive all states as if those features do not exist. In such cases, traditional planning algorithms can create the same plan under very different scenarios. For similar reasons, standard RL (Sutton and Barto (2018)), IRL (Ng and Russell (2000a); Ziebart *et al.* (2008)), and intention recognition algorithms (Mao and Gratch (2004); Schrempf and Hanebeck (2005)) would be easily misled as a result of incomplete domain knowledge.

This problem is referred to as the problem of ***domain concretization***, which is inverse to the problem of domain abstraction. This thesis focuses on state abstraction and leaves action abstraction in future work. Even though state abstractions in planning have significant computational advantages, if not engineered properly, they can lead to unsound and incomplete domain specifications (Marthi *et al.* (2007); Srivastava *et al.* (2016)). Incomplete domain specification also arises naturally from domain ramifications (Finger (1987)) and qualifications (McCarthy (1977); Ginsberg and Smith (1988)) which may be viewed as intentional state abstractions.

Consider a manufacturing domain where a robot is tasked to deliver a model to a human worker, which is produced by a 3D printer. The human performs some demonstrations to teach the robot about how this task is supposed to be done. Depending on how long the

model has cooled down before the delivery, the robot is expected to either apply a coolant first or deliver it immediately to the human worker. However, if the temperature is an unknown domain feature to the robot, it may result in immediate delivery regardless of the temperature, which could lead to safety risks.

In this thesis, the domain concretization problem is restricted to deterministic domains. This problem is formulated using a STRIPS-like language (Fikes and Nilsson (1971)) and methods for searching candidate models are provided. The search algorithm generates candidate models based on the incomplete model provided by the domain designer and teacher traces from human users. This problem is first formulated as a search problem in the model space. To expedite the search, the thesis then presents a sample-based search method by considering only models that are consistent with the traces. Additionally, an online version that is more practical and has a computational advantage is presented. This is because the online version uses only one trace in each iteration and maintains a small set of models for future refinements.

For planning, the algorithm generates a robust plan that achieves the goal under the maximum number of candidate models. This is done by transforming the planning problem into a Conformant Probabilistic Planning problem (Domshlak and Hoffmann (2007)). The algorithm has been tested on various IPC domains and a simulated robotics domain where incompleteness was introduced by removing specific predicates from the complete domain model. Results show that the robust plan generated by the algorithm increases the plan success rate under the complete model without impacting the cost much.

Chapter 2

RELATED WORK

2.1 Domain Abstractions

The ongoing research on state abstraction has established its necessity and computational advantages. There exists work where authors developed abstractions that retained properties of the ground domain. The authors in Abel *et al.* (2016) have studied abstractions that produce optimal behaviors similar to those with the ground domain. In Srivastava *et al.* (2016), the authors have investigated and categorized several abstraction mechanisms that retain the properties of the ground domain like the Markov property. The fact that the agent's domain model is missing some domain features in this work makes the model an abstraction of the ground model. Domain concretization is the inverse of domain abstraction: instead of searching for an abstraction, here the algorithm tries to recover the ground domain model from the abstract (incomplete) model.

2.2 Partial Observability

It may be attempting to solve the domain concretization problem using a POMDP formulation (Kaelbling *et al.* (1998)), or reinforcement learning methods with POMDPs (Jaakkola *et al.* (1999)), where the state is partially observable and the agent uses observations to update its belief about the state of the world. Note however that a POMDP still requires complete knowledge about the ground state space, which is not available in the problem formulation due to missing domain knowledge. More specifically, this means that, neither the space of belief states nor the observation function is known, unlike that in POMDP. Hence, POMDP solutions cannot be used to solve this problem.

2.3 Model-lite Planning

The agent's model in this problem could be considered an approximate domain model and hence planning in such a case becomes a type of model-lite planning (Kambhampati (2007)). Many existing approaches have considered planning in such approximate (or incomplete) domain models. Authors in Weber and Bryce (2011); Nguyen *et al.* (2017) introduce planning systems that can generate plans for an incomplete domain where actions could be missing some preconditions or effects. The problem of domain concretization can be viewed as a more general problem where the information of possible missing knowledge (e.g., possible preconditions) is missing.

2.4 Learning Action Models from Traces

The idea of learning action model using plan traces has been there for quite some time. While some authors have considered refining incomplete domain models (Zhuo *et al.* (2013a,b)), others have focused more on learning action models from scratch (Yang *et al.* (2007); Zhuo *et al.* (2010)). Authors in Zhuo and Yang (2014) have used transfer learning to learn the action model using a small amount of training data. In Zhuo and Kambhampati (2013), authors have gone one step further by developing an algorithm that can learn action models even when the traces are noisy. One common assumption in all these methods is the complete knowledge of the state space, about which, the agent in this problem, does not have.

Chapter 3

DOMAIN CONCRETIZATION

3.1 Problem Analysis

Consider the motivating example of the manufacturing domain mentioned in the previous section. The robot is expected to deliver a 3D model to a human, produced by a 3D printer. The human teacher performs some demonstrations to teach the robot how the task is supposed to be done. The human has the complete domain model M^* but the robot's domain model is incomplete $\widetilde{M}$, where the temperature is an unknown feature. In such a case the traces (demonstrations) are generated by the teacher using M^* and *projected* onto $\widetilde{M}$ that is observed by the robot. In the projection, the robot does not (or cannot) observe the missing feature since it is not present in its domain model (i.e., the robot would consider it irrelevant to the task scenario and/or lacks the appropriate sensor for feature extraction). Before further discussion let us first list the assumptions:

1. The human teacher is rational and the traces are optimal in the complete domain model M^*.
2. The complete domain model M^* is deterministic.
3. The robot still knows about all the possible actions.
4. The unknown features are *completely* missing from the domain model.
5. The unknown features do not appear in the goal, since the goal is often provided by the human user who is also the teacher.

Given the assumptions mentioned above, an observed (projected) trace would still contain all the actions in the teacher's trace but the state trajectory associated with it will be viewed differently. To make this clearer, let us have a look at an example trace. Consider the following teacher trace/plan that is observed by the robot in our motivating example:

- Plan z_1: there is a *hot* 3D model (that is just printed) in the start state. The action sequence is: $\langle$ `apply_coolant`, `deliver` $\rangle$.

In the teacher's perspective, the state sequence for this action sequence is: $\langle$ {`hot`}, {`wet`}, {`wet, delivered`} $\rangle$, assuming that `apply_coolant` has two effects, one being making the model `wet`, and the other being removing `hot`.

The robot observes the action and state sequences above except for the `hot` feature, which requires the `apply_coolant` action to be first applied in the complete model. Since the feature is completely missing in the robot's model, the `apply_coolant` action would also be missing it (i.e., the robot is unaware that this action would remove `hot` from the state or, in other words, make the model cooler). Hence, the state sequence that is observed by the robot would be: $\langle$ {}, {`wet`}, {`wet, delivered`} $\rangle$. Hence, it would consider that the `apply_coolant` action is unnecessary, assuming that the action `deliver` does not require the model to be `wet`.

Hence, given the observed initial state {}, in the robot's model $\widetilde{M}$, the optimal plan for this "same" scenario is (note again that the robot does not observe the `hot` feature in the initial state):

- Plan z_0: there is a 3D model in the start state. The action sequence is: $\langle$ `deliver` $\rangle$.

Obviously, there is an inconsistency here: with the assumption that the teacher is rational and cost captured by plan length, the teacher is expected to never choose anything longer than z_0. The solution to the problem of *domain concretization* is hence hinged on addressing this type of inconsistency, which we refer to as *plan inconsistency*. Let us analyze

this further to understand how trace inconsistency can help us concretize our domain models and its limitations. Consider a teacher's plan z_1 (under the complete model M^*). Denote the robot's optimal plan (under the incomplete model $\widetilde{M}$) for the same observed/projected initial ($\widetilde{s_0}$) and goal states (g) as z_0. Two possibilities are here:

1. Cost(z_1) $\neq$ Cost(z_0) – Inconsistency: the teacher has chosen a more or less costly plan than the optimal plan in the robot's model, which should not occur if the robot's model is complete. In fact, given our assumptions above, the only possibility here is that Cost(z_1) $>$ Cost(z_0). And that is because when a feature is completely missing in the domain model, it will be missing as both preconditions and effects. Hence, the missing features would only make the incomplete model less constrained to satisfy the goal.

2. Cost(z_1) $=$ Cost(z_0) – Here, there are two possibilities:

   (a) z_1 is a valid plan for $(\widetilde{s_0}, g)$ in $\widetilde{M}$: Undetermined (deemed consistent): z_1 could be a valid plan in an incomplete model $\widetilde{M}$ or it could be that $\widetilde{M}$ is complete. Since we cannot decide which case is true here, we will have to wait until inconsistencies are detected.

   (b) z_1 is not a valid plan for $(\widetilde{s_0}, g)$ in $\widetilde{M}$ – Inconsistency: z_1 is not a valid plan in the robot's model but a valid plan in the complete model.

In our problem solution, we address only the cases when any plan inconsistencies above are detected. However, even when all the plan inconsistencies are addressed, it does not mean that the complete model is recovered due to the case of 2.(a) above. On the other hand, as more teacher traces are provided, our solution is expected detect more inconsistencies as long as the complete model is not recovered.

3.2 Background

Given the focus on deterministic domains, the problem is defined using a STRIPS-like language. Here, a planning problem is defined by a triplet $P = \langle s_0, g, M \rangle$, where s_0 is the start state, g is the set of goal prepositions that must be **T** (true) in the goal state and M is the domain model. The domain model $M = \langle O, R \rangle$ where R is the set of predicates and O is the set of operators. The set of prepositions F and the set of actions A, are generated by instantiating all the predicates in R and all the operators in O respectively. Hence, M can also be defined as $M = \langle A, F \rangle$. A state is either the set of prepositions $s \subseteq F$ that are **T** or $s_\perp = \{\perp\}$. Since $\perp \notin F$, it is a dead state and once $s_\perp$ is reached goal (g) can never be achieved. The actions change the current state by adding or deleting some prepositions. Each action a $\in A$ is specified by a set of preconditions $Pre(a)$, a set of add effects $Add(a)$ and a set of delete effects $Del(a)$, where $Pre(a)$, $Add(a)$ and $Del(a) \subseteq F$.

For a model M, the resulting state after execution of plan π in state s is determined by the transition function γ, which is defined as follows:

$$\gamma(\pi, s) = \begin{cases} s & \text{if } \pi = \langle\rangle; \\ \gamma(\langle a \rangle, \gamma(\pi', s)) & \pi = \pi' \circ \langle a \rangle. \end{cases} \tag{3.1}$$

In this problem the *Generous Execution (GE)* semantics are used as defined in Nguyen *et al.* (2017). According to GE semantics, if an action a is not executable, it does not change the world state s. Hence, the transition function γ for an action sequence with a single action $a \in A$ and state s in a model M under GE semantics is defined as follows:

$$\gamma(\langle a \rangle, s) = \begin{cases} (s \setminus Del(a)) \cup Add(a) & \text{if } Pre(a) \subseteq s; \\ s & \text{otherwise.} \end{cases} \tag{3.2}$$

A plan π is a valid plan for a problem $P = \langle s_0, g, M \rangle$ iff $\gamma(\pi, s_0) \supseteq g$. The cost of a plan π is the cumulative cost of all the actions in that plan. To make it simpler, in this work

it is assumed that all the actions have the same unit cost. If the cost of all the actions is the same then the cost of a plan will simply be equal to the plan length. One may also note that the proposed solution will still work even if the cost for each action is different.

3.3 Motivating Example

Complete domain model (denoted by M^*): The specification of a packing domain is shown in Fig. 3.1 to motivate the problem. The domain has 4 operators. As the name suggests, `open_box` is used to open boxes where items are to be placed. The `grasp` operator is used to grasp an item. The actions `place` and `stack` are used to put items into boxes. `place` is used when the box is empty and `stack` is used when the box already contains some items. The goal is to store all items in boxes using the minimum number of boxes. Some of the items may be fragile. For these items, they must not be stacked on. To avoid such situations, a predicate `not_fragile` is introduced foritems that are not fragile.

Incomplete domain model (denoted by $\widetilde{M}$): In the robot's domain model that is incomplete, the incompleteness may be due to missing the predicate `not_fragile` (shown in bold in Fig. 3.1). In such a case, the robot may choose a plan in which it stacks an item over a fragile item which is highly undesirable.

3.4 Problem Formulation

The incomplete model $\widetilde{M}$ (the robot's model in Fig. 3.1) is defined as $\widetilde{M} = \langle \widetilde{O}, \widetilde{R} \rangle$. The incomplete model is incomplete in the sense that it is missing some predicates from the ground truth domain $M^* = \langle O^*, R^* \rangle$. In the packing domain (Fig. 3.1), the robot's model is missing the predicate of `not_fragile`. Denote $\widehat{R}$ as the set of missing predicates, such that:

- $\widetilde{R} \subseteq R^*$ such that $\widetilde{R} = R^* \setminus \widehat{R}$

```
(:action open_box
:parameters (?b - box)
:precondition (and (handempty))
:effect (and (box_open ?b)))


(:action grasp
:parameters (?i - item)
:precondition (and (on_shelf ?i) (handempty))
:effect (and (holding ?i) (not (on_shelf ?i)) (not (handempty))))


(:action place
:parameters (?i1 - item ?b - box)
:precondition (and (holding ?i1) (box_open ?b) (box_empty ?b))
:effect (and (item_packed ?i1) (handempty) (on_top ?i1 ?b)
      (not (holding ?i1)) (not (box_empty ?b))))


(:action stack
:parameters (?i1 - item ?i2 - item ?b - box)
:precondition (and (holding ?i1) (not_fragile ?i2)
      (box_open ?b) (on_top ?i2 ?b))
:effect (and (item_packed ?i1) (handempty) (on_top ?i1 ?b)
     (not (on_top ?i2 ?b)) (not (holding ?i1))))
```

Figure 3.1: Domain Model of the Packing Domain

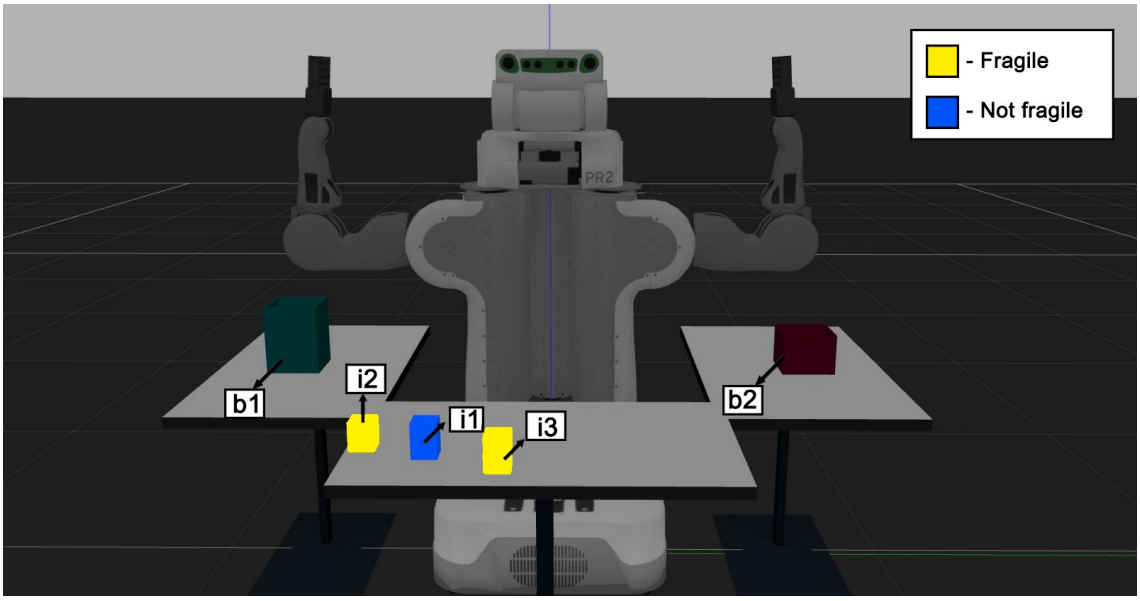

Figure 3.2: Simulation View of the Packing Domain

- $\widetilde{Pre(o)} = Pre(o) \setminus \widehat{R}$
- $\widetilde{Add(o)} = Add(o) \setminus \widehat{R}$
- $\widetilde{Del(o)} = Del(o) \setminus \widehat{R}$

Definition 1. *The problem setting of **Domain Concretization** is defined as a setting where the agent has only access to an incomplete domain model $\widetilde{M}$ and teacher traces under M^*.*

In this thesis, it is assumed that the human user knows M^* so he/she can provide a set of successful teacher traces ζ^* based on M^*. Each trace $z^* \in \zeta^*$ is a tuple $\langle s_0^*, g, \tau \rangle$ where g and s_0^* are the goal and initial state respectively and τ is an action sequence $\langle a_1, a_2, ...a_n \rangle$ where each $a_i \in A^*$. Since the robot has *incomplete predicate set* $\widetilde{R}$, it observes *incomplete traces* $\widetilde{\zeta}$ = $\langle \widetilde{s_0}, g, \tau \rangle$, where $\widetilde{s_0} = s_0^* \setminus \widehat{R}$. It is assumed that the robot still has knowledge about all the actions, and hence τ would not be affected by the incompleteness. Additionally, it is also assumed that the missing predicate cannot be present in the goal state since the goal is provided by the human user.

Definition 2. *The problem of Planning under Incomplete Domain Knowledge (PIDK) is defined as* $\widetilde{P} = \langle \widetilde{s_0}, g, \widetilde{M}, \widetilde{\zeta} \rangle$*, which is the problem of generating a plan that has the highest probability of success for* $P^* = \langle s_0^*, g, M^* \rangle$.

To achieve this, using $\widetilde{\zeta}$ and $\widetilde{M}$, the proposed solution finds a set of candidate models $\mathcal{M}$. Once $\mathcal{M}$ is generated, the planning problem is compiled to a Conformant Probabilistic Planning problem, and a conformant plan is found that has the highest probability of success under the set of candidate models $\mathcal{M}$. The details of the method are provided in following sections.

3.5 Candidate Model Generation

When searching for candidate models, the method makes the assumption that the candidate models have the minimum number of new predicates (i.e., new state features) and the minimum number of model changes. Note that a new predicate could be introduced to different actions as an additional precondition or effect in the domain. One of the motivations for this assumption could be attributed to the principle of *Occam's Razor* (Blumer *et al.* (1987)). Another reason is due to the fact that here, the model-space search problem has infinite possible solutions as more dummy predicates can always be introduced. This is somewhat similar to the unidentifiability problem in reward learning (Ng and Russell (2000b); Amin and Singh (2016); Armstrong and Mindermann (2017)) and is referred to it the problem of *model unidentifiability*.

The problem of generating candidate models is transformed to a search problem in the model space. A variable σ is defined whose value indicates the number of new predicates that are going to be added. Initially, σ is set to 1 and is gradually increased if no models are found for the current value. Further, A set X is defined as the set of possible typed predicates for each new predicate. Each state in the search space is a domain model (M) generated by adding one or more predicates from X to one or more possible missing po-

sitions in the incomplete domain model ($\widetilde{M}$). Since these predicates could also be present in the start state, each candidate model M is verified against one or multiple possible start states s_0 (details in the search section), based on the prepositions added to $\widetilde{s_0}$. The model M is accepted as a candidate model only if it passes the model test below for at-least one s_0. The model-space search is defined as follows:

- **Initial State:** $\widetilde{M}$ (The given incomplete model).
- **Action Set (Λ):** $\alpha_\chi^{Pre(o)}$ (or $\alpha_\chi^{Add(o)}/\alpha_\chi^{Del(o)}$) $\in \Lambda$,

  $$\forall \chi \in X \text{ and } \forall o \in O \text{ where } O \in M.$$

  The actions in the model space search represent a predicate χ being added to the $Pre(o)$ (or $Add(o)/Del(o)$) of the current model M.
- **Successor Function (T):** $T(M, \alpha_\chi^{Pre(o)}) = M'$.

  T produces new model M' where $R' = R \cup \chi$ and $Pre(o') = Pre(o) \cup \chi$ where $o' \in O'$ and $o \in O$. Similarly, $T(M, \alpha_\chi^{Add(o)})$ and $T(M, \alpha_\chi^{Del(o)})$ can be defined.
- **Model (Goal) Test:** $C_1(M) \wedge C_2(M) \wedge C_3(M)$.

  $C_1(M)$, $C_2(M)$ and $C_3(M)$ are defined as follows-

  - *Plan Validity Test*, $C_1(M)$ is True if :

    $$\gamma^M(\tau, s_0) \supseteq g; \forall \langle \widetilde{s_0}, g, \tau \rangle \in \widetilde{\zeta} \tag{3.3}$$

    This ensures that the traces are executable and achieve the goal under M. This condition essentially detects the type of inconsistency mentioned in case 2.(b) of the problem analysis section (3.1).

– *Well-Justification Test*, $C_2(M)$ is True if :

$$\forall a_i \in \tau, \gamma^M(\tau \setminus \{a_i\}, s_0) \not\supseteq g; \forall \langle \widetilde{s_0}, g, \tau \rangle \in \widetilde{\zeta} \tag{3.4}$$

This ensures that the traces are well-justified (Fink and Yang (1997)) in M, which means that if any action is removed from the trace the goal will not be achieved. This is a direct consequence of the assumption that the traces are optimal. This condition detects the type of inconsistency mentioned in case 1 of the problem analysis section (3.1).

– *Plan Optimality Test*, $C_3(M)$ is True if :

$$Cost(\pi^M) = Cost(\tau); \forall \langle \widetilde{s_0}, g, \tau \rangle \in \widetilde{\zeta} \tag{3.5}$$

where π^M is the optimal plan for problem $P = \langle s_0,\ g,\ M \rangle$; $Cost(\pi^M)$ and $Cost(\tau)$ represents plan cost of π^M and τ respectively. This condition ensures that the traces are optimal under M. This condition detects the type of inconsistency mentioned in case 1 of the problem analysis section (3.1).

The failure of any of these conditions signifies that the traces are *inconsistent*. The notion of inconsistency has been explained in section 3.1.

In the packing domain in Fig. 3.1, let τ = $\langle$`open_box b1`, `grasp i1`, `place i1 b1`, `grasp i2`, `stack i2 i1 b1`, `open_box b2`, `grasp i3`, `place i3 b2`$\rangle$. Applying the algorithm to τ and $\widetilde{M}$, C_3 will fail. This is because the optimal plan π^M = $\langle$`open_box b1`, `grasp i1`, `place i1 b1`, `grasp i2`, `stack i2 i1 b1`, `grasp i3`, `stack i3 i2 b1`$\rangle$ is shorter (and hence less costly) than τ. This indicates that the trace is inconsistent as mentioned in the case 1 of the problem analysis (3.1) section.

Theorem 1: Conditions $C_1(M)$ and $C_3(M)$ are necessary and sufficient to ensure that the model M can generate τ, $\forall \langle \widetilde{s_0}, g, \tau \rangle \in \widetilde{\zeta}$.

Proof: If $C_1(M)$ is true then $\gamma^M(\tau, s_0) \supseteq g$ which means τ is a valid plan for $\langle s_0, g, M\rangle$. Also, if $C_3(M)$ is true then cost of optimal plan of M equals cost of τ which implies τ has minimum cost under M. Hence, by definition τ is the optimal plan for the given problem under model M. Since this is true for all τ, it can be concluded that M can generate $\tau\ \forall\langle\widetilde{s_0}, g, \tau\rangle \in \widetilde{\zeta}$. Similarly, if a model M can generate τ for a problem $\langle s_0, g, M\rangle$ then τ an optimal plan for the problem. Hence, by definition of optimal plan, τ is valid and has minimum cost which implies that it satisfies $C_1(M)$ and $C_3(M)$. This proves that the model test is sound and complete.

Theorem 2: $C_2(M)$, is a necessary condition for a trace τ where $\langle\widetilde{s_0}, g, \tau\rangle \in \widetilde{\zeta}$ to be optimal.

Proof: This could be proven by contradiction. Assume that model M doesn't satisfy $C_2(M)$ but still τ is an optimal plan for M. This implies that for some action a_i, $\tau \setminus \{a_i\}$ is a valid plan. Hence, there is a valid plan $\tau \setminus \{a_i\}$ which has lesser cost than τ as $Cost(\tau \setminus \{a_i\}) = Cost(\tau) - Cost(a_i)$. This implies that τ is not an optimal plan which contradicts the initial assumption. Hence proved.

Generating an optimal plan for a problem is a known challenging problem. Since $C_3(M)$ involves the generation of an optimal plan for model M, the algorithm would become very slow if it checks for an optimal plan for every model. To improve this the proposed algorithm checks for $C_3(M)$ only if $C_2(M)$ fails. This reduces the search time by a significant amount. Moreover, Theorem 2 ensures that this process will not lose any candidate models.

Search: This formulation can be solved by any standard search algorithm and here a uniform cost search is chosen. The cost of the path from M to M' is the number of changes introduced into the domain model M to generate M'. The search starts with $\widetilde{M}$ (the given incomplete model) as the initial state for the model-space search. Using the action set Λ and the transition function T, the set of models for the subsequent steps is generated and

put into a priority queue. The model with the least cost is then popped and passed through the model test. The model is tested with multiple s_0's for each trace. If X' ($X' \subset X$) is the set of new predicates added to the current domain model M, the algorithm generates U as a set of all the prepositions that can be instantiated from each predicate in X'. Now, $s_0 = \widetilde{s_0} \cup \mu$ where $\mu \in \mathfrak{P}(U)$. Each μ is chosen such that $|\mu|$ is minimum. First, the algorithm tests the model with $|\mu|$ = 0 and if the model test is passed it does not go further. If model test fails, the algorithm increases $|\mu|$ by 1 and tests with every μ such that $|\mu| = 1$. This process goes on until a predefined threshold is reached and in that case, the model test fails. Setting this threshold to 0 denotes an assumption that missing predicate could never be present in the start state.

Now, if the model test is passed, all the models having the same cost are popped and tested. Each model that passes the model test becomes a candidate model. If the model test fails, the set of models for the subsequent steps is generated in a similar way as mentioned before. This process continues until some model passes the model test or the cost increases to be above a predefined threshold. In the latter case, σ is increased by 1 and the whole search starts again. In the end, sets of models are obtained such that, within each set of models, s_0's used to satisfy the model test remains the same. Each model could be in multiple sets based on the s_0's that satisfy the model test. The candidate model set $\mathcal{M}$ is a weighted union of these sets of models based on how many times each model appears in those sets.

In the example of packing domain shown in Fig 3.1, the search starts with $\widetilde{M}$ as model M and σ = 1 (1 missing predicate). Denote the missing predicate as `pred_1`. Since M fails $C_3(M)$, the search generates X that includes all the possible typed predicates like `(pred_1 ?b - box)`, `(pred_1 ?b - box ?i - item)` etc. Using X, the set Λ is generated that has all the possibilities where the predicates in X can be added, like $\alpha_{\chi}^{Pre(open\_box)}$ $\forall \chi \in X$, which means χ will be added to the precondition of operator

`open_box`. Using current model M, T and $\alpha \in \Lambda$ next model M' is generated. Now, the model M' is tested for the goal conditions and based on the result of the goal test the search continues as explained above.

3.6 Generating Candidate Models using Sample-Based Search

One obvious way to perform the model search is by brute-forcing through all the possible models, which is computationally expensive and not scalable as it considers all the possibilities where the given predicate could be added. Hence, a sample-based search is presented which reduces the search space by a considerable amount and makes the search faster. The idea is to return information for refining the model to satisfy the traces only, instead of checking all possibilities. This means that instead of considering all the possible actions in the action set, the algorithm chooses a subset of actions that could possibly lead to a valid model. Since these changes are necessary to make the model pass the model test, changes that would not lead to a candidate model are ignored.

The action set Λ' now is a set of actions that each encodes multiple simultaneous changes. Such a process should also decide what s_0's to use for a model M in the model test. The search process is similar as before except that if a model fails the model test, instead of returning false, it returns a set of actions, $B \subseteq \Lambda'$ that will be used to generate models for the next step. Instead of checking the model M for all possible s_0's, the algorithm checks for the ones that are returned along with the action set. The returned action set B is as follows:

- *Unsatisfied Precondition*: This happens when the trace is not executable in the current model M because of some unsatisfied precondition.

  If for a model M, $C_1(M)$ is false and, for an action $a_i \in \tau$, $1 \leq i \leq |\tau|$, $Pre(a_i) \nsubseteq \gamma^M(\langle a_1, a_2, ...a_{i-1}\rangle, s_0)$. Then, the returned action set $B = \{\{\alpha_\chi^{Add(o_j)}\}\}$, $\forall a_j \in \tau$

and $\forall \chi \in X$, where o_j is the operator corresponding to action a_j and $1 \leq j \leq i-1$. Also, possible additions to start state s_0 will be $Pre(a_i) \setminus \gamma^M(\langle a_1, a_2, ...a_{i-1} \rangle, s_0)$. This is because missing preposition could either be present in the add effect of any of the previous actions or in the start state. This addresses the type of trace inconsistency mentioned in case 2.(b) of the problem analysis section (3.1).

- *Un-justified action*: This happens when some action in the trace is not well-justified in the current model M.

  If for a model M, $C_2(M)$ is false and for some action a_i, $\gamma^M(\tau \setminus \{a_i\}, s_0) \supseteq g$. Then, the returned action set $B = \{\{\alpha_\chi^{Add(o_i)}, \alpha_\chi^{Pre(o_j)}\}\} \cup \{\{\alpha_\chi^{Add(o_i)}, \alpha_\chi^{Pre(o_j)}, \alpha_\chi^{Del(o_j)}\}\}$ $\forall a_j \in \tau$ and $\forall \chi \in X$. o_i and o_j are operators corresponding to actions a_i and a_j respectively and $i+1 \leq j \leq |\tau|$. Intuitively, this generates B such that a_i cannot be removed from τ which makes it well-justified under M. The set in B represents that these changes will be introduced simultaneously in the model M to obtain M'. This addresses the type of trace inconsistency mentioned in case 1 of the problem analysis section (3.1).

- *Sub-optimal Trace*: This happens when the optimal plan of the current model M is shorter than the trace.

  In this case, $C_3(M)$ becomes false, which means there has to be some action that was not possible in τ under M^* but was possible in π^M under M. Hence, the operator corresponding to that action is missing some precondition. In such a case, $\exists a_i, a'_i$ such that $a_i \in \tau$ and $a'_i \in \pi^M$ and $a_i \neq a'_i$, $1 \leq i \leq n$ $(n = |\pi^M|)$. Then, the returned action set $B = \{\{\alpha_\chi^{Pre(o_j)}\}\} \cup \{\{\alpha_\chi^{Pre(o_j)}, \alpha_\chi^{Del(o_j)}\}\}$ $\forall a'_j \in \pi^M$, where o_j is the operator corresponding to a'_j and $i \leq j \leq n$. This addresses the type of trace inconsistency mentioned in case 1 of problem analysis section (3.1).

The three conditions mentioned above augment the model such that the traces are no

longer *inconsistent*. The notion of the trace inconsistency has been discussed in depth in the problem analysis section (3.1).

For the packing domain in Fig. 3.1, consider τ = ⟨`open_box b1`, `grasp i1`, `place i1 b1`⟩. If some model M is missing the predicate `(box_open ?b)`, $C_2(M)$ will fail because the goal will be achieved even after the deletion of `open_box b1` from τ. In that case, $B = \{\{\alpha_\chi^{Add(open\_box)}, \alpha_\chi^{Pre(grasp)}\},\{\alpha_\chi^{Add(open\_box)}, \alpha_\chi^{Pre(place)}\}\} \cup$
$\{\{\alpha_\chi^{Add(open\_box)}, \alpha_\chi^{Pre(grasp)}, \alpha_\chi^{Del(grasp)}\}, \{\alpha_\chi^{Add(open\_box)}, \alpha_\chi^{Pre(place)}, \alpha_\chi^{Del(place)}\}\}\ \forall \chi \in X.$

Using the actions in B for each model-space state the successive states are generated in the search and it continues as mentioned in the previous section.

Theorem 3: (*Soundness*) The candidate models found by the sample-based search process can generate all the given traces, with the minimum number of changes to the incomplete model $\widetilde{M}$.

Proof: This is pretty straightforward as while generating the action set B in the sample-based search method, the process checks to see if the conditions ($C_1(M)$, $C_2(M)$ and $C_3(M)$) are satisfied. It accepts the model as a candidate model only if these are satisfied. Using Theorem 1, it can be said that if the sample-based search process finds a candidate model, the candidate model will be able to generate all the given traces. Uniform cost search ensures that the changes between the candidate models and the given incomplete domain model $\widetilde{M}$ are minimum. Hence Proved.

Theorem 4: *(Completeness)* The sample-based search finds all the models satisfying the model test with the minimum number of changes to the incomplete model $\widetilde{M}$.

Proof: We prove this by induction. Let us start from the first iteration through all the traces. In this iteration, since the unknown features are completely missing from the incomplete domain model $\widetilde{M}$, it makes the model a less constrained model than the complete model M for achieving any goal. Hence, a trace will always be achieving the goal in $\widetilde{M}$ when it works under the complete model. This means $C_1(M)$ will always be satisfied. For

any trace, $C_2(M)$ can still be false and in that case, it means that there is some action a_i that is not well-justified under $\widetilde{M}$. To make that action well-justified, it is necessary that a_i has some add effect that is a precondition for one of the subsequent actions in the trace. All these possibilities are included in the set B discussed above. [1] These changes are necessary to ensure that model M satisfies the conditions.

Similarly, for any trace, $C_3(M)$ can also be false and in that case, the model $\widetilde{M}$ must have an optimal plan with shorter (lesser cost) than the trace. In that case, to satisfy $C_3(M)$, this optimal plan must be inexecutable in the complete model. This is necessarily achieved by updating $\widetilde{M}$ to add new preconditions to actions following the action a_i up to which the trace and this optimal plan are the same. Here also, B contains all the possibilities. Since a delete effect may also appear as a precondition in the same action, each option in B that includes the addition of a new predicate to the preconditions comes with a paired option that includes the addition of the new predicate to the delete effects as well.

Now let us assume that for the first k iterations, all necessary changes to $\widetilde{M}$ with all possibilities are considered. For the $(k+1)^{th}$ iteration, since in the previous iterations we have added some new preconditions, the current (intermediate) model M may no longer be less constrained than the complete model for achieving a goal. Hence, for any trace, it might not be achieving its goal in M, which means that $C_1(M)$ may fail. In this case, it is necessary that there is some action a_i for which one of the preconditions is not satisfied. In this case, to make the trace executable, the only possibilities are to add the predicate (for the unsatisfied precondition) either to the add affects of some action before a_i in the trace, or to the start state. Again we can see that B includes all the possibilities in which predicate could be added to make the model satisfy this condition for each trace. For the $(k+1)^{th}$ iteration we can use similar reasoning as in the first iteration to argue that B includes all

[1] Note that in B, these possibilities are organized according to each trace and a new model is created for addressing each trace for the next iteration.

the possibilities for $C_2(M)$ and $C_3(M)$ as well.

Hence, by induction, we show that the sample-based search includes all the possible ways in which the given incomplete model can be modified to satisfy all the given traces. The Uniform Cost Search (UCS) ensures that the changes are minimal. Hence, the sample-based search process will find all the candidate models with the minimum number of changes.

3.7 Generating Candidate Models using Online Search

In the real world, it is desirable to have an online search method that takes traces into account as they arrive. The search procedure is similar to the sample-based method mentioned above and starts with the incomplete model $\widetilde{M}$. The difference being that the model test is performed against just one trace at a time. Within each iteration, the search continues till it finds $\mathcal{M}$. Once $\mathcal{M}$ is generated, it begins the next iteration by adding models in $\mathcal{M}$ (from the previous iteration) to the priority queue and then starting the search process again. It continues iterating until all available traces are checked. The set of models $\mathcal{M}$ generated after the last iteration becomes the final set of candidate models.

In terms of computation, on average, the online sample-based search is expected to perform better. This is because in each iteration it uses only those models that satisfy the traces in the previous iterations. This restricts the number of models to be checked in each iteration since it is only constrained by one trace at any time. It also makes the online method highly dependent on the sequence of traces and not guaranteed to find the complete model. This is because it could find some incorrect model M_1, higher in the search-tree and at the same level reject some model M_2 that would have lead to the correct model by introducing a few more changes (deeper in the tree). In that iteration, the online search would not consider models that are deeper than M_1. Since in the next iteration only M_1 will be considered, there is a possibility that the correct model would never be found. The

online method, in that case, doesn't return any solution. The best way to deal with such a situation is to randomize the traces and perform multiple training iterations until a set of candidate models that satisfy all the traces is found.

3.8 Planning under Incomplete Domain Knowledge

After generating the set of candidate models $\mathcal{M}$, the algorithm finds a robust plan for $\widetilde{P} = \langle \widetilde{s_0},\ g,\ \widetilde{M}, \widetilde{\zeta} \rangle$ such that it has the highest probability of achieving the goal under the weighted set of candidate models $\mathcal{M}$. Similar to Nguyen *et al.* (2017), the problem of generating a robust plan is compiled into a Conformant Probabilistic Problem (CPP). A Conformant Probabilistic Problem (Domshlak and Hoffmann (2007)) is defined as $P' = \langle I,\ g,\ D,\ \rho \rangle$ where I is the belief over the initial state, g is set of propositions that needs to be **T** for goal state, D is the domain model and ρ is the acceptable goal satisfaction probability. The domain model $D = \langle F',\ A' \rangle$, where F' is the set of prepositions and A' is the set of actions. Each $a' \in A'$ has the set of preconditions $Pre(a') \subseteq F'$ and $E(a')$, the set of conditional effects. Each $e' \in E(a')$ is a pair of $con(e')$ and $\dot{o}(e')$ where $con(e') \subseteq F'$ which enables e' and $\dot{o}(e')$ is a set of outcomes ϵ. The outcome ϵ is a triplet $\langle Pr(\epsilon),\ add(\epsilon),\ del(\epsilon) \rangle$ where $add(\epsilon)$ adds prepositions to and $del(\epsilon)$ deletes prepositions from the current model with probability $Pr(\epsilon)$.

A compilation that translates the original planning problem $\widetilde{P}$ to a conformant probabilistic planning problem P' is defined as follows:

- For each candidate model $M_i \in \mathcal{M}$ a preposition m_i is introduced. Let the set of these prepositions be $\widehat{M}$. Further, a set $\widehat{F} = \bigcup_{i=1}^{n} F_i$ is introduced, where F_i is the set of prepositions instantiated by predicates $R_i \in M_i$ where $M_i \in \mathcal{M}$ and n is the total number of model in $\mathcal{M}$. For the compiled problem set of prepositions $F' = \widehat{M} \cup \widehat{F}$.

- For each model $M_i \in \mathcal{M}$, U_i' is created which is the set of new prepositions that were

not present in $\widetilde{M}$. Using this, now a domain model is created D from $\widetilde{M}$ as follows:

- A new action a'_0 that initializes the start state with new/missing prepositions is introduced. The action a'_0 is defined as $Pre(a'_0) = \emptyset$. $\forall m_i \in \widehat{M}$, a conditional effect $e'_i \in E(a'_0)$ is created such that $con(e'_i) = \{m_i\}$ and each outcome $\epsilon_j \in \dot{o}(e'_i)$ has $add(\epsilon_j) = \mu'$ and $del(\epsilon_j) = \emptyset$ where $\mu' \in \mathfrak{P}(U'_i)$. For each outcome, $Pr(\epsilon_j) = 1/|\mathfrak{P}(U'_i)|$. This essentially initializes the start state for each model M_i, considering all the possibilities for new prepositions to be present in it with equal probability. In the packing domain, if $U'_i = \{$ `(pred_0 i1)`, `(pred_0 i2)`$\}$, all the possibilities are $\{\}$, $\{$`(pred_0 i1)`$\}$, $\{$`(pred_0 i1)`$\}$, $\{$`(pred_0 i1)`, `(pred_0 i2)`$\}$. All of these will be considered to be present in the start state with equal probability for each, which is 0.25 here.
- For each action $a \in \widetilde{A}$ in $\widetilde{M}$, if model $M_i \in \mathcal{M}$ adds a preposition $u'_i \in U'_i$ to $\widetilde{Pre(a)}$, a conditional rule $e_i \in E(a)$ is created, such that $con(e_i) = \widetilde{Pre(a)} \cup \{m_i\} \cup \{u'_i\}$, $add(e_i) = \widetilde{Add(a)}$ and $del(e_i) = \widetilde{Del(a)}$, $m_i \in \widehat{M}$ is the binary predicate corresponding to M_i. For example, if in model M_i, action `place i1` has the new preposition `(pred_0 i1)` in it's precondition, then the action in the compiled domain will have a conditional rule where $con(e_i) = \widetilde{Pre(\texttt{place i1})} \cup \{m_i\} \cup \{$`(pred_0 i1)`$\}$ and $add(e_i)$ and $del(e_i)$ will remain the same.
- For each action $a \in \widetilde{A}$ in $\widetilde{M}$, if model $M_i \in \mathcal{M}$ adds a preposition $u'_i \in U'_i$ to $\widetilde{Add(a)}$, a conditional rule $e_i \in E(a)$ is created, such that $con(e_i) = \widetilde{Pre(a)} \cup \{m_i\}$, $add(e_i) = \widetilde{Add(a)} \cup \{u'_i\}$, $del(e_i) = \widetilde{Del(a)}$, $m_i \in \widehat{M}$, is the binary predicate corresponding to M_i.
- For each action $a \in \widetilde{A}$ in $\widetilde{M}$, if model $M_i \in \mathcal{M}$ adds a preposition $u'_i \in U'_i$ to $\widetilde{Del(a)}$, a conditional rule $e_i \in E(a)$ is created, such that $con(e_i) = \widetilde{Pre(a)} \cup$

$\{m_i\}$, $add(e_i) = \widetilde{Add(a)}$, $del(e_i) = \widetilde{Del(a)} \cup \{u_i'\}$, $m_i \in \widehat{M}$, is the binary predicate corresponding to M_i.

The modified domain becomes D for the problem P'.

- The initial belief state $I = (\bigwedge_{f \in \widetilde{s_0}} f) \wedge (oneof(m_1, m_2, ...m_n))$ where $oneof$ returns **T** when exactly one of its input is **T**. The probability of each m_i is equal to its weight in $\mathcal{M}$ and all the other prepositions are certain.

- In the compiled problem, $\rho' = \rho$ represents the probability of success of the conformant plan generated in the problem $\widetilde{P}$.

The process of generating the robust plan is quite straightforward. Initially, ρ = 1, and a conformant plan is calculated for the given problem using conformant probabilistic planner. If a conformant plan is not found ρ is decreased by Δ until one is found. The plan so obtained is a robust plan for problem $\widetilde{P} = \langle \widetilde{s_0}, g, \widetilde{M}, \widetilde{\zeta} \rangle$ and has the highest probability of success given the weighted set of candidate models $\mathcal{M}$.

Theorem 5: If $\pi' = \langle a_0', a_1', a_2', a_3', ...a_n' \rangle$ is a plan for the complied problem P' with goal probability ρ then, ρ is also the probability of success of the plan $\pi = \langle a_1, a_2, a_3, ...a_n \rangle$ in the problem $\widetilde{P}$.

Proof: The compilation defines a bijective mapping between each state $i_j \in I$ of the problem P' and each model $M_j \in \mathcal{M}$ of the problem $\widetilde{P}$. Also, the probability of i_j in I is same as the probability of M_j in $\mathcal{M}$. Let the belief state after execution of action a_0' in I be $b_{s_0'}$. The action a_0' initializes multiple start states for each model in $\mathcal{M}$. One can think of it as generating multiple sub-models for each $M_j \in \mathcal{M}$ based on different start states. Let the set of all these sub-models be $\mathcal{M}'$. It is easy to see that there is a bijective mapping between each $s_{0j}' \in b_{s_0'}$ and model $M_j' \in \mathcal{M}'$ with same probability as well. Moreover, if s_{0j} is the start state for model $M_j' \in \mathcal{M}'$, any prepositions $p \in s_{0j}'$

iff $p \in s_{0j}$. The application of plan $\pi' \setminus \{a'_0\}$ in belief state $b_{s'_0}$ generates sequence of belief states $\langle b_{s'_1}, b_{s'_2}, ...b_{s'_n}\rangle$. Similarly, executing π in s_{0j} for model M'_j generates state sequence $\langle s_{1j}, s_{2j}, ...s_{nj}\rangle$. To any state $s'_{ij} \in b_{s'_i}$, every action $a' \in A'$ adds/deletes same set of prepositions against the same conditions as action $a \in A$ to s_{ij} for $M'_j \in \mathcal{M}'$. Hence, using induction one can say that for any state in the sequences mentioned above preposition $p \in s'_{kj}$ iff $p \in s_{kj}$ $(\forall k \in \{1, 2, ...n\})$. Therefore, $s_{nj} \supseteq g$ iff $s'_{nj} \supseteq g$. Since the all actions (except the dummy action a'_0) are deterministic, if plan π' achieves goal for $s'_{0j} \in b_{s'_0}$, then π achieves goal for s_{0j} in $M'_j \in \mathcal{M}'$. Hence, if π' achieves goal with probability ρ in P', then the probability of success of π in the problem $\widetilde{P}$ is ρ. Hence Proved.

Chapter 4

EVALUATION

The evaluations are conducted using two experiments. In the first one, the algorithm was tested on various International Planning Competition (IPC) like domains. In the second experiment, a more complex version of the motivating packing domain was simulated to show the practical benefits of the proposed algorithm. To generate optimal plans Fast-Downward planner (Helmert (2006)) was used and for solving Conformant Probabilistic Planning problem state of the art Probabilistic-FF (PFF) planner (Domshlak and Hoffmann (2007)) was used.

4.1 Synthetic Domains

For synthetic evaluations the following International Planning Competition (IPC) domains were used.

Rovers- In the rovers domain, there are multiple rovers each equipped with different capabilities like sampling soil, sampling rock and capturing the image. The goal in this domain is to sample soil, rock and capture images for the given waypoints.

Gold-miner- In this domain, there is a grid and the task is to pick up gold from a particular cell in the grid and deposit it in some other cell. Some cells have a laser or a bomb that could be used to clear blocked cells.

Storage- In this domain, there are some crates, containers, hoists, depots and different types of areas. Each hoist can move in different areas of the depot according to the connections specified by the spatial map. The objective is to move a certain number of crates from some containers to some depots by hoists.

Driverlog- In this domain, there are some drivers, trucks, locations and objects (pack-

ages). The driver can either walk or drive the truck. Walking paths are different from driving paths and the possible traversals of both are given in the beginning. The packages can be loaded into or unloaded from the truck (driver is not needed for this). The goal is to transport packages from one location to another and end the truck/trucks and the driver/drivers at specified locations.

Blocksworld- In this domain, there are multiple blocks on a table. The objective is to stack some or all of them in a specific order. The blocks could already be stacked in the beginning in an incorrect or partially correct order. In that case, one might have to unstack the blocks, put them on the table and stack again in the required order.

In all the above-mentioned domains, the incompleteness was introduced by deleting some predicates that could be generated by one action and are preconditions to some other action. For example in the rovers domain, to capture an image the precondition is that camera should be calibrated. In the gold-miner domain, incompleteness was introduced in a similar way as before, i.e. by removing the predicates, e.g. hold laser that would be needed to fire a laser.

For each domain, a few incomplete domains and problems were created by deleting either a single or multiple (up to 3) predicates. For this experiment, only those predicates were deleted, that were not present in the initial state. Using the complete domain, optimal plans were generated which were used as the teacher traces. These traces were then projected onto the incomplete model and were given to the agent. The robust plan generated by the algorithm was then verified by checking its execution against the complete domain.

Table 4.1 shows the detailed results of the experiments. For each incomplete domain the results are presented for all the 3 methods: Brute Force (BF), Sample-based Search (SS) and Online sample-based Search (OS). Here each incomplete domain was given sufficient traces such that the robust plan generated in the end was successful for every test case. This happens when there are very few candidate models. "-" in the table represents the

situation where the model search time exceeded the predefined limit for that domain. Since the brute-force approach is very expensive, it timed-out in almost every domain tested when multiple predicates were missing. It can clearly be seen that sample-based search (either SS or OS) reduces the number of models to be searched by a considerable amount. One can also see that as the number of missing predicates increases, the time taken by the algorithm to generate candidate models increases by a huge amount which is due to the exponential increase in the search space of the possible models. In such cases, one can see that the online search method performed much better than the sample-based search method except for cases where only 1 predicate was missing. It can also be seen that without using the proposed method and running the planner on the incomplete domain, the plan almost always fails. One can also see that the plan generated by the proposed algorithm has a little more cost than the optimal plan in the complete domain.

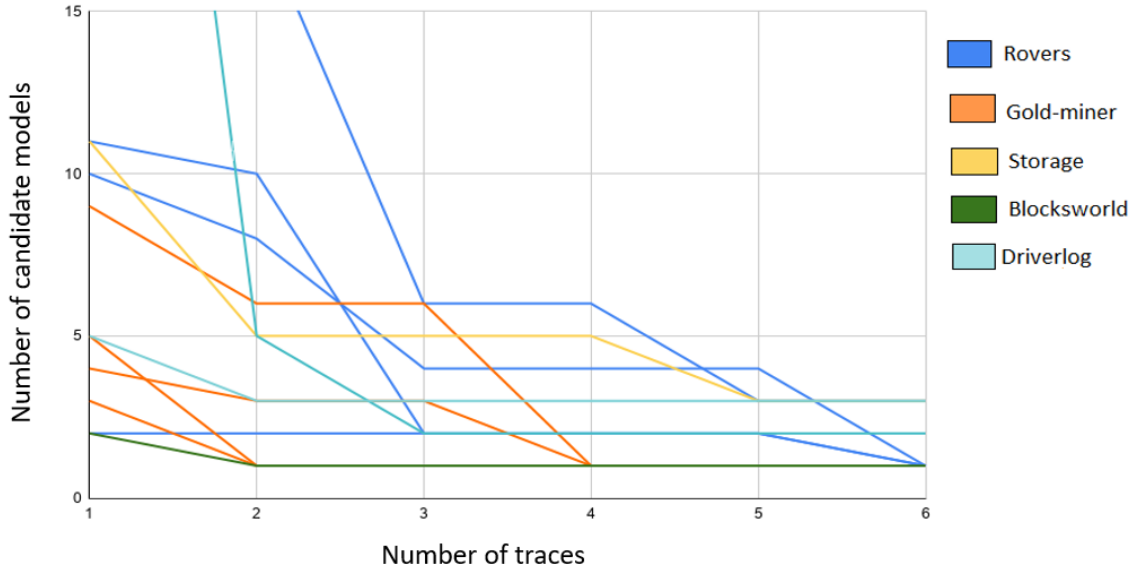

Figure 4.1: Candidate Models VS Number of Traces

Fig. 4.1 shows the variation of the number of candidate models generated with the number of traces. It can clearly be seen that as the number of traces increase, there are fewer but more accurate candidate models generated.

Doms	# T	Model Searched			Candidate Models			Time(secs)			Plan Success			Avg Cost Inc	
		BF	SS	OS	BF	SS	OS	BF	SS	OS	SS	OS	Default	SS	OS
One predicate missing at a time															
Rovers	3	1500	400	400	2	2	2	205.78	23.32	37.80	8/8	8/8	0/8	+0.25	+0.25
	3	1500	200	200	2	2	2	47.62	10.56	10.98	8/8	8/8	0/8	+0.63	+0.63
	5	10300	1400	300	3	3	3	520.29	103.89	184.52	11/11	11/11	0/11	+0.64	+0.64
Miner	2	700	30	30	1	1	1	8.74	0.90	1.12	12/12	12/12	0/12	+1.25	+1.25
	4	2520	430	70	1	1	1	517.69	156.14	31.07	10/10	10/10	0/10	+1.00	+1.00
	2	140	10	10	1	1	1	2.42	0.49	0.62	12/12	12/12	0/12	+1.25	+1.25
Storage	4	4350	580	220	3	3	3	128.15	31.11	23.12	3/3	3/3	0/3	+2.00	+2.00
Blocksworld	2	2530	280	240	1	1	1	21.79	5.09	7.33	5/5	5/5	0/5	+10.80	+10.80
Driverlog	3	1160	180	320	2	2	2	77.24	10.56	17.8	3/3	3/3	0/3	+2.66	+2.66
	3	38420	4280	920	3	3	3	864.45	278.6	56.12	7/7	7/7	0/7	+5.00	+5.00
Two predicates missing at a time															
Rovers	3	-	26500	26600	-	4	4	-	645.11	484.96	7/7	7/7	0/7	+0.57	+0.57
	6	-	297000	31900	-	3	3	-	8873.6	1484.72	8/8	8/8	0/8	+0.25	+0.25
	6	-	250700	32000	-	3	3	-	8456.40	1671.37	8/8	8/8	0/8	+0.00	+0.00
Miner	4	-	21900	3600	-	1	1	-	1023.41	192.62	10/10	10/10	0/10	+1.00	+1.00
	4	-	6600	620	-	1	1	-	450.05	41.72	10/10	10/10	0/10	+1.00	+1.00
	2	121300	260	260	1	1	1	1610.92	3.29	2.83	12/12	12/12	0/12	+1.25	+1.25
Three predicates missing at a time															
Rovers	6	-	-	117500	-	-	3	-	-	25610.00	-	8/8	0/8	-	+0.25
Miner	4	-	451500	11420	-	1	1	-	11104.00	364.22	10/10	10/10	0/10	+1.00	+1.00

Table 4.1: Detailed Results of Synthetic Domains

4.2 Simulated Domain

In this experiment, a simulated robotics domain was created which is a more complex version of the motivating packing domain. In this domain instead of one now there are two constraints for putting items in boxes. As before, the first constraint is that a fragile item cannot be stacked. The second constraint is that a fragile item cannot be dropped into the box and instead it has to be placed carefully. Now, there are two grasping actions: `horizontal_grasp` and `vertical_grasp`. For `horizontal_grasp`, the surroundings of the item to be picked up should be clear. For `vertical_grasp`, clear surroundings are not a necessity but in situations where the item to be picked is placed in

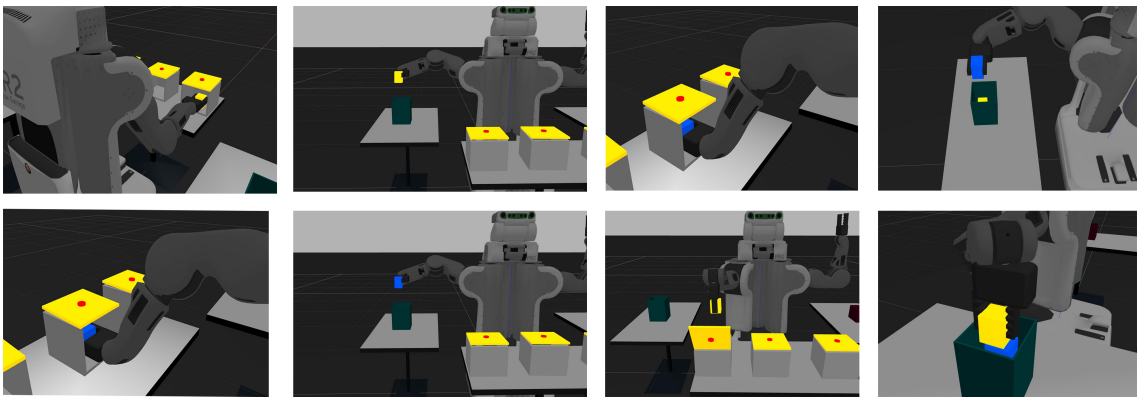

Figure 4.2: Comparison of Action Sequences Generated with a Standard Planning Method (Top) and the Proposed Method (Bottom).

the container (as shown in Fig. 4.2), the container needs to be opened first by pressing a button on the top. On the other hand for `horizontal_grasp`, this is not needed. Furthermore, if the robot has picked up an item horizontally it is constrained to use `drop` action instead of `place`. For vertical grasp both `place` and `drop` are possible. The goal of the robot is the same as before, to put all the items into the boxes by using the minimum number of boxes.

Fig. 4.2 shows the setup of the simulated experiments. The upper sequence (left to right) is the one that did not use the proposed algorithm. It can be seen that the robot was not able to distinguish between fragile item (in yellow) and non-fragile item (in blue). Hence, it used `horizontal_grasp` for picking up the fragile item and `drop` to put the fragile item into the box which could damage the fragile item. Furthermore, in the subsequent actions the robot stacked an item over the fragile item, which is also undesirable. On the other hand, in the bottom sequence showing the actions executed using the proposed algorithms (SS and OS), the robot first picked up the non-fragile item using `horizontal_grasp` and then put it into the box using the `drop` action. Then it used `vertical_grasp` followed by `place` to stack the fragile item carefully over non-fragile

item. This was a successful sequence as none of the constraints of the complete domain were violated.

The proposed algorithms were tested on few more scenarios where using the proposed algorithm the robot was able to learn what types of items are not fragile. The robust plan generated by the proposed algorithm was successful for all the test scenarios. This experiment showed that the robot was able to learn what types of items were not fragile and acted accordingly, with such knowledge completely missing in the domain initially.

Chapter 5

DISCUSSION AND CONCLUSION

In this thesis, I have formally introduced the problem of *Domain Concretization* and have discussed its prevalence (Sharma *et al.* (2020)). Further, a solution has been presented that uses teacher traces and the incomplete domain model to generate a set of candidate models and then finds a robust plan that achieves the goal under the maximum number of candidate models. I have formulated the model search process and developed a sample-based search to make the search more efficient. For practical use, an online version of this search method has been presented where the algorithm used one trace at a time to refine the candidate models. The proposed methods were tested on IPC domains and a simulated robotics domain where the proposed methods significantly increased plan success rate without increasing the cost much.

In this thesis, it is assumed that the human has more knowledge than the robotic agent in a planning and decision making context. In such cases, the human can either play as a teacher (Schaal (1996); Silver *et al.* (2012); Ng and Russell (2000a)) or critic (Christiano *et al.* (2017); Wirth *et al.* (2016)). The knowledge could be about the reward, domain model, or computational model. My work here is focused on the knowledge of domain model. Along this line, there has been work on eliciting domain models directly from humans (Talamadupula *et al.* (2010a,b); Talamadupula and Kambhampati (2013); Manikonda *et al.* (2014)). My work may be viewed in the general direction of learning from demonstrations for domain models, closely connecting to work on learning or refining domain models from teacher traces (Zhuo *et al.* (2013a)).

There has been work in the other direction where the robot has more knowledge than the human. In this direction, the focus has so far been on either the domain (Chakraborti

et al. (2017); Zhang *et al.* (2017)) or computational model (Choudhury *et al.* (2019)), while assuming humans have the correct reward model. In human-robot interaction, this is because the robot is assumed to be assisting humans and hence must respect their reward model. Regarding domain models, Zakershahrak *et al.* (2020a,b); Gong (2018); Gong and Zhang (2018) have developed a solution where the robot generates explanations or uses signaling to modify the human's domain model and make it more complete. Some authors have also looked at interactive scenarios where the robot interacts with the human to not only maintain a behavioral model of its human teammate to project the team status, but also to be aware that its human teammate's expectation of itself (Zakershahrak and Zhang (2018)). In a similar type of interactive scenario, some authors have considered the possibility of the human having an incorrect understanding of the robot's domain model, while the robot tries to learn the human preferences from the human feedbacks (Gong and Zhang (2020)).

In the reality, however, either the human or the robot may have more knowledge than the other. In such a case, it would be necessary to integrate these above work to allow knowledge discrepancy in both directions. Research in this direction is so far missing.

Chapter 6

FUTURE WORK

This problem opens up several research directions and few of them are mentioned here. The process of candidate model generation, even with sample-based and online methods, is quite computationally expensive, especially when the number of missing predicates increases. Finding more efficient heuristics and constraints would be an interesting future direction to work on. Another possible future direction could be to consider solving the problem in a better iterative process something similar to the online method. In the online method presented here, if an intermediate step learns only incorrect models and not any correct model then the next steps fail to learn the correct model. Future work could include solving this problem and enabling the algorithm to learn the correct model even if some previous steps learn only incorrect domain models.

The current work is limited to deterministic domains. Solving the problem of domain concretization for stochastic domains could be another possible future direction. Furthermore, currently, we assume that the human model is complete. Considering the incomplete human model with a complete robot's model where the robot's model could be used to refine the human model could be a very interesting future direction to work on. This could be further extended to the interactive scenarios where the human and the robot works as a team, both having incomplete domain models.

REFERENCES

Abel, D., D. E. Hershkowitz and M. L. Littman, "Near optimal behavior via approximate state abstraction", in "Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48", ICML'16, p. 2915–2923 (JMLR.org, 2016).

Amin, K. and S. Singh, "Towards resolving unidentifiability in inverse reinforcement learning", arXiv:1601.06569 (2016).

Armstrong, S. and S. Mindermann, "Occam's razor is insufficient to infer the preferences of irrational agents", arXiv:1712.05812 (2017).

Blumer, A., A. Ehrenfeucht, D. Haussler and M. K. Warmuth, "Occam's razor", Information Processing Letters **24**, 6, 377 – 380, URL http://www.sciencedirect.com/science/article/pii/0020019087901141 (1987).

Chakraborti, T., S. Sreedharan, Y. Zhang and S. Kambhampati, "Plan explanations as model reconciliation: Moving beyond explanation as soliloquy", in "Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17", pp. 156–163 (2017), URL https://doi.org/10.24963/ijcai.2017/23.

Choudhury, R., G. Swamy, D. Hadfield-Menell and A. D. Dragan, "On the utility of model learning in hri", in "2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)", pp. 317–325 (2019).

Christiano, P. F., J. Leike, T. Brown, M. Martic, S. Legg and D. Amodei, "Deep reinforcement learning from human preferences", in "Advances in Neural Information Processing Systems", vol. 30, pp. 4299–4307 (Curran Associates, Inc., 2017), URL https://proceedings.neurips.cc/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

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

Fikes, R. E. and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving", Artificial Intelligence **2**, 3, 189 – 208, URL http://www.sciencedirect.com/science/article/pii/0004370271900105 (1971).

Finger, J. J., *Exploiting Constraints in Design Synthesis*, Ph.D. thesis, Stanford, CA, USA (1987).

Fink, E. and Q. Yang, "Formalizing plan justifications", URL https://kilthub.cmu.edu/articles/Formalizing_Plan_Justifications/6605831 (1997).

Ginsberg, M. L. and D. E. Smith, "Reasoning about action ii: The qualification problem", Artif. Intell. **35**, 3, 311–342, URL https://doi.org/10.1016/0004-3702(88)90020-3 (1988).

Gong, Z., "Robot signaling its intentions in human-robot teaming", in "HRI Workshop on Explainable Robotic Systems", (2018).

Gong, Z. and Y. Zhang, "Behavior explanation as intention signaling in human-robot teaming", 2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN) pp. 1005–1011 (2018).

Gong, Z. and Y. Zhang, "What is it you really want of me? generalized reward learning with biased beliefs about domain dynamics", in "AAAI", (2020).

Helmert, M., "The fast downward planning system", J. Artif. Int. Res. **26**, 1, 191–246 (2006).

Jaakkola, T., S. Singh and M. Jordan, "Reinforcement learning algorithm for partially observable markov decision problems", Advances in Neural Information Processing Systems **7** (1999).

Kaelbling, L. P., M. L. Littman and A. R. Cassandra, "Planning and acting in partially observable stochastic domains", Artif. Intell. **101**, 1–2, 99–134 (1998).

Kambhampati, S., "Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models", in "AAAI", (2007).

Manikonda, L., T. Chakraborti, S. De, K. Talamadupula and S. Kambhampati, "Ai-mix: Using automated planning to steer human workers towards better crowdsourced plans", in "HCOMP", (2014).

Mao, W. and J. Gratch, "A utility-based approach to intention recognition", in "AAMAS 2004", (2004).

Marthi, B., S. Russell and J. Wolfe, "Angelic semantics for high-level actions", in "Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling", ICAPS'07, p. 232–239 (AAAI Press, 2007).

McCarthy, J., "Epistemological problems of artificial intelligence", in "Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2", IJCAI'77, p. 1038–1044 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1977).

Ng, A. Y. and S. J. Russell, "Algorithms for inverse reinforcement learning", in "Proceedings of the Seventeenth International Conference on Machine Learning", ICML '00, p. 663–670 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000a).

Ng, A. Y. and S. J. Russell, "Algorithms for inverse reinforcement learning", in "Proceedings of the Seventeenth International Conference on Machine Learning", ICML '00, p. 663–670 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000b).

Nguyen, T., S. Sreedharan and S. Kambhampati, "Robust planning with incomplete domain models", Artificial Intelligence **245** (2017).

Schaal, S., "Learning from demonstration", in "Proceedings of the 9th International Conference on Neural Information Processing Systems", NIPS'96, p. 1040–1046 (MIT Press, Cambridge, MA, USA, 1996).

Schrempf, O. and U. Hanebeck, "A generic model for estimating user intentions in human-robot cooperation", in "ICINCO", (2005).

Sharma, A., P. R. Medikeri and Y. Zhang, "Domain concretization from examples: Addressing missing domain knowledge via robust planning", arXiv:2011.09034 (2020).

Silver, D., J. Bagnell and A. Stentz, "Active learning from demonstration for robust autonomous navigation", 2012 IEEE International Conference on Robotics and Automation pp. 200–207 (2012).

Srivastava, S., S. Russell and A. Pinto, "Metaphysics of planning domain descriptions", in "Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence", AAAI'16, p. 1074–1080 (AAAI Press, 2016).

Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction* (A Bradford Book, Cambridge, MA, USA, 2018).

Talamadupula, K., J. Benton, S. Kambhampati, P. Schermerhorn and M. Scheutz, "Planning for human-robot teaming in open worlds", ACM Trans. Intell. Syst. Technol. **1**, 2, URL `https://doi.org/10.1145/1869397.1869403` (2010a).

Talamadupula, K., J. Benton, P. Schermerhorn, S. Kambhampati and M. Scheutz, "Integrating a closed world planner with an open world robot: A case study", in "Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence", AAAI'10, p. 1561–1566 (AAAI Press, 2010b).

Talamadupula, K. and S. Kambhampati, "Herding the crowd: Automated planning for crowdsourced planning", arXiv:1307.7720 (2013).

Weber, C. and D. Bryce, "Planning and acting in incomplete domains", ICAPS'11, p. 274–281 (AAAI Press, 2011).

Wirth, C., J. Fürnkranz and G. Neumann, "Model-free preference-based reinforcement learning", in "AAAI", (2016).

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

Zakershahrak, M., Z. Gong, N. Sadassivam and Y. Zhang, "Online explanation generation for planning tasks in human-robot teaming", in "IROS", (2020a).

Zakershahrak, M., S. R. Marpally, A. Sharma, Z. Gong and Y. Zhang, "Order matters: Generating progressive explanations for planning tasks in human-robot teaming", arXiv:2004.07822 (2020b).

Zakershahrak, M. and Y. Zhang, "Interactive plan explicability in human-robot teaming", in "RO-MAN", (2018).

Zhang, Y., S. Sreedharan, A. Kulkarni, T. Chakraborti, H. Zhuo and S. Kambhampati, "Plan explicability and predictability for robot task planning", 2017 IEEE International Conference on Robotics and Automation (ICRA) pp. 1313–1320 (2017).

Zhuo, H., T. Nguyen and S. Kambhampati, "Refining incomplete planning domain models through plan traces", in "IJCAI", (2013a).

Zhuo, H., Q. Yang, D. Hu and L. Li, "Learning complex action models with quantifiers and logical implications", Artif. Intell. **174**, 1540–1569 (2010).

Zhuo, H. H. and S. Kambhampati, "Action-model acquisition from noisy plan traces", in "Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence", IJCAI '13, p. 2444–2450 (AAAI Press, 2013).

Zhuo, H. H., T. Nguyen and S. Kambhampati, "Model-lite case-based planning", in "Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence", AAAI'13, p. 1077–1083 (AAAI Press, 2013b).

Zhuo, H. H. and Q. Yang, "Action-model acquisition for planning via transfer learning", Artificial Intelligence **212** (2014).

Ziebart, B. D., A. Maas, J. A. Bagnell and A. K. Dey, "Maximum entropy inverse reinforcement learning", AAAI'08, p. 1433–1438 (AAAI Press, 2008).