2012

Planning, Acting, and Learning in Incomplete Domains

Christopher H. Weber
Utah State University

PLANNING, ACTING, AND LEARNING IN INCOMPLETE DOMAINS

by

Christopher H. Weber

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Daniel L. Bryce Major Professor	Dr. Curtis E. Dyreson Committee Member
Dr. Stephen J. Allan Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2011

ABSTRACT

Planning, Acting, and Learning in Incomplete Domains

by

Christopher H. Weber, Master of Science

Utah State University, 2011

Major Professor: Dr. Daniel L. Bryce
Department: Computer Science

The engineering of complete planning domain descriptions is often very costly because of human error or lack of domain knowledge. Learning complete domain descriptions is also very challenging because many features are irrelevant to achieving the goals and data may be scarce. Given incomplete knowledge of their actions, agents can ignore the incompleteness, plan around it, ask questions of a domain expert, or learn through trial and error.

Our agent `Goalie` learns about the preconditions and effects of its incompletely-specified actions by monitoring the environment state. In conjunction with the plan failure explanations generated by its planner `DeFault`, `Goalie` diagnoses past and future action failures. `DeFault` computes failure explanations for each action and state in the plan and counts the number of incomplete domain interpretations wherein failure will occur. The question-asking strategies employed by our extended `Goalie` agent using these conjunctive normal form-based plan failure explanations are goal-directed and attempt to approach always successful execution while asking the fewest questions possible. In sum, `Goalie`:

i) interleaves acting, planning, and question-asking;

ii) synthesizes plans that avoid execution failure due to ignorance of the domain model;

iii) uses these plans to identify relevant (goal-directed) questions;

iv) passively learns about the domain model during execution to improve later replanning attempts;

v) and employs various targeted (goal-directed) strategies to ask questions (actively learn).

Our planner `DeFault` is the first to reason about a domain's incompleteness to avoid potential plan failure. We show that `DeFault` performs best by counting prime implicants (failure diagnoses) rather than propositional models. Further, we show that by reasoning about incompleteness in planning (as opposed to ignoring it), `Goalie` fails and replans less often, and executes fewer actions. Finally, we show that goal-directed knowledge acquisition - prioritizing questions based on plan failure diagnoses - leads to fewer questions, lower overall planning and replanning time, and higher success rates than approaches that naively ask many questions or learn by trial and error.

(59 pages)

DEDICATION

To my family, and my teachers along the way...

ACKNOWLEDGMENTS

This work is indebted to the co-authors of the papers which form its main body, Daniel Bryce and Daniel Morwood of the Department of Computer Science, Utah State University.

This work is supported by DARPA contract HR001-07-C-0060.

Christopher H. Weber

CONTENTS

Page

ABSTRACT iii

DEDICATION v

ACKNOWLEDGMENTS vi

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1 INTRODUCTION 1

2 PLANNING AND ACTING IN INCOMPLETE DOMAINS 3
- 2.1 Abstract 3
- 2.2 Introduction 3
- 2.3 Background and Representation 5
  - 2.3.1 STRIPS Domains 5
  - 2.3.2 Incomplete STRIPS Domains 5
  - 2.3.3 Discussion 7
- 2.4 Planning in Incomplete Domains 8
- 2.5 Heuristics in Incomplete Domains 11
  - 2.5.1 Planning Graph Heuristics 11
  - 2.5.2 Incomplete Domain Heuristics 11
  - 2.5.3 Heuristic Computation 13
- 2.6 Counting Models and Prime Implicants 14
- 2.7 Acting in Incomplete Domains 14
- 2.8 Empirical Evaluation 17
  - 2.8.1 Domains 17
  - 2.8.2 Test Setup 19
  - 2.8.3 Off-line Planning Results 20
  - 2.8.4 On-line Planning and Execution Results 22
- 2.9 Related Work 24
- 2.10 Conclusion 25

3 GOAL-DIRECTED KNOWLEDGE ACQUISITION 26
- 3.1 Abstract 26
- 3.2 Introduction 26
- 3.3 Background and Representation 28
  - 3.3.1 Incomplete STRIPS Domains 28
  - 3.3.2 Incomplete STRIPS Plans 29

3.4 Belief Maintenance and Planning 30
3.4.1 Filtering Observations 30
3.5 Planning 32
3.5.1 Incomplete Domain Relaxed Plans 33
3.6 Goal-Directed Knowledge Acquisition 34
3.6.1 Uninformed QA 35
3.6.2 Non-QA 35
3.6.3 Goal-Directed QA 36
3.6.4 Plan Relevant Questions 36
3.6.5 Plan Failure Diagnosis Relevant Questions 37
3.6.6 Ranking Relevant Questions 37
3.7 Empirical Evaluation 38
3.7.1 Domains 39
3.7.2 Test Setup 40
3.7.3 Results 40
3.8 Conclusion 43

4 CONCLUSION 44

REFERENCES 46

APPENDIX 48

LIST OF TABLES

Table Page

2.1 Number of Plans Having a Greater Number of Successful Domain Interpretations (i.e., Better Quality). **Bold** Indicates Best Performers. 21

2.2 Instances Solved By Domain. 21

3.1 FF Vs. `DeFault` Performance Ratio (Number Solved / Number of Steps / Time / Questions). 40

3.2 Extreme Strategy Average Performance (Number Solved / Number of Steps / Time (seconds) / Questions). **Bold** Indicates Best Performers. 41

3.3 Goal-Directed KA Average Performance Using `DeFault` (Number Solved / Number of Steps / Time (seconds) / Questions). 41

3.4 Plan Vs. Relaxed Plan Ratio in Shannon Entropy Strategy. 42

LIST OF FIGURES

Figure Page

2.1 Incomplete plan example. 10

2.2 Cumulative time in all domains. 22

2.3 Total number of actions comparison, `Goalie` with `DeFault-`FF vs. `Goalie` with `DeFault-`$PI1$. 23

2.4 Total number of plans generated comparison, `Goalie` with `DeFault-`FF vs. `Goalie` with `DeFault-`$PI1$. 23

2.5 Total execution time comparison, `Goalie` with `DeFault-`FF vs. `Goalie` with `DeFault-`$PI1$. 24

CHAPTER 1

INTRODUCTION

Two research papers constitute the main substance of this multi-paper thesis:

1. Planning and Acting in Incomplete Domains [1]
2. Goal-Directed Knowledge Acquisition[2]

These papers compose the bodies of Chapters 2 and 3, respectively.

Their mutual subject is an agent we have developed that plans, acts and learns in incomplete domains. Paper two extends the learning capability of the agent as presented in the first paper; while the agent of the first paper learns passively about its environment as it acts, the second paper's extended agent also learns actively via a question-asking ability. This extended learning ability allows the agent to perform far more effectively in incomplete domains.

The engineering of complete planning domain descriptions is often very costly because of human error or lack of domain knowledge. Learning complete domain descriptions is also very challenging because many features are irrelevant to achieving the stated goals and data may be scarce. Given incomplete knowledge of their actions, agents can ignore the incompleteness, plan around it, ask questions of a domain expert, or learn through trial and error.

Our agent `Goalie` learns about the preconditions and effects of its incompletely-specified actions by monitoring the environment state. In conjunction with the plan failure explanations generated by its planner `DeFault`, `Goalie` diagnoses past and future action failures. `DeFault` computes failure explanations for each action and state in the plan and counts the number of interpretations of the incomplete domain wherein failure will occur. The

---

[1] Co-Author: Daniel Bryce, Department of Computer Science, Utah State University. Accepted for publication in Proceedings of ICAPS'11.

[2] Co-Authors: Daniel Bryce and Daniel Morwood, Department of Computer Science, Utah State University.

question-asking strategies employed by our extended `Goalie` agent using these conjunctive normal form-based plan failure explanations are goal-directed and attempt to approach always successful execution while asking the least questions possible. In sum, `Goalie`:

i) interleaves acting, planning, and question-asking;

ii) synthesizes plans that avoid execution failure due to ignorance of the domain model;

iii) uses these plans to identify relevant (goal-directed) questions;

iv) passively learns about the domain model during execution to improve later replanning attempts;

v) and employs various targeted (goal-directed) strategies to ask questions (actively learn).

Our planner `DeFault` is the first to utilize a forward state space-based heuristic search that reasons about a domain's incompleteness to avoid potential plan failure. We show that `DeFault` performs best by counting prime implicants (failure diagnoses) rather than propositional models. Further, we show that by reasoning about incompleteness during planning (as opposed to ignoring it), `Goalie` fails and replans less often, and executes fewer actions. Finally, we show that goal-directed knowledge acquisition - prioritizing questions based on plan failure diagnoses - leads to fewer questions, lower overall planning and replanning time, and higher success rates than approaches that naively ask many questions or learn by trial and error.

CHAPTER 2
PLANNING AND ACTING IN INCOMPLETE DOMAINS

2.1 Abstract

Engineering complete planning domain descriptions is often very costly because of human error or lack of domain knowledge. Learning complete domain descriptions is also very challenging because many features are irrelevant to achieving the goals and data may be scarce. We present a planner and agent that respectively plan and act in incomplete domains by:

i) synthesizing plans to avoid execution failure due to domain model ignorance; and

ii) passively learning about the domain model during execution to improve later replanning attempts.

Called `DeFault`, our planner is the first to utilize a forward state space-based heuristic search that reasons about a domain's incompleteness to avoid potential plan failure. While `DeFault` computes failure explanations for each action and state in the plan and counts the number of interpretations of the incomplete domain in which failure will occur, our agent `Goalie` learns about the preconditions and effects of incompletely-specified actions by monitoring the environment state. `Goalie` works in conjunction with `DeFault`'s plan failure explanations to diagnose past and future action failures. We show that `DeFault` performs best by counting prime implicants (failure diagnoses) rather than propositional models. Further, we show that by reasoning about incompleteness (as opposed to ignoring it), `Goalie` fails and replans less and executes fewer actions.

2.2 Introduction

The knowledge engineering required to create complete and correct domain descriptions for planning problems is often very costly and difficult (Kambhampati 2007; Wu, Yang, and Jiang 2007). Machine learning techniques have been applied with some success

(Wu, Yang, and Jiang 2007), but still suffer from impoverished data and limitations of the algorithms (Kambhampati 2007). In particular, we are motivated by applications in instructable computing (Mailler et al. 2009) wherein a domain expert teaches an intelligent system about a domain, but can often leave out whole procedures (plans) and aspects of action descriptions. In such cases, the alternative to making domains complete is to plan around the incompleteness. That is, given knowledge of the possible action descriptions, we seek out plans that will succeed despite any (or most of the) incompleteness in the domain formulation.

While prior work (Garland and Lesh 2002) (henceforth abbreviated, GL) has categorized risks to a plan and described plan quality metrics in terms of the risks (essentially single-fault diagnoses of plan failure (de Kleer and Williams 1987)), no prior work using or extending the Stanford Research Institute Problem Solver (STRIPS) model (Fikes and Nilsson 1971) has sought to deliberately synthesize low-risk plans for incomplete domains (notable work in Markov decision processes (Nilim and El Ghaoui 2005; Choudhary et al. 2006) and model-based reinforcement learning (Sutton and Barto 1998) has explored similar issues). Our planner `DeFault` labels partial plans with propositional explanations of failure due to incompleteness (derived from the semantics of assumption-based truth maintenance systems (Bryce 2011)) and either counts failure models or prime implicants (diagnoses) to bias search. Our agent `Goalie` passively learns about the incomplete domain as it executes actions, like Chang and Amir (2006) (henceforth abbreviated, CA). Unlike CA, `Goalie` executes plans that are robust to domain incompleteness. Within `Goalie`, we compare the use of robust plans generated by our planner `DeFault`, and plans that are generated in the spirit of CA which are not intentionally robust, i.e., they are only optimistically successful. We demonstrate that the effort to synthesize robust plans is justified because `DeFault` executes fewer overall actions and fails and replans less.

This paper is organized as follows. The next section details incomplete STRIPS, the language we use to describe incomplete domains. We follow with our approach to plan synthesis and search heuristics. We discuss alternatives to reasoning about failure explana-

tions, including model counting and prime implicant counting. We describe our execution monitoring and replanning strategy, and then provide an empirical analysis, discussion of related work, and conclusion.

2.3 Background and Representation

The incomplete STRIPS model minimally relaxes the classical STRIPS model to allow for possible preconditions and effects. In the following, we review the STRIPS model and present the incomplete STRIPS model.

2.3.1 STRIPS Domains

A STRIPS planning domain D defines the tuple (P, A, I, G), where:

P is a set of propositions;

A is a set of action descriptions;

$I \subseteq P$ defines a set of initially true propositions; and

$G \subseteq P$ defines the goal propositions.

Each action $a \in A$ defines:

$\text{pre}(a) \subseteq P$, a set of preconditions;

$\text{add}(a) \subseteq P$, a set of add effects; and

$\text{del}(a) \subseteq P$, a set of delete effects.

A plan $\pi = (a_0, ..., a_{n-1})$ in D is a sequence of actions. This sequence of actions corresponds to a sequence of states $(s_0, ..., s_n)$, where:

$s_0 = I$;

$\text{pre}(a_t) \subseteq s_t$ for $t = 0, ..., n-1$;

$G \subseteq s_n$; and

$s_{t+1} = s_t \backslash \text{del}(a_t) \cup \text{add}(a_t)$ for $t = 0, ..., n-1$.

2.3.2 Incomplete STRIPS Domains

Incomplete STRIPS domains are identical to STRIPS domains, with the exception that the actions are incompletely specified. Much like planning with incomplete state information (Domshlak and Hoffmann 2007; Bryce, Kambhampati, and Smith 2008), the action

incompleteness is not completely unbounded. The preconditions and effects of each action can be any subset of the propositions P; the incompleteness is with regard to a lack of knowledge about which of the subsets corresponds to each precondition and effect. To narrow the possibilities, we find it convenient to refer to the *known*, *possible*, and *impossible* preconditions and effects.

For example, an action's preconditions must consist of the known preconditions, and it must not contain the impossible preconditions, but we do not know if it contains the possible preconditions. The union of the known, possible, and impossible preconditions must equal P. Therefore, an action can represent any two, and we can infer the third. We choose to represent the known and possible, but note that GL represent the known and impossible, noting that the trade-off makes our representation more appropriate if there are fewer possible action features.

An incomplete STRIPS domain $\tilde{D}$ defines the tuple (P, $\tilde{A}$, I, G), where:

P is a set of propositions;

$\tilde{A}$ is a set of incomplete action descriptions;

$I \subseteq P$ defines a set of initially true propositions; and

$G \subseteq P$ defines the goal propositions.

Each action $\tilde{a} \in \tilde{A}$ defines:

$\text{pre}(\tilde{a}) \subseteq P$, a set of known preconditions;

$\widetilde{\text{pre}}(\tilde{a}) \subseteq P$, a set of possible preconditions;

$\text{add}(\tilde{a}) \subseteq P$, a set of known add effects;

$\widetilde{\text{add}}(\tilde{a}) \subseteq P$, a set of possible add effects;

$\text{del}(\tilde{a}) \subseteq P$, a set of known delete effects; and

$\widetilde{\text{del}}(\tilde{a}) \subseteq P$, a set of possible delete effects.

Consider the following incomplete domain:

$P = \{p, q, r, g\}$; $\tilde{A} = \{\tilde{a}, \tilde{b}, \tilde{c}\}$; $I = \{p, q\}$; and $G = \{g\}$.

The actions are defined:

$\text{pre}(\tilde{a}) = \{p, q\},\ \widetilde{\text{pre}}(\tilde{a}) = \{r\},\ \widetilde{\text{add}}(\tilde{a}) = \{r\},\ \widetilde{\text{del}}(\tilde{a}) = \{p\};$

$\text{pre}(\tilde{b}) = \{p\},\ \text{add}(\tilde{b}) = \{r\},\ \text{del}(\tilde{b}) = \{p\},\ \widetilde{\text{del}}(\tilde{b}) = \{q\}$; and

$\text{pre}(\tilde{c}) = \{r\},\ \widetilde{\text{pre}}(\tilde{c}) = \{q\},\ \text{add}(\tilde{c}) = \{g\}.$

The set of incomplete domain features F is comprised of the following propositions for each $\tilde{a} \in \tilde{A}$:

$\widetilde{\text{pre}}(\tilde{a}, p)$ if $p \in \widetilde{\text{pre}}(\tilde{a})$;

$\widetilde{\text{add}}(\tilde{a}, p)$ if $p \in \widetilde{\text{add}}(\tilde{a})$; and

$\widetilde{\text{del}}(\tilde{a}, p)$ if $p \in \widetilde{\text{del}}(\tilde{a})$.

An interpretation $\mathsf{F}^i \subseteq \mathsf{F}$ of the incomplete STRIPS domain defines a STRIPS domain, in that every feature $f \in \mathsf{F}^i$ indicates that a possible precondition or effect is a respective known precondition or known effect. Those features not in F^i are not preconditions or effects.

A plan π for $\tilde{D}$ is a sequence of actions that when applied *can lead* to a state wherein the goal is satisfied. A plan $\pi = (\tilde{a}_0, ..., \tilde{a}_{n-1})$ in an incomplete domain $\tilde{D}$ is a sequence of actions that corresponds to the *optimistic* sequence of states $(s_0, ..., s_n)$, where:

$s_0 = I$;

$\text{pre}(\tilde{a}_t) \subseteq s_t$ for $t = 0, ..., n$;

$G \subseteq s_n$; and

$s_{t+1} = s_t \backslash \text{del}(\tilde{a}_t) \cup \text{add}(\tilde{a}_t) \cup \widetilde{\text{add}}(\tilde{a}_t)$ for $t = 0, ..., n-1$.

For example, the plan $(\tilde{a}, \tilde{b}, \tilde{c})$ corresponds to the state sequence

$(s_0 = \{p, q\},\ s_1 = \{p, q, r\},\ s_2 = \{q, r\},\ s_3 = \{q, r, g\})$,

where the goal is satisfied in s_3.

2.3.3 Discussion

Our definition of the plan semantics sets a loose requirement that plans with incomplete actions succeed under the most *optimistic* conditions: possible preconditions need not be satisfied and the possible add effects (but not the possible delete effects) are assumed to

occur when computing successor states. This notion of optimism is similar to that of GraphPlan (Blum and Furst 1995) in that both assert every proposition that could be made true at a particular time even if only a subset of the propositions can actually be made true. In GraphPlan, there *may* exist a plan to establish a proposition if the proposition appears in the planning graph. In our definitions there *does* exist an interpretation of the incomplete domain that will establish a proposition if it appears in a state (Bryce 2011), and this interpretation *may* correspond to the true domain. In GraphPlan, failing to assert a proposition that may be established could eliminate plans, and in our case, failing to assert a proposition would prevent us from computing interpretations of the incomplete domain that achieve the goal.

We ensure that the plan is valid for the least constraining (most optimistic) interpretation of the incomplete domain. If the plan can achieve the goal in the most optimistic interpretation, it may achieve the goal in others; if the goal is not reachable in this interpretation, it cannot be reached in any interpretation (Bryce 2011). As we show, we can efficiently determine the interpretations in which a plan is invalid and use the number of such failed interpretations as a plan quality metric.

2.4 Planning in Incomplete Domains

We present a forward state space planner called `DeFault` that attempts to minimize the number of interpretations of the incomplete domain that can result in plan failure. `DeFault` generates states reached under the optimistic interpretation of the incomplete domain, but labels each state proposition with the interpretations (a failure explanation) in which it is impossible to achieve the proposition. As such, the number of interpretations labeling the goals reached by a plan indicates the number of failed interpretations. By counting interpretations (i.e., propositional model counting), we can determine the quality of a plan.

`DeFault` labels propositions and actions with domain interpretations that, respectively, fail to achieve the proposition or fail to achieve the preconditions of an action. That is, labels indicate the cases wherein a proposition is false (i.e., the plan fails to establish the proposition). Labels $d(\cdot)$ are represented as propositional sentences over F whose models

correspond to domain interpretations.

Initially, each proposition $p_0 \in s_0$ is labeled $d(p_0) = \bot$ to denote that there are no failed interpretations affecting the initial state, and each $p_0 \notin s_0$ is labeled $d(p_0) = \top$. For all $t \geq 0$, we define:

$$d(\tilde{a}_t) = d(\tilde{a}_{t-1}) \vee \bigvee_{p \in \text{pre}(\tilde{a})} d(p_t) \vee \bigvee_{p \in \widetilde{\text{pre}}(\tilde{a})} (d(p_t) \wedge \widetilde{\text{pre}}(\tilde{a}_t, p)) \tag{2.1}$$

$$d(p_{t+1}) = \begin{cases} d(p_t) \wedge d(\tilde{a}_t) & : p \in \text{add}(\tilde{a}_t) \\ d(p_t) \wedge (d(\tilde{a}_t) \vee \neg\widetilde{\text{add}}(\tilde{a}_t, p)) & : p \in \widetilde{\text{add}}(\tilde{a}_t) \\ \top & : p \in \text{del}(\tilde{a}_t) \\ d(p_t) \vee \widetilde{\text{del}}(\tilde{a}_t, p) & : p \in \widetilde{\text{del}}(\tilde{a}_t) \\ d(p_t) & : \text{otherwise} \end{cases} \tag{2.2}$$

where $d(\tilde{a}_{-1}) = \bot$. The intuition behind the label propagation is that in Equation 2.1 an action will fail in the domain interpretations $d(\tilde{a}_t)$ where a prior action failed, a known precondition is not satisfied, or a possible precondition (which is a known precondition for the interpretation) is not satisfied. As defined by Equation 2.2, the plan will fail to achieve a proposition at time $t + 1$ in all interpretations in which:

i) the plan fails to achieve the proposition at time t and the action fails;

ii) the plan fails to achieve the proposition at time t and the action fails or it does not add the proposition in the interpretation;

iii) the action deletes the proposition;

iv) the plan fails to achieve the proposition at time t or in the interpretation the action deletes the proposition; or

v) the action does not affect the proposition and any prior failed interpretations still apply.

A consequence of our definition of action failure is that each action fails if any prior action fails. This definition follows from the semantics that the state becomes undefined if we apply an action whose preconditions are not satisfied. While we use this notion in

plan synthesis, we explore the semantics that the state does not change (i.e., it is defined) upon failure when we discuss acting in incomplete domains. The reason that we define action failures in this manner is that we can determine all failed interpretations affecting a plan $d(\pi)$, defined by $d(\tilde{a}_{n-1}) \vee \bigvee_{g \in G} d(g_n)$. By $d(\tilde{a}_t)$, it is possible to determine the interpretations that fail to successfully execute the plan up to and including time t.

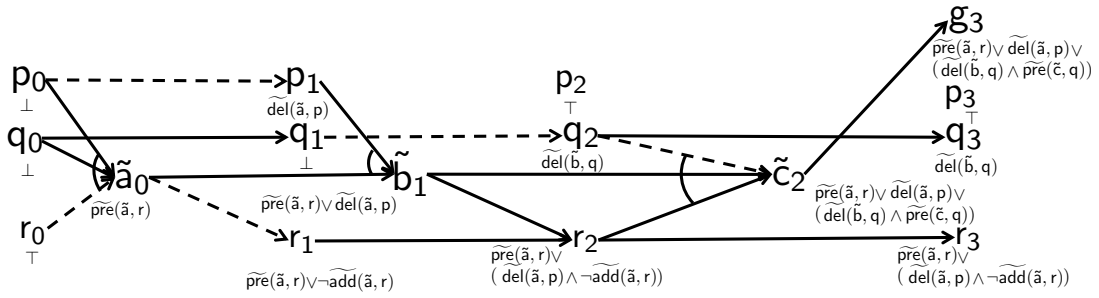

Fig. 2.1: Incomplete plan example.

Consider the plan depicted in Figure 2.1. The propositions in each state and each action at each time are labeled by the propositional sentence below it. The edges in the figure connecting the propositions and actions denote what must be true to successfully execute an action or achieve a proposition. The dashed edges indicate that action incompleteness affects the ability of an action or proposition to support a proposition.

Note that $\tilde{a}$ possibly deletes p, so the edge denoting its persistence is dashed. The propositional sentences $d(\cdot)$ below each proposition and action denote the domain interpretations in which an action will fail or a proposition will not be achieved.

Further, $\tilde{b}$ at time one, $\tilde{\mathsf{b}}_1$, will fail if either $\widetilde{\mathrm{pre}}(\tilde{a}, r)$ or $\widetilde{\mathrm{del}}(\tilde{a}, p)$ is true in the interpretation. Thus,

$$d(\pi) = \widetilde{\mathrm{pre}}(\tilde{a}, r) \vee \widetilde{\mathrm{del}}(\tilde{a}, p) \vee (\widetilde{\mathrm{del}}(\tilde{b}, q) \wedge \widetilde{\mathrm{pre}}(\tilde{c}, q)),$$

and any domain interpretation satisfying $d(\pi)$ will fail to execute the plan and achieve the goal.

2.5 Heuristics in Incomplete Domains

Similar to propagating failed interpretation labels in a plan, we can propagate labels in the relaxed planning problem to compute a search heuristic. The primary heuristic is the number of actions in a relaxed plan. While we do not use the number of failed domain interpretations as the primary heuristic, we use the failure labels to bias the selection of the relaxed plan actions and break ties between search nodes with an equivalent number of actions in their relaxed plans. As in recent trends in satisficing planning (classical, conformant, etc.) we want high quality solutions, but not at the expense of returning no solution. We solve the relaxed planning problem using a planning graph and thus begin with a brief description of planning graphs.

2.5.1 Planning Graph Heuristics

A relaxed planning graph is a layered graph of sets of vertices $(\mathcal{P}_t, \mathcal{A}_t, ..., \mathcal{A}_{t+m}, \mathcal{P}_{t+m+1})$. The planning graph built for state s_t defines:

$$\mathcal{P}_t = \{p_t | p \in s_t\};$$

$$\mathcal{A}_{t+k} = \{a_t | \forall_{p \in \text{pre}(a)} p_t \in \mathcal{P}_{t+k}, a \in A \cup A(P)\}; \text{ and}$$

$$\mathcal{P}_{t+k+1} = \{p_{t+k+1} | a_{t+k} \in \mathcal{A}_{t+k}, p \in \text{add}(a)\};$$

for $k = 0, ..., m$. The set $A(P)$ includes noop actions for each proposition, such that

$$A(P) = \{a(p) | p \in P,\ \text{pre}(a(p)) = \text{add}(a(p)) = p,\ \text{del}(a(p)) = \emptyset\}.$$

The h^{FF} heuristic (Hoffmann and Nebel 2001) solves this relaxed planning problem by choosing actions from $\mathcal{A}_{t+m}$ to support the goals in $\mathcal{P}_{t+m+1}$, and recursively so for each chosen action's preconditions when counting the number of chosen actions.

2.5.2 Incomplete Domain Heuristics

Propagating failed interpretations in the planning graph resembles propagating failed interpretations over a plan. The primary difference is how we define the failed interpretations for a proposition when the proposition has multiple sources of support. Recall that we allow only serial plans and that at each time each state proposition is supported by persistence and/or a single action (action choice is handled in the search space). In a level

of the relaxed planning graph, there are potentially many actions supporting a proposition, and we select the supporter with the fewest failed interpretations. The chosen supporting action, denoted $\hat{a}_{t+k}(p)$, determines the failed interpretations affecting proposition p at the level $t+k+1$.

A relaxed planning graph with propagated labels is a layered graph of sets of vertices of the form $(\hat{\mathcal{P}}_t, \hat{\mathcal{A}}_t, ..., \hat{\mathcal{A}}_{t+m}, \hat{\mathcal{P}}_{t+m+1})$. The relaxed planning graph built for state $\tilde{s}_t$ defines:

$\hat{\mathcal{P}}_0 = \{\hat{p}_t | p \in \tilde{s}_t\}$;

$\hat{\mathcal{A}}_{t+k} = \{\hat{a}_{t+k} | \forall_{p \in \text{pre}(\tilde{a})} \hat{p}_{t+k} \in \hat{\mathcal{P}}_{t+k}, \tilde{a} \in \tilde{A} \cup A(P)\}$; and

$\hat{\mathcal{P}}_{t+k+1} = \{\hat{p}_{t+k+1} | \hat{a}_{t+k} \in \hat{\mathcal{A}}_{t+k}, p \in \text{add}(\tilde{a}) \cup \widetilde{\text{add}}(\tilde{a})\}$;

for $k = 0, ..., m$. Much like the successor function used to compute next states, the relaxed planning graph assumes an optimistic semantics for effects by adding possible add effects to proposition layers. However, as we will explain below, it associates failed interpretations with the possible adds.

Each planning graph vertex has a label, denoted $\hat{d}(\cdot)$. The failed interpretations $\hat{d}(\hat{p}_t)$ affecting a proposition are defined such that:

$\hat{d}(\hat{p}_t) = d(p_t)$;

and for $k \geq 0$,

$$\widehat{d}(\tilde{a}_{t+k}) = \bigvee_{p \in \text{pre}(\tilde{a})} \hat{d}(\hat{p}_{t+k}) \vee \bigvee_{p \in \widetilde{\text{pre}}(\tilde{a})} (\hat{d}(\hat{p}_{t+k}) \wedge \widetilde{\text{pre}}(\tilde{a}, p)) \tag{2.3}$$

$$\hat{d}(\hat{p}_{t+k+1}) = \begin{cases} \widehat{d}(\hat{a}_{t+k}(p)) & : p \in \text{add}(\hat{a}_{t+k}(p)) \\ \widehat{d}(\hat{a}_{t+k}(p)) \vee \neg\widetilde{\text{add}}(\hat{a}_{t+k}(p), p) & : p \in \widetilde{\text{add}}(\hat{a}_{t+k}(p)) \end{cases} \tag{2.4}$$

Every action in every level k of the planning graph will fail in any interpretation in which its preconditions are not supported (Equation 2.3). A proposition will fail to be achieved in any interpretation in which the chosen supporting action fails to add the proposition (Equation 2.4).

We note that the rules for propagating labels in the planning graph differ from the rules for propagating labels in the state space. In the state space, the action failure labels

include interpretations wherein any prior action fails. In the relaxed planning problem, an action's failure labels include only the interpretations affecting its preconditions, and not prior actions; it is not clear which actions will be executed prior to achieving a proposition because many actions may be used to achieve other propositions at the same time.

2.5.3 Heuristic Computation

We terminate the relaxed planning graph expansion at level $t + k + 1$ when one of the following conditions is met:

i) the planning graph reaches a fixed point where the explanations do not change, $\hat{d}(\hat{p}_{t+k}) = \hat{d}(\hat{p}_{t+k+1})$ for all $p \in P$; or

ii) the goals have been reached at $t + k + 1$ and the fixed point has not yet been reached.

Our $h^{\sim FF}$ heuristic makes use of the chosen supporting action $\hat{a}_{t+k}(p)$ for each proposition requiring support in the relaxed plan. Hence, it measures the number of actions used while attempting to minimize failed interpretations (the supporting actions are chosen by comparing failure explanations). Our $h^{\sim M}$ heuristic measures the number of interpretations that fail to reach the goals in the last level:

$$h^{\sim M} = |M(\vee_{p \in G}\ \hat{d}(\hat{p}_{t+m+1}))|,$$

where $m + 1$ is the last level of the planning graph, and $M(\cdot)$ is the set of models of a formula.

`DeFault` uses both heuristics, treating $h^{\sim FF}$ as the primary heuristic and using $h^{\sim M}$ to break ties. While it is likely that swapping the role of the heuristics may lead to higher quality plans (fewer failed interpretations), our informal experiments determined that the scalability of `DeFault` is greatly limited in such cases – measuring failed interpretations is not correlated with solution depth in the search graph, unlike relaxed plan length. The relaxed plans are informed by the propagated explanations because we use the failure explanation to bias action selection.

2.6 Counting Models and Prime Implicants

Failure explanations $d(\cdot)$ and $\hat{d}(\cdot)$ are propositional sentences that help bias decisions in our heuristic-based search. Namely, we assume that we can count the number of propositional models of these sentences to indicate how many interpretations of the incomplete domain will fail to successfully execute a plan. Model counting is intractable (Roth 1996), but by representing the sentences as Ordered Binary Decision Diagrams (OBDDs) (Bryant 1986), model counting is polynomial in the size of the OBDD (Darwiche and Marquis 2002), although it can be exponential sized in the worst case).

In addition to OBDDs and model counting, we explore counting prime implicants (PIs) (also called diagnoses). Counting PIs allows us to compute heuristic $h^{\sim PI}$, which is similar to $h^{\sim M}$. A set of PIs is a set of conjunctive clauses – similar to a DNF, wherein no clause is subsumed by another. These are used in model-based diagnosis to represent diagnoses – sets of incomplete features that must interact to cause system failure (de Kleer and Williams 1987). We find it useful to bound the cardinality – the number of conjuncts – of the PIs, effectively over-approximating the models of a propositional sentence.

Instead of counting the models of two labels $d(\cdot)$ and $d'(\cdot)$, we can compare the number of PIs (as in $h^{\sim PI}$). Our intuition is that having fewer diagnoses of failure is preferred, just as is having fewer models of failure (even though having fewer PIs does not always imply fewer models). The advantage is that counting PIs is much less expensive than counting models, especially if we bound the cardinality of the PIs. Finally, when counting PIs, we use a heuristic that compares two sets in terms of the number of cardinality-one PIs, and if equal, the number of cardinality-two PIs, and so on. The intuition behind comparing PIs in this fashion is that smaller PIs are typically satisfied by a larger number of models and are thus more representative of the number of models. That is, a sentence with one cardinality-one PI will have more models than a sentence with one cardinality-two PI.

2.7 Acting in Incomplete Domains

Acting in incomplete domains provides an opportunity to learn about the domain by observing the states resulting from execution. In the following, we describe what our agent

Goalie can learn from acting in incomplete domains and how it achieves its goals. Goalie will continue to execute a plan until it is faced with an action that is guaranteed to fail or in hindsight it has determined that the plan failed.

Goalie maintains propositional sentence ϕ defined over $\mathsf{F} \cup \{fail\}$, which describes the current knowledge of the incomplete domain. The proposition $fail$ denotes whether Goalie believes that its current plan may have failed – it is not always possible to determine if an action applied in the past did not have its preconditions satisfied. Initially, Goalie believes $\phi = \top$, denoting its complete lack of knowledge of the incomplete domain and whether its current plan will fail. If Goalie executes $\tilde{a}$ in state s and transitions to state s', then it updates its knowledge as $\phi \wedge o(s, \tilde{a}, s')$, where:

$$o(s, \tilde{a}, s') = \begin{cases} (fail \wedge o^-) \vee o^+ & : s = s' \\ o^+ & : s \neq s' \end{cases} \tag{2.5}$$

$$o^- = \bigvee_{\substack{\widetilde{\mathrm{pre}}(\tilde{a},p) \in \mathsf{F}: \\ p \notin s}} \widetilde{\mathrm{pre}}(\tilde{a}, p) \tag{2.6}$$

$$o^+ = o^{pre} \wedge o^{add} \wedge o^{del} \tag{2.7}$$

$$o^{pre} = \bigwedge_{\substack{\widetilde{\mathrm{pre}}(\tilde{a},p) \in \mathsf{F}: \\ p \notin s}} \neg\widetilde{\mathrm{pre}}(\tilde{a}, p) \tag{2.8}$$

$$o^{add} = \bigwedge_{\substack{\widetilde{\mathrm{add}}(\tilde{a},p) \in \mathsf{F}: \\ p \in s' \setminus s}} \widetilde{\mathrm{add}}(\tilde{a}, p) \wedge \bigwedge_{\substack{\widetilde{\mathrm{add}}(\tilde{a},p) \in \mathsf{F}: \\ p \notin s \cup s'}} \neg\widetilde{\mathrm{add}}(\tilde{a}, p) \tag{2.9}$$

$$o^{del} = \bigwedge_{\substack{\widetilde{\mathrm{del}}(\tilde{a},p) \in \mathsf{F}: \\ p \in s \setminus s'}} \widetilde{\mathrm{del}}(\tilde{a}, p) \wedge \bigwedge_{\substack{\widetilde{\mathrm{del}}(\tilde{a},p) \in \mathsf{F}: \\ p \in s \cap s'}} \neg\widetilde{\mathrm{del}}(\tilde{a}, p) \tag{2.10}$$

We assume that the state will remain unchanged when Goalie executes an action whose precondition is not satisfied by the state, and because the state is observable, Equation 2.5 references both the case in which the state does not change and the case in which it does change. If the state does not change, either the action failed and one of its unsatisfied possible preconditions is a precondition (Equation 3.1), or the action succeeded (Equation

3.1). If the state changes, `Goalie` knows that the action succeeded. If an action succeeds, `Goalie` can conclude that:

i) each possible precondition that was not satisfied is not a precondition (Equation 3.1);

ii) each possible add effect that appears in the successor but not the predecessor state is an add effect, and each that does not appear in either state is, in fact, not an add effect (Equation 3.1);

iii) each possible delete effect that appears in the predecessor but not the successor is a delete effect, and each that appears in both states is not a delete effect (Equation 3.1).

Using ϕ, it is possible to determine if the next action in a plan, or any subsequent action, can or will fail. If $\phi \wedge d(\tilde{a}_{t+k})$ is satisfiable, then $\tilde{a}_{t+k}$ *can* fail, and if $\phi \models d(\tilde{a}_{t+k})$, then $\tilde{a}_{t+k}$ *will* fail. `Goalie` will execute an action if it may not fail, even if later actions in its plan will fail. If `Goalie` determines that its next action will fail, or a prior action failed ($\phi \models fail$), it will replan. `Goalie` uses ϕ to modify the actions during replanning by checking for each incomplete domain feature $f \in \mathsf{F}$ if $\phi \models f$ or if $\phi \models \neg f$. Each such literal entailed by ϕ indicates that the respective action has the possible feature as a known or impossible feature; all other features remain as possible features.

Algorithm 1 is the strategy used by `Goalie`. First, the algorithm initializes the agent's knowledge and plan (lines 1 and 2). Then, while the plan is non-empty and the goal is not achieved (line 3), the agent proceeds with execution. The agent selects the next action in the plan (line 4) and determines if it can apply the action (line 6). If it applies the action, the next state is returned by the environment/simulator (line 7) and the agent updates its knowledge (line 8 and Equation 2.5) and state (line 9). Otherwise, the agent determines that the plan will fail (lines 10 and 11). If the plan has failed (line 12), the agent forgets its knowledge of the plan failure (line 13) and finds a new plan using its new knowledge (line 14). `Goalie` cannot guaranteed success unless it can find a plan that will not fail (i.e., $d(\pi) = \bot$).

`Goalie` is not hesitant to apply actions that may fail because trying actions is its only way to learn about them. However, `Goalie` is able to determine when an action will fail

Algorithm 1: `Goalie`$(s, G, \tilde{A})$

Input: state s, goal G, actions $\tilde{A}$

$\phi \leftarrow \top$;
$\pi \leftarrow Plan(s, G, \tilde{A}, \phi)$;
while $\pi \neq ()$ *and* $G \not\subseteq s$ **do**
  $\tilde{a} \leftarrow \pi.first()$;
  $\pi \leftarrow \pi.rest()$;
  if $pre(\tilde{a}) \subseteq s$ *and* $\phi \not\models \bigvee_{\widetilde{pre}(\tilde{a},p) \in \mathsf{F}: p \notin s} \widetilde{pre}(\tilde{a}, p)$ **then**
    $s' \leftarrow Execute(\tilde{a})$;
    $\phi \leftarrow \phi \wedge o(s, \tilde{a}, s')$;
    $s \leftarrow s'$;
  else
    $\phi \leftarrow \phi \wedge fail$;
  if $\phi \models fail$ **then**
    $\phi \leftarrow \exists_{fail} \phi$;
    $\pi \leftarrow Plan(s, G, \tilde{A}, \phi)$;

and so replans. More conservative strategies are possible if we assume that `Goalie` can query a domain expert about action features to avoid potential plan failure, but we leave such goal-directed knowledge acquisition for future work.

2.8 Empirical Evaluation

The empirical evaluation is divided into four sections: the domains used for the experiments, the test setup used, results for off-line planning, and results for planning and execution. The questions we seek to answer include:

Q1 – Does reasoning about incompleteness lead to high quality plans?

Q2 – Does counting prime implicants perform better than counting models?

Q3 – As incomplete features increase, does stronger reasoning about incompleteness help?

Q4 – Does reasoning about incompleteness reduce the number of execution failures?

2.8.1 Domains

We use four domains in the evaluation: a modified Pathways, Bridges, a modified PARC Printer, and Barter World. For these domains, we created multiple instances by injecting

incomplete features with probabilities 0.25, 0.5, 0.75, and 1.0. An instance may possess up to 10,000 incomplete features. Planning results are taken from 10 random instances (varying F) of each problem. Within these, each planning and execution result is one of ten ground-truth domains selected by the simulator. The problem generators and our planner are available at: http://www.cs.usu.edu/∼danbryce/software/default.jar.

The Pathways (PW) domain from the International Planning Competition (IPC) involves actions that model chemical reactions in signal transduction pathways. Pathways is a naturally incomplete domain in which the lack of knowledge of the reactions is quite common, and are an active research topic in biology (Choudhary et al. 2006).

The Bridges (BR) domain consists of a traversable grid in which the task is to find a different treasure at each corner of the grid. Grids are square and vary in dimension (2-16). There are three versions:

BR1 – a bridge might be required to cross between some grid locations (a possible precondition);

BR2 – BR1, plus many of the bridges may have a troll living underneath that will take all the treasure accumulated (a possible delete effect);

BR3 – BR2, plus the corners may give additional treasures (possible add effects).

The PARC Printer (PP) domain from the IPC involves planning paths for sheets of paper through a modular printer. A source of domain incompleteness is that a module accepts only certain paper sizes, but its documentation is incomplete. Thus, in using the module, paper size becomes a possible precondition to actions.

The Barter World (BW) domain involves navigating a grid and bartering items to travel between locations. The domain is incomplete because actions that acquire items are not always known to be successful (possible add effects) and traveling between locations may both require certain items (possible preconditions) and result in the loss of an item (possible delete effects). Grids vary in dimension (2-16) and items in number (1-4).

2.8.2 Test Setup

The tests were run on a Linux machine with a 3 Ghz Xeon processor, a memory limit of 2GB, and a time limit of 20 minutes per run for the off-line planning invocation and 60 minutes for each on-line planning and execution test. All code is written in Java and run on the 1.6 JVM.

We use five configurations of the planner: `DeFault-`FF, `DeFault-`PIk (k = 1, 2, 3), and `DeFault-`BDD, each of which differ in how they reason about domain incompleteness. `DeFault-`FF does not compute failure explanations and uses the FF heuristic; inspired by the planner used by CA, it is likely to find a plan that will work for only the most optimistic domain interpretation. `DeFault-`PIk, where k is the bound on the cardinality of the prime implicants, counts prime implicants to compare failure explanations. `DeFault-`BDD uses OBDDs to represent and count failure explanations. `DeFault` uses a greedy best-first search with deferred heuristic evaluation and a dual-queue for preferred and nonpreferred operators (Helmert 2006).

The number of failed interpretations for a plan π found by any of the planners is found by counting models of an OBDD representing $d(\pi)$. The versions of the planner are compared by the proportion of interpretations of the incomplete domain that achieve the goal and total planning time in seconds. The plot in the following section depict these results using the cumulative planning time to identify the performance over all problems and domains. We also report detailed results on the number of solved problems per domain and the relative quality of solutions (successful domain interpretations).

Additionally, we compare the off-line planning results to the conformant probabilistic planner POND (with N=10 particles in its heuristic) (Bryce, Kambhampati, and Smith 2008) that solves translated instances of the incomplete STRIPS problems. We set the minimum required probability of goal satisfaction to the minimum proportion of successful domain interpretations of plans found by the other approaches. We do not provide the details of the translation because the results are very poor, but refer the reader to an extended version of this work (Bryce 2011). We attempted a comparison to PFF (Domsh-

lak and Hoffmann 2007), but the implementation proved unstable for all but the smallest instances (results not shown).

2.8.3 Off-line Planning Results

Table 2.1 shows when each configuration finds a better solution (number of successful interpretations) than another; for example, $PI1$ finds a better solution than FF 629 times. Table 2.2 lists the number of solved problems for each planner. Figure 2.2 plots the cumulative total planning time (by every 100^{th} data point to enhance readability).

We see that Q1 is answered positively by the results. Table 2.1 shows that plan quality is improved by reasoning about incompleteness (through `DeFault`-PIk or `DeFault`-BDD), though Table 2.2 shows that scalability suffers. However, we note that minimizing the number of failed interpretations can be phrased as a conformant probabilistic planning problem, which is notoriously difficult (Domshlak and Hoffmann 2007; Bryce, Kambhampati, and Smith 2008), and expecting the scalability of a classical planner is perhaps unreasonable.

Q2 is answered overall positively by our experiments because the PIk counting approaches solve more problems with better quality and in less time than the BDD model counting approach (see Table 2.2 and Figure 2.2).

Q3 is answered negatively because as the probability of injecting incomplete features grows from 0.25 to 1.0, $PI3$ initially solves the most problems, but then $PI2$, and finally $PI1$ solves the most problems in each domain (see Table 2.2). A possible explanation for this result is that as incompleteness increases, more complete reasoning about it becomes too costly, and thus a more coarse approach is needed; however, the BDD approach, while not the best, seems to degrade less as the incompleteness increases. It is likely that the OBDD package implementation (Vahidi 2011) is to be credited for the BDD approach's performance because model counting can become prohibitively expensive in larger problems.

Table 2.2 indicates that POND is not competitive and suggests that existing approaches are not directly applicable to planning in incomplete domains. We note that `DeFault` is inspired by POND, but employs more approximate reasoning about incompleteness by using bounded prime implicants (see (Bryce 2011) for a more thorough discussion).

Table 2.1: Number of Plans Having a Greater Number of Successful Domain Interpretations (i.e., Better Quality). **Bold** Indicates Best Performers.

	FF	PI1	PI2	PI3	BDD
FF	0	155	161	161	123
PI1	**629**	0	**79**	**78**	**208**
PI2	**619**	77	0	46	**208**
PI3	**594**	62	**51**	0	**199**
BDD	**512**	189	189	187	0

Table 2.2: Instances Solved By Domain.

Domain	FF	PI1	PI2	PI3	BDD	POND
PP 0.25	130	83	85	**86**	80	10
PP 0.5	130	87	**88**	87	80	0
PP 0.75	130	82	**83**	81	80	0
PP 1.0	13	**10**	9	9	8	0
PP	403	262	**265**	263	248	10
BR1 0.25	40	**22**	**22**	**22**	**22**	2
BR1 0.5	39	**20**	**20**	**20**	**20**	2
BR1 0.75	36	**19**	**19**	**19**	**19**	2
BR1 1.0	4	**2**	**2**	**2**	**2**	1
BR2 0.25	38	20	20	20	**21**	3
BR2 0.5	35	**25**	**25**	**25**	23	3
BR2 0.75	35	**22**	21	21	21	2
BR2 1.0	4	**2**	**2**	**2**	**2**	1
BR3 0.25	45	**36**	**36**	**36**	**36**	1
BR3 0.5	47	**33**	**33**	**33**	32	2
BR3 0.75	46	39	39	39	**41**	1
BR3 1.0	5	**4**	**4**	**4**	3	1
BR	374	**244**	243	243	242	21
BW 0.25	150	106	128	**129**	108	60
BW 0.5	150	134	**137**	134	118	45
BW 0.75	150	**140**	138	137	111	27
BW 1.0	15	**14**	**14**	**14**	11	2
BW	465	394	**417**	414	348	155
PW 0.25	160	**40**	**40**	**40**	**40**	19
PW 0.5	160	**70**	60	50	60	13
PW 0.75	170	**60**	50	40	**60**	12
PW 1.0	19	5	6	6	**7**	2
PW	509	**175**	156	136	167	46
Total	1751	1075	**1081**	1056	1005	232

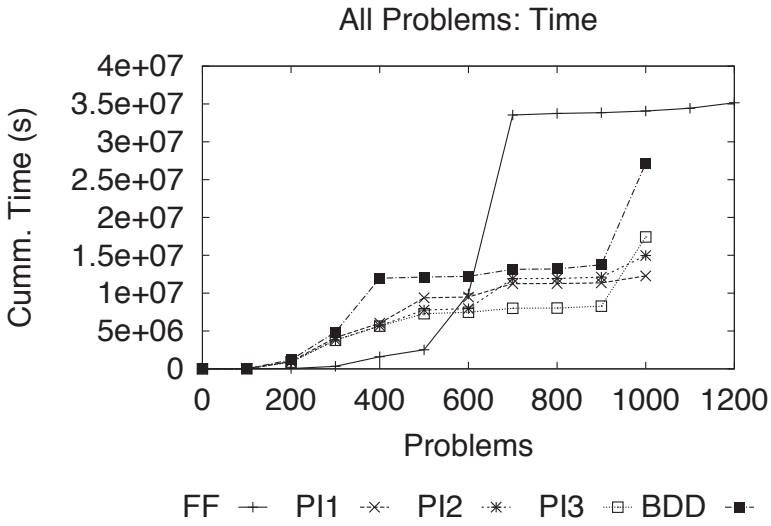

Fig. 2.2: Cumulative time in all domains.

2.8.4 On-line Planning and Execution Results

So that we may judge whether planning and execution strategies such as that of CA benefit when the planner reasons about incompleteness, we provide Figures 2.3, 2.4, and 2.5. These scatter plots are comparisons between Goalie using `DeFault-`FF and `DeFault-`$PI1$ to synthesize plans, contrasting the respective:

Figure 2.3 – number of actions applied to achieve the goal;

Figure 2.4 – number of plans generated; and

Figure 2.5 – total planning and execution time.

Q4 is answered mostly positively. By investigating the plots of the number of actions taken (Figure 2.3) and the number of plans generated (Figure 2.4), it is apparent that `DeFault-`$PI1$ takes fewer actions as the instances minimum required steps increases, and it tends to fail and replan less often. The plot of the total time taken (Figure 2.5) shows that the planners are somewhat mixed or even for times less than 10 seconds. However, for times greater than 10 seconds, it appears that using `DeFault-`FF in `Goalie` can take up to an order of magnitude less time. However, there are several difficult instances in which `DeFault-`$PI1$ does outperform `DeFault-`FF.

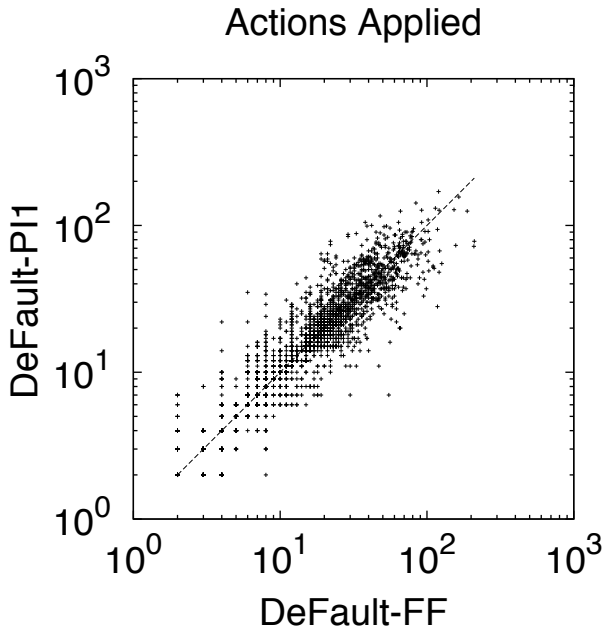

Fig. 2.3: Total number of actions comparison, `Goalie` with `DeFault-`FF vs. `Goalie` with `DeFault-`$PI1$.

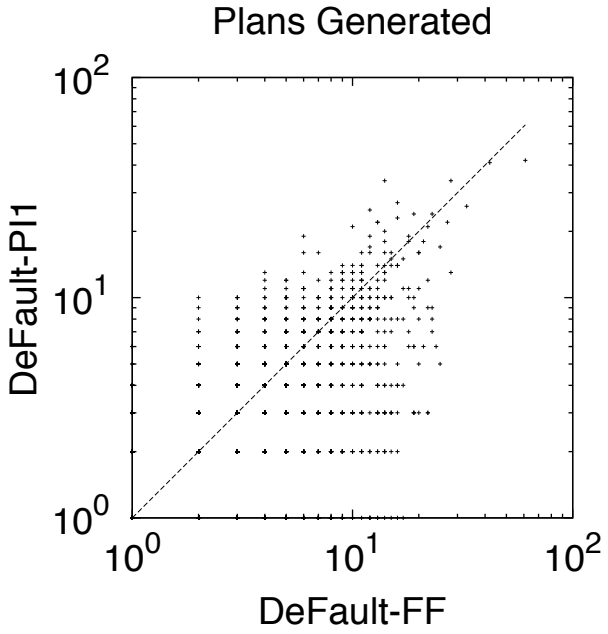

Fig. 2.4: Total number of plans generated comparison, `Goalie` with `DeFault-`FF vs. `Goalie` with `DeFault-`$PI1$.

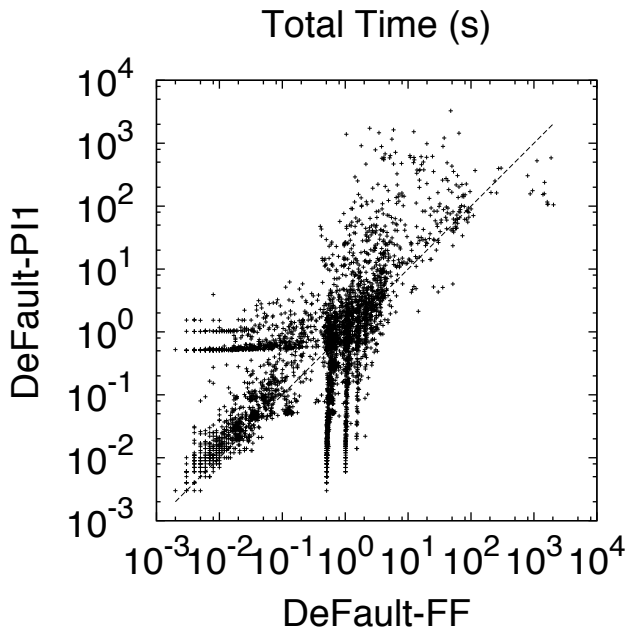

Fig. 2.5: Total execution time comparison, `Goalie` with `DeFault`-FF vs. `Goalie` with `DeFault`-$PI1$.

We expect that more efficient implementations of reasoning about prime implicants (e.g., tries) could lower the cost of planning with `DeFault`, thus allowing it to capitalize on its more robust plans.

2.9 Related Work

Planning in incomplete domains is noticeably similar to planning with incomplete information. However, for incomplete domains it is the actions, not the states, that are incomplete. Incomplete domains can be translated to conformant probabilistic planning domains, and planners such as POND (Bryce, Kambhampati, and Smith 2008) and PFF (Domshlak and Hoffmann 2007) are applicable. However, while the translation is theoretically feasible, current CPP planners are not reasonable approaches to the problem (Bryce 2011).

Our investigation is an instantiation of model-lite planning (Kambhampati 2007). As pointed out by Kambhampati (2007), constraint-based hierarchical task networks are an

alternative that avoid specifying all preconditions and effects through methods and constraints corresponding to underlying, implicit causal links.

As previously stated, this work is a natural extension of the Garland and Lesh (2002) model for evaluating plans in incomplete domains. We note that their enhanced-STRIPS formulation of incomplete domains has come to define the term "incomplete domains" as a research area. Our methods for computing plan failure explanations are different in that we compute them in the forward direction and allow for multiple, interacting faults instead of single faults. In addition to calculating the failure explanations of partial plans, we use a relaxed planning heuristic informed by failure explanations.

Prior work of Chang and Amir (2006) addresses planning with incomplete models, but does not attempt to synthesize robust plans, which is similar to our `DeFault`-FF planner. We have shown that incorporating knowledge about domain incompleteness into the planner can lead to an agent that replans less and thus fails less often. We also differ from CA in that:

i) we do not assume direct feedback from the environment about action failures; and

ii) we are able to learn about action preconditions.

2.10 Conclusion

We have presented the first research to utilize heuristic search to find robust plans for incomplete domains. Our planner `DeFault`:

i) performs forward search while maintaining plan failure explanations; and

ii) estimates the future failures by propagating failure explanations on planning graphs.

We have shown that, compared to a configuration that essentially ignores aspects of the incomplete domain, `DeFault` scales reasonably well while finding much higher quality plans. We have also shown that representing plan failure explanations with prime implicants leads to better scalability than complete model-counting (using OBDDs). Our agent `Goalie` uses `DeFault` to generate robust plans that fail less often and require less replanning than more optimistic planning approaches that ignore incompleteness.

CHAPTER 3
GOAL-DIRECTED KNOWLEDGE ACQUISITION

3.1 Abstract

Agents with incomplete knowledge of their actions can either plan around the incompleteness, ask questions of a domain expert, or learn through trial and error. We present and evaluate several approaches to formulating plans under incomplete information, using the plans to identify relevant (goal-directed) questions, and interleaving acting, planning, and question asking.

We show that goal-directed knowledge acquisition leads to fewer questions and lower overall planning and replanning time than naive approaches that ask many questions, or learn by trial and error. Moreover, we show that prioritizing questions based on plan failure diagnoses leads to fewer questions on average to formulate a successful plan.

3.2 Introduction

Knowledge engineering (Bertoli, Botea, and Fratini 2009) and machine learning (Wu, Yang, and Jiang 2007; Oates and Cohen 1996) have been applied to constructing representations for planning, but pose intensive human and/or data requirements, only to leave a potential mismatch between the environment and model (Kambhampati 2007). Recently, Weber and Bryce (see Chapter 2) showed that instead of placing effort upon making domains complete it is possible to plan with incomplete knowledge of an agent's action descriptions (i.e., plan around the incompleteness). Agents executing such robust plans fail and replan less often, achieving their goals in less overall time than agents that ignore incompleteness when planning (Chang and Amir 2006). While Weber and Bryce (see Chapter 2) demonstrate that planning in incomplete domains can help agents learn about domains, they ignore cases wherein domain experts are available to help engineer the agent's knowledge. We extend the work found in Chapter 2 (including their `DeFault` planner) to consider

agents that can query a domain expert, as in instructable computing (Mailler et al. 2009), but carefully select their questions for knowledge acquisition (KA) in a goal-directed fashion to reduce domain expert fatigue and overall task completion time.

Selecting questions is a problem that has been studied in areas such as preference elicitation (Boutilier 2002), machine learning (Gervasio, Yeh, and Myers 2011), and model-based diagnosis (de Kleer, Mackworth, and Reiter 1992). KA in planning with incomplete domain knowledge is unique in that plans have a rich causal structure that makes questions highly coupled, frequent replanning can change which questions are relevant, and planning relaxations provide opportunities for selecting relevant questions at a fraction of the cost of analyzing full plans.

Our approach to formulating questions relies on planning with incomplete information, deriving a plan failure explanation (set of diagnoses), and ranking questions based upon these diagnoses so that questions are goal-directed. Upon finding a plan that will provably succeed, our agent executes the plan to achieve its goals. We compare our methods with simpler strategies that either ask questions about all possible incomplete domain features prior to planning, or ask no questions and learn by trial and error. This work investigates several issues, including whether:

i) planning with incompleteness is helpful when agents have the ability to ask questions;
ii) how plans can effectively support goal-directed knowledge acquisition; and
iii) relaxed plans provide the same benefit of actual plans in guiding knowledge acquisition.

We find that, indeed:

i) planning with incompleteness leads to fewer required questions and lower overall time to solve a task;
ii) ranking goal-directed questions by a measure of their one-step Shannon entropy is the most effective method to knowledge acquisition; and
iii) relaxed plans improve the speed with which goal-directed questions are identified with no impact on agent performance.

Our presentation includes a discussion of incomplete STRIPS, belief maintenance and planning in incomplete domains, strategies for KA (goal-directed and otherwise), an empirical evaluation in several domains, related work, and a conclusion.

3.3 Background and Representation

Incomplete STRIPS minimally relaxes the classical STRIPS model to allow for possible preconditions and effects (Garland and Lesh 2002). Incomplete STRIPS domains are identical to STRIPS domains, with the exception that the actions are incompletely specified. Much like planning with incomplete state information (Bonet and Geffner 2000), the action incompleteness is not completely unbounded. The preconditions and effects of each action can be any subset of the propositions P; the incompleteness is with regard to a lack of knowledge about which of the subsets corresponds to each precondition and effect.

3.3.1 Incomplete STRIPS Domains

An incomplete STRIPS domain $\tilde{D}$ defines the tuple (P, $\tilde{A}$, I, G, F), where:

P is a set of propositions;

$\tilde{A}$ is a set of incomplete action descriptions;

$I \subseteq P$ defines a set of initially true propositions;

$G \subseteq P$ defines the goal propositions; and

F is a set of propositions describing incomplete domain features.

Each action $a \in A$ defines:

$pre(a) \subseteq P$, a set of known preconditions;

$add(a) \subseteq P$, a set of known add effects; and

$del(a) \subseteq P$, a set of known delete effects.

The set of incomplete domain features F is comprised of propositions of the form:

$pre(a, p)$, a possible precondition of a;

$add(a, p)$, a possible add effect of a; and

$del(a, p)$, a possible delete effect of a.

Consider the following incomplete domain:

$P = \{p, q, r, g\}$;

$A = \{a, b, c\}$;

$I = \{p, q\}$;

$G = \{g\}$; and

$F = \{pre(a, r),\ add(a, r),\ del(a, p),\ del(b, q),\ pre(c, q)\}$.

The known features of the actions are defined:

$pre(a) = \{p, q\}$;

$pre(b) = \{r\},\ add(b) = \{r\}$; and

$pre(c) = \{r\},\ add(c) = \{g\}$.

An interpretation $F^i \subseteq F$ of the incomplete STRIPS domain defines a STRIPS domain, in that every feature $f \in F^i$ indicates that a possible precondition or effect is a respective known precondition or known effect; those features not in F^i are not preconditions or effects.

3.3.2 Incomplete STRIPS Plans

A plan π for D is a sequence of actions that when applied *can lead* to a state in which the goal is satisfied. A plan $\pi = (a_0, ..., a_{n-1})$ in an incomplete domain D is a sequence of actions that corresponds to the *optimistic* sequence of states $(s_0, ..., s_n)$, where:

$s_0 = I$;

$pre(a_t) \subseteq s_t$ for $t = 0, ..., n$;

$G \subseteq s_n$; and

$s_{t+1} = s_t \backslash del(a_t) \cup add(a_t) \cup \{p | add(a, p) \in F\}$ for $t = 0, ..., n - 1$.

For example, the plan (a, b, c) corresponds to the state sequence

$(s_0 = \{p, q\},\ s_1 = \{p, q, r\},\ s_2 = \{q, r\},\ s_3 = \{q, r, g\})$,

where the goal is satisfied in s_3. We note that $r \in s_1$ even though r is only a possible add effect of a; without listing r in s_1, the known precondition of b would not be satisfied. It is certainly possible that in the true domain r is not an add effect of a. However, in the

absence of contrary information, we optimistically assume r is an add effect so that we can synthesize a plan. Pessimistically disallowing such plans is admissible, but constraining, and we prefer to find a plan that may work to finding no plan at all. Naturally, we prefer plans that succeed under more interpretations.

3.4 Belief Maintenance and Planning

An agent can act, ask questions, and plan. Acting and asking a question provide observations of the incomplete domain, and planning involves predicting future states (in the absence of observations). In the following, we discuss how observations can be filtered to update an agent's knowledge ϕ (defined over the literals of F), and what can be assumed about predicted states (when taking knowledge into account). We denote by $d(\pi)$ a plan's failure explanations/diagnoses, which are represented by a propositional sentence over F.

We use ϕ to reason about actions and plans by making queries of the form:

$\phi \models add(a, p)$ – Is p a known add effect of a?;

$\phi \not\models add(a, p)$ and $\phi \not\models \neg add(a, p)$ – Is p a possible/unknown add effect of a?; or

$\phi \models d(\pi)$ – Is the current knowledge consistent with every interpretation where π is guaranteed to fail?).

It is often the case that it is unknown if an incomplete feature $\mathsf{f} \in F$ exists in the true domain that is consistent with ϕ (i.e., $\phi \not\models f$ and $\phi \not\models \neg f$), and we denote this by "$\phi?f$".

3.4.1 Filtering Observations

An agent that acts in incomplete STRIPS domains starts with no knowledge of the incomplete features (i.e., $\phi = \top$). However, taking actions provides state transition observations of the form $o(s, a, s')$, and asking questions (i.e., "Is f true or false?") provides observations of the form f or $\neg f$.

Thus, the function `filter` returns the updated knowledge ϕ' after an observation, and is defined:

$$\texttt{filter}(\phi, f) = \phi \wedge f$$

$$\texttt{filter}(\phi, \neg f) = \phi \wedge \neg f$$

$$\texttt{filter}(\phi, o(s, a, s)) = \phi \wedge ((fail \wedge o^-) \vee o^+)$$

$$\texttt{filter}(\phi, o(s, a, s')) = \phi \wedge o^+$$

where:

$$\begin{aligned}
o^- &= \bigvee_{\substack{pre(a,p)\in F:\\ p\notin s}} pre(a,p) \\
o^+ &= o^{pre} \wedge o^{add} \wedge o^{del} \\
o^{pre} &= \bigwedge_{\substack{pre(a,p)\in F:\\ p\notin s}} \neg pre(a,p) \\
o^{add} &= \bigwedge_{\substack{add(a,p)\in F:\\ p\in s'\setminus s}} add(a,p) \wedge \bigwedge_{\substack{add(a,p)\in F:\\ p\notin s\cup s'}} \neg add(a,p) \\
o^{del} &= \bigwedge_{\substack{del(a,p)\in \mathsf{F}:\\ p\in s\setminus s'}} del(a,p) \wedge \bigwedge_{\substack{del(a,p)\in F:\\ p\in s\cap s'}} \neg del(a,p)
\end{aligned}$$

We assume that the state will remain unchanged upon executing an action whose precondition is not satisfied. As the state is observable, $\texttt{filter}(\phi, o(s, a, s))$ references the case wherein the state does not change, and $\texttt{filter}(\phi, o(s, a, s'))$ references the case wherein the state does change. If the state does not change, then either the action failed (o^-) and one of its unsatisfied possible preconditions is a precondition, or the action succeeded (o^+). We use the $fail$ proposition to denote interpretations under which a plan failed because it is not always observable that the plan has failed. If the state changes, then the agent knows that the action succeeded. If an action succeeds, the agent can conclude that:

i) each possible precondition that was not satisfied is not a precondition (o^{pre});

ii) each possible add effect that appears in the successor but not the predecessor state is an add effect, and each that does not appear in either state is not an add effect (o^{add});

iii) each possible delete effect that appears in the predecessor but not the successor is a delete effect, and each that appears in both states is not (o^{del}).

3.5 Planning

We label predicted state propositions and actions with domain interpretations that will respectively fail to achieve the proposition or fail to achieve the preconditions of an action. That is, labels indicate the cases in which a proposition will be false; i.e., the plan fails to establish the proposition. Labels $d(\cdot)$ are represented as propositional sentences over F whose models correspond to failed domain interpretations.

Initially, each proposition $p_0 \in s_0$, in the state from which a plan is generated, is labeled $d(p_0) = \bot$ to denote that there are no interpretations in the current state where a proposition may be false (the state is fully-observable), and each $p_0 \notin s_0$ is labeled $d(p_0) = \top$ to denote they are known false. For all $t \geq 0$, we define:

$$d(a_t) = d(a_{t-1}) \vee \bigvee_{\substack{p \in pre(a) \text{ or} \\ \phi \models pre(a,p)}} d(p_t) \vee \bigvee_{p:\phi?pre(a,p)} (d(p_t) \wedge pre(a_t, p))$$

$$d(p_{t+1}) = \begin{cases} d(p_t) \wedge d(a_t) & : p \in add(a_t) \text{ or } \phi \models add(a_t, p) \\ d(p_t) \wedge (d(a_t) \vee \neg add(a_t, p)) & : \phi?add(a_t, p)) \\ \top & : p \in del(a_t) \text{ or } \phi \models del(a_t, p) \\ d(p_t) \vee del(a_t, p) & : \phi?del(a_t, p) \\ d(p_t) & : otherwise \end{cases}$$

where $d(a_{-1}) = \bot$. The intuition behind the label propagation is that an action will fail in the domain interpretations $d(a_t)$ wherein a prior action failed, a known precondition is not satisfied, or a possible precondition is not satisfied. As defined for $d(p_{t+1})$, the plan will fail to achieve a proposition at time $t + 1$ in all interpretations wherein:

i) the plan fails to achieve the proposition at time t and the action fails;

ii) the plan fails to achieve the proposition at time t and the action fails or it does not add the proposition in the interpretation;

iii) the action deletes the proposition;

iv) the plan fails to achieve the proposition at time t or in the interpretation the action deletes the proposition; or

v) the action does not affect the proposition and prior failures apply.

A consequence of our definition of action failure is that each action fails if any prior action fails. This definition follows from the semantics that the state becomes undefined if we apply an action whose preconditions are not satisfied. While we use this notion in plan synthesis, we explore the semantics that the state does not change (i.e., it is defined) upon failure when acting in incomplete domains. The pragmatic reason that we define action failures in this manner is that we can determine all failed interpretations affecting a plan $d(\pi)$, by defining

$$d(\pi) = d(a_{n-1}) \vee \bigvee_{p \in G} d(p_n).$$

That is, failure to execute an action is propagated to a failure to achieve the goal. For example, our plan example from the previous section has the failure explanation label

$$d(\pi) = pre(a, r) \vee del(a, p) \vee (del(b, q) \wedge pre(c, q)).$$

3.5.1 Incomplete Domain Relaxed Plans

The `DeFault` planner (see Chapter 2) guides its expansion of plans that are labeled with failure explanations by computing relaxed plans with failure explanations. We also use such relaxed plan failure expalantions to select questions. Finding a relaxed plan that attempts to minimize failure explanations involves propagating failed interpretation labels in a planning graph. Propagating labels relies on selecting an action to support each proposition, and we select supporter $a_{t+k}(p)$ at step k of the planning graph for state s_t with the fewest failed interpretations, denoted by its label $\hat{d}(a_{t+k}(p))$.

A relaxed planning graph with propagated labels is a layered graph of sets of vertices of the form $(\hat{\mathcal{P}}_t, \hat{\mathcal{A}}_t, ..., \hat{\mathcal{A}}_{t+m}, \hat{\mathcal{P}}_{t+m+1})$. The relaxed planning graph built for state $\tilde{s}_t$ defines:

$\mathcal{P}_0 = \{p_t | p \in s_t\}$;

$\mathcal{A}_{t+k} = \{a_{t+k} | \forall_{p \in pre(a)} p_{t+k} \in \mathcal{P}_{t+k}, a \in A \cup A(P)\}$; and

$\mathcal{P}_{t+k+1} = \{p_{t+k+1} | a_{t+k} \in \mathcal{A}_{t+k}, p \in add(a) \cup \{p | \phi \not\models \neg add(a, p)\}$;

for $k = 0, ..., m$. Much like the successor function used to compute next states, the relaxed planning graph assumes optimistic semantics for action effects by adding possible add effects to proposition layers. In addition, as explained below, it associates failed interpretations with the possible adds.

Each planning graph vertex has a label, denoted $\hat{d}(\cdot)$. The failed interpretations $\hat{d}(p_t)$ affecting a proposition are defined such that:

$$\hat{d}(p_t) = d(p_t);$$

and for $k \geq 0$,

$$\hat{d}(a_{t+k}) = \bigvee_{\substack{p \in pre(a) \text{ or} \\ \phi \models pre(a,p)}} \hat{d}(p_{t+k}) \vee \bigvee_{\phi ? pre(a,p)} (\hat{d}(p_{t+k}) \wedge pre(a,p))$$

$$\hat{d}(p_{t+k+1}) = \begin{cases} \hat{d}(\hat{a}_{t+k}(p)) & : p \in add(\hat{a}_{t+k}(p)) \text{ or } \phi \models add(\hat{a}_{t+k}(p), p) \\ \hat{d}(\hat{a}_{t+k}(p)) \vee \neg add(\hat{a}_{t+k}(p), p) & : \phi ? add(\hat{a}_{t+k}(p), p) \end{cases}$$

Every action in every level k of the planning graph will fail in any interpretation in which their preconditions are not supported. A proposition will fail to be achieved in any interpretation in which the chosen supporting action fails to add the proposition.

The relaxed planning graph expansion terminates at the level $t + k + 1$ wherein the goals have been reached at $t+k+1$. The $h^{\sim FF}$ heuristic makes use of the chosen supporting action $\hat{a}_{t+k}(p)$ for each proposition that requires support in the relaxed plan, and, hence, measures the number of actions used while attempting to minimize failed interpretations. The failure explanation of the relaxed plan is defined by

$$d(\hat{\pi}) = \bigvee_{p \in G} \hat{d}(p_{t+m+1}).$$

3.6 Goal-Directed Knowledge Acquisition

We describe several approaches to formulating questions for a domain expert in incomplete domains, which include asking all possible questions in an uninformed manner, asking no questions but learning by trial and error, and using plans to select goal directed questions. We discuss the approaches by increasing sophistication and follow with an empirical evaluation.

3.6.1 Uninformed QA

Uninformed QA (UQA) is the strategy taken by an agent that cannot plan, will not plan, or will not act under uncertainty. Thus, the agent must ask the domain expert about each incomplete feature of the domain without regard to its relevance to goal achievement. As such, uninformed QA-based agents ask a set of questions $Q_F = F$ about every feature in F, formulate a classical plan, and execute the plan.

3.6.2 Non-QA

Non-QA is the strategy taken by an agent that would rather act under uncertainty and ask no questions of the domain expert. Non-QA agents are potentially reckless because learning about the domain sometimes requires that they apply actions whose preconditions may be unsatisfied.

Algorithm 2 is the strategy used by the non-QA agent. The algorithm involves initializing the agent's knowledge and plan (lines 1 and 2). Then, while the plan is non-empty and the goal is not achieved (line 3), the agent proceeds as follows.

Algorithm 2: Non-QA(s, G, A)

Input: state s, goal G, actions A

1 $\phi \leftarrow \top$;
2 $\pi \leftarrow \texttt{plan}(s, G, A, \phi)$;
3 **while** $\pi \neq ()$ *and* $G \not\subseteq s$ **do**
4 $\quad a \leftarrow \pi.first()$;
5 $\quad \pi \leftarrow \pi.rest()$;
6 $\quad$ **if** $pre(a) \subseteq s$ *and* $\phi \not\models d(\pi)$ **then**
7 $\quad\quad s' \leftarrow Execute(a)$;
8 $\quad\quad \phi \leftarrow \texttt{filter}(\phi, o(s, a, s'))$;
9 $\quad\quad s \leftarrow s'$;
10 $\quad$ **else**
11 $\quad\quad \phi \leftarrow \phi \wedge fail$;
12 $\quad$ **end**
13 $\quad$ **if** $\phi \models fail$ **then**
14 $\quad\quad \phi \leftarrow \exists_{fail}\phi$;
15 $\quad\quad \pi \leftarrow \texttt{plan}(s, G, A, \phi)$;
16 $\quad$ **end**
17 **end**

The agent selects the next action in the plan (line 4) and determines if it can apply the action (line 6). If it applies the action, the next state is returned by the environment/simulator (line 7) and the agent updates its knowledge (line 8) and state (line 9); otherwise, the agent determines that the plan will fail (line 11). If the plan has failed (line 13), the agent forgets its knowledge of the plan failure by projecting over $fail$ (line 14) and finds a new plan using its new knowledge (line 15).

Using ϕ, it is possible to determine if the next action in a plan, or any subsequent action, can or will fail. If $\phi \wedge d(\pi)$ is satisfiable, then π *can* fail, and if $\phi \models d(\pi)$, then π *will* fail. For example, upon executing a, the non-QA agent might observe the state transition

$o_1 = o(\{p, q\}, a, \{p, q\})$, and thus

$\phi' = \texttt{filter}(\phi, o_1) = \neg del(a, r)$.

The agent must replan because $\phi' \models d(\pi)$.

3.6.3 Goal-Directed QA

Similar to the non-QA agent, goal-directed QA agents plan under uncertainty, but they differ in that they ask about action features that are relevant to the plan. There are a number of strategies for using a plan to generate questions, and we assume that the goal-directed QA agent continues to ask questions and replan until it finds a plan that is guaranteed to succeed. The strategies include:

i) asking about all incomplete features related to actions in a plan;

ii) asking about only those features that can cause failure; and

iii) ranking the questions based on the diagnoses of plan failure to "fail fast."

3.6.4 Plan Relevant Questions

Without computing failure explanations, it is possible to determine relevant questions by inspecting the actions in the plan and their possible preconditions and effects such that the set of questions is defined:

$$Q_\pi = \{pre(a_t, p) \mid \phi?pre(a_t, p), a_t \in \pi\} \cup$$
$$\{add(a_t, p) \mid \phi?add(a_t, p), a_t \in \pi\} \cup$$
$$\{del(a_t, p) \mid \phi?del(a_t, p), a_t \in \pi\}$$

By using Q_π and no plan failure explanation, an agent is relegated to asking each question because it cannot determine if the plan will fail given the answers it has received thus far. The agent can only replan afterward and determine if the plan contains actions with incomplete features. The example plan would lead to $Q_\pi = F$ because the incomplete feature in F relates to an action in the plan. However, as we saw in $d(\pi)$, $add(a, r)$ is irrelevant because the plan will succeed no matter its value.

3.6.5 Plan Failure Diagnosis Relevant Questions

A question is relevant to a plan π if the incomplete feature f is entailed by a potential diagnosis δ of plan failure. Each diagnosis δ of the plan failure explanation $d(\pi)$ is a conjunction of incomplete features that must interact to destroy the plan. Thus, if $\delta \models d(\pi)$ and $\delta \models \mathsf{f}$, then

$$Q_{d(\pi)} = \{\mathsf{f} | \delta \models d(\pi), \delta \models \mathsf{f} \text{ or } \delta \models \neg\mathsf{f}\}.$$

The example plan yields

$$Q_{d(\pi)} = \{pre(a, r),\ del(a, p),\ del(b, q),\ pre(c, q)\}$$

because each feature appears in a diagnosis.

3.6.6 Ranking Relevant Questions

The features in smaller cardinality diagnoses have a greater impact on the plan because a smaller number of unfavorable answers are needed to prove the plan will fail; asking about these features will enable an agent to fail fast. Moreover, features appearing in more diagnoses have a high impact on plan failure. Using this *diagnosis-impact* (DI) measure, we can prioritize by selecting the f where:

$$\mathsf{f} = \underset{\mathsf{f} \in Q_{d(\pi)}}{\operatorname{argmax}} \sum_{\substack{\delta : \delta \models d(\pi), \\ \delta \models \mathsf{f}}} \frac{1}{|\{\mathsf{f} | \delta \models \mathsf{f}\}|^2}$$

The denominator of the expression above is squared to penalize the contribution of larger diagnoses. DI determines the incomplete feature most likely to cause the plan to fail. Using DI for the example plan questions will select $pre(a, r)$ and $del(a, p)$ as equally preferred questions because both appear in a size one diagnosis.

An alternative, based on *Shannon entropy* (SE), selects the question f where:

$$\mathsf{f} = \underset{\mathsf{f} \in Q_{d(\pi)}}{\operatorname{argmin}} - \sum_{\mathsf{f} \in \{\mathsf{f}, \neg \mathsf{f}\}} p_{\mathsf{f}} \log p_{\mathsf{f}}$$

$$p_{\mathsf{f}} = |M(\texttt{filter}(\phi, \mathsf{f}) \wedge d(\pi))| / 2^{|F|}$$

and $M(\cdot)$ denotes the set of propositional models of some sentence. The probability p_{f} that the plan fails under the true domain model is the proportion of propositional models wherein after filtering f the plan will fail. If it is the case that $\texttt{filter}(\phi, \mathsf{f}) \models d(\pi)$, then π will fail and the agent must replan; when computing p_{f} and the plan fails, we compute a new plan π' given the knowledge $\phi' = \texttt{filter}(\phi, \mathsf{f})$ and replace $d(\pi)$ with $d(\pi')$. If no plan π' exists, we define $p_{\mathsf{f}} = 1$ to denote that the absence of a plan indicates failure. For example, both $pre(a, r)$ and $del(a, p)$ are equally preferred questions because their entropy is 0.16, whereas both $del(b, q)$ and $pre(c, q)$ have entropy 0.32.

3.7 Empirical Evaluation

The empirical evaluation is divided into three sections: the domains used for the experiments, the test setup used, and results. The questions that we seek to answer include:

Q1 – Does reasoning about incompleteness effect KA strategy performance?

Q2 – Which KA strategy has best solution time and number of questions trade-off?

Q3 – Do relaxed plans reduce the cost of KA?

3.7.1 Domains

We use the same four domains as presented in Chapter 2 for our evaluation: a modified Pathways, Bridges, a modified PARC Printer, and Barter World. Again, in all domains, we derived multiple instances by randomly (with probabilities 0.25, 0.5, 0.75, and 1.0 for each action) injecting incomplete features. Such variations of the domains create instances that include up to 10,000 incomplete features each. All results are taken from 10 random instances (varying F) of each problem and three ground-truth domains selected by the simulator. For the reader's convenience, we again present the descriptions of these four domains.

The Pathways (PW) domain from the International Planning Competition (IPC) involves actions that model chemical reactions in signal transduction pathways. Pathways is a naturally incomplete domain where the lack of knowledge of the reactions is quite common, and are an active research topic in biology.

The Bridges (BR) domain consists of a traversable grid wherein the task is to find a different treasure at each corner of the grid. Grids are square and vary in dimension (2-16). Again, as presented in Chapter 2, there are three versions (summed in these results):

BR1 – a bridge might be required to cross between some grid locations (a possible precondition);

BR2 – BR1, plus many of the bridges may have a troll living underneath that will take all the treasure accumulated (a possible delete effect);

BR3 – BR2, plus the corners may give additional treasures (possible add effects).

The PARC Printer (PP) domain from the IPC involves planning paths for sheets of paper through a modular printer. A source of domain incompleteness is that a module accepts only certain paper sizes, but its documentation is incomplete. Thus, in using the module, paper size becomes a possible precondition to actions.

The Barter World (BW) domain involves navigating a grid and bartering items to travel between locations. The domain is incomplete because actions that acquire items are not always known to be successful (possible add effects) and traveling between locations

may require certain items (possible preconditions) and also may result in the loss of an item (possible delete effects). Grids vary in dimension (2-16) and items in number (1-4).

3.7.2 Test Setup

The tests were run on a Linux machine with a 3 Ghz Xeon processor, a memory limit of 2GB, and a 60-minute time limit for each simulation instance. All code is written in Java and run on the 1.6 JVM. `DeFault` uses a greedy best-first search with deferred heuristic evaluation and a dual-queue for preferred and nonpreferred operators (Helmert 2006).

3.7.3 Results

Table 3.1 shows the performance ratio for FF versus `DeFault` as our agent's planner; a result greater than 1.0 indicates that `DeFault` is the better performer. Table 3.2 contrasts the performance of non-QA against uniformed QA. Table 3.3 compares the various KA strategies to each other. Lastly, Table 3.4 shows the performance ratio of relaxed plans over fully-formed plans as the source for question derivation, wherein a result greater than 1.0 indicates that the relaxed plan is the better performer.

Table 3.1: FF Vs. `DeFault` Performance Ratio (Number Solved / Number of Steps / Time / Questions).

Domain	Non-QA	SE
BR 0.25	1.0/1.2/6.4/-	1.0/1.0/23.4/3.8
BR 0.5	0.9/1.2/4.5/-	1.1/1.0/4.4/1.7
BR 0.75	1.0/1.0/3.9/-	1.2/1.0/1.3/1.2
BR 1.0	1.1/1.0/1.3/-	1.3/1.0/2.4/1.2
PW 0.25	1.0/1.0/1.7/-	1.3/0.9/3.7/1.6
PW 0.5	0.5/1.1/1.6/-	1.5/1.1/6.5/1.7
PW 0.75	0.7/1.1/1.8/-	1.2/1.0/1.6/1.1
PW 1.0	0.8/0.9/0.8/-	1.1/1.0/1.1/1.1
PP 0.25	0.8/1.0/2.3/-	1.1/1.0/2.2/1.1
PP 0.5	2.3/1.0/3.6/-	1.4/1.0/0.9/0.9
PP 0.75	-/-/-/-	2.1/1.0/1.0/1.0
PP 1.0	-/-/-/-	1.9/1.1/0.6/0.8
BW 0.25	0.8/1.2/9.7/-	1.0/0.9/18.5/3.3
BW 0.5	0.7/1.2/12.2/-	1.0/1.0/18.9/2.3
BW 0.75	0.7/1.0/11.5/-	1.2/0.9/8.8/1.4
BW 1.0	0.5/0.9/27.3/-	1.5/0.9/14.0/1.3

Table 3.2: Extreme Strategy Average Performance (Number Solved / Number of Steps / Time (seconds) / Questions). **Bold** Indicates Best Performers.

Domain	Non-QA	Q_F
BR 0.25	271/22.1/1.6/-	**274**/**20.2**/**0.5**/57.5
BR 0.5	199/27.1/3.0/-	**251**/**17.9**/**0.4**/114.1
BR 0.75	131/18.2/1.9/-	**200**/**13.5**/**0.2**/65.7
BR 1.0	89/**12.0**/0.9/-	**190**/12.1/**0.2**/107.8
PW 0.25	91/**13.5**/0.4/-	**121**/16.8/**0.3**/28.9
PW 0.5	80/**21.4**/4.5/-	**180**/27.2/**0.5**/75.8
PW 0.75	70/**12.4**/0.5/-	**130**/15.2/**0.3**/73.2
PW 1.0	40/**11.2**/0.5/-	**127**/14.5/**0.3**/104.3
PP 0.25	37/**9.5**/**0.3**/-	**151**/11.0/0.4/36.4
PP 0.5	6/**9.3**/**0.4**/-	**151**/11.0/0.4/75.3
PP 0.75	-/-/-/-	**150**/**11.0**/**0.4**/109.4
PP 1.0	-/-/-/-	**150**/**11.0**/**0.4**/145.2
BW 0.25	276/13.6/4.7/-	**321**/**11.5**/**0.8**/232.8
BW 0.5	179/11.0/7.7/-	**239**/**9.2**/**0.4**/205.7
BW 0.75	118/12.1/2.6/-	**207**/**8.1**/**0.3**/172.8
BW 1.0	11/12.5/3.6/-	**20**/**8.6**/**0.3**/189.8

Table 3.3: Goal-Directed KA Average Performance Using `DeFault` (Number Solved / Number of Steps / Time (seconds) / Questions).

Domain	Q_π	$Q_{d(\pi)}$	DI	SE
BR 0.25	**272**/19.8/1.8/4.1	**272**/**19.8**/**1.8**/2.8	**272**/19.9/1.9/**2.2**	271/20.0/15.4/2.3
BR 0.5	**232**/17.0/**3.1**/18.2	231/17.1/3.7/12.4	226/16.8/4.4/**8.9**	219/**16.1**/146.3/10.3
BR 0.75	**191**/12.3/1.9/24.5	**191**/12.3/**1.9**/17.5	183/12.0/3.1/13.2	171/**11.2**/75.9/**12.6**
BR 1.0	172/11.0/3.6/35.0	169/10.9/**3.4**/22.1	**173**/10.7/3.5/17.1	151/**9.5**/128.9/**15.9**
PW 0.25	**91**/**13.3**/0.5/1.3	**91**/**13.3**/0.4/**1.0**	**91**/**13.3**/**0.3**/**1.0**	**91**/**13.3**/1.1/**1.0**
PW 0.5	130/27.5/10.3/10.9	130/27.5/**10.2**/7.7	**141**/24.8/11.5/5.0	101/**16.4**/42.0/**3.0**
PW 0.75	**130**/15.8/1.8/11.9	**130**/16.3/1.8/7.2	110/**14.5**/**1.6**/**5.6**	110/14.7/54.5/5.9
PW 1.0	**127**/14.6/**1.5**/21.7	**127**/15.2/2.1/12.8	117/14.5/2.6/**10.2**	118/**14.2**/83.6/10.8
PP 0.25	**138**/10.6/1.4/3.5	**138**/10.6/1.3/3.5	136/**10.5**/**1.0**/**2.9**	136/**10.5**/6.1/3.0
PP 0.5	**126**/10.3/1.9/7.9	**126**/10.3/**1.8**/7.9	121/10.1/1.8/**6.0**	110/**9.8**/24.3/6.4
PP 0.75	**83**/9.5/2.3/14.8	**83**/9.5/2.0/14.8	79/9.4/**2.0**/**11.1**	71/**9.1**/29.9/11.8
PP 1.0	**84**/9.1/2.9/21.3	83/9.1/**2.7**/21.1	80/**8.4**/3.8/**17.6**	78/8.4/36.5/18.1
BW 0.25	**321**/**11.9**/10.0/10.9	**321**/12.0/**9.5**/6.5	318/12.6/10.8/**3.7**	320/12.6/116.0/4.9
BW 0.5	**230**/**8.9**/**11.3**/20.7	**230**/9.2/12.2/15.2	227/9.6/15.6/**9.5**	228/9.5/171.1/11.5
BW 0.75	189/**7.4**/**3.2**/31.7	**190**/7.6/4.0/23.9	184/8.4/16.3/14.6	178/7.9/83.2/**14.5**
BW 1.0	13/**5.9**/**2.1**/28.5	14/6.5/3.0/25.6	**15**/7.1/7.9/**17.5**	13/6.3/201.9/20.2

Table 3.4: Plan Vs. Relaxed Plan Ratio in Shannon Entropy Strategy.

Domain	SE
BR 0.25	1/1.0/8.3/1.0
BR 0.5	1/1.0/33.1/1.1
BR 0.75	1/1.0/51.1/1.1
BR 1.0	1/1.0/55.3/1.1
PW 0.25	1/1.0/5.4/1.0
PW 0.5	1/1.0/6.0/1.0
PW 0.75	1/1.0/7.4/1.0
PW 1.0	1/1.0/18.5/1.0
PP 0.25	1/1.0/4.9/1.0
PP 0.5	1/1.0/8.8/1.1
PP 0.75	1/1.0/9.7/1.1
PP 1.0	1/1.0/8.8/1.0
BW 0.25	1/1.0/14.4/1.1
BW 0.5	1/1.0/15.0/1.1
BW 0.75	1/1.0/20.4/1.1
BW 1.0	1/1.0/18.1/0.9

To answer Q1, Table 3.1 compares two planners and two KA strategies. The entries in the table are the ratio of the number of problems solved, number of steps taken by the agent, total time, and questions asked when using a planner that ignores incompleteness (`DeFault` using the FF heuristic (Hoffmann and Nebel 2001)) and a planner that plans with incompleteness (`DeFault`). The columns list results for the non-QA agent and the SE strategy. We see that total time and number of steps is reduced in the non-QA agent when planning with incompleteness (as also found in Chapter 2), and that the total time and number of questions is reduced when using SE. The results demonstrate that planning with incompleteness is beneficial whether asking questions or not.

To answer Q2, Tables 3.2 and 3.3 list the average performance of `DeFault` with various KA strategies. We see that in comparing the two extremes, non-QA or Q_F, Q_F leads to more solved instances and lower total time, but as expected a high number of questions. In comparing the approaches that use a plan to limit and/or bias KA in Table 3.3, we note the number of solved instances is relatively similar, but the methods that prioritize questions based on the plan failure explanations (DI or SE) ask the fewest questions. The total time

taken is relatively similar among the methods, except for a noticeably higher average time with SE – which is due to many hypothesized replanning episodes to compute the entropy.

To answer Q3, and follow up on the high cost of SE, Table 3.4 lists the performance ratios of using SE with actual plans versus relaxed plans when replanning is required. We note that the performance is nearly identical, with the exception of much lower total times – which makes SE competitive with DI.

3.8 Conclusion

We have found that reasoning about incompleteness during planning leads to more effective trial-and-error and KA-based agents. Further, we have shown that plans can be used to help maintain a goal-directed focus on knowledge acquisition and lower the overall strain upon a domain expert to answer many questions. In particular, methods for prioritizing questions based on plan failure explanations are able to greatly reduce the number of questions required to synthesize successful plans. Finally, we have demonstrated that when determining best questions, relaxed plans can be an advantageous substitute for actual plans.

CHAPTER 4
CONCLUSION

Two research papers constitute the main substance of this multi-paper thesis:

1. Planning and Acting in Incomplete Domains
2. Goal-Directed Knowledge Acquisition

These papers compose the bodies of Chapters 2 and 3, respectively.

Their mutual subject is our agent `Goalie` that plans, acts and learns in incomplete domains. In Chapter 3 we saw the learning capability of `Goalie` extended from it's initial presentation in Chapter 2. While in Chapter 2 `Goalie` learns passively about its environment as it acts, in Chapter 3 `Goalie` learns actively via its newly-granted ability to ask questions. We have shown that this extended learning ability allows `Goalie` to perform far more effectively in incomplete domains. In sum, `Goalie`:

i) interleaves acting, planning, and question-asking;
ii) synthesizes plans that avoid execution failure due to ignorance of the domain model;
iii) uses these plans to identify relevant (goal-directed) questions;
iv) passively learns about the domain model during execution to improve later replanning attempts;
v) and employs various targeted (goal-directed) strategies to ask questions (actively learn).

`Goalie`'s planner `DeFault` is the first to utilize a forward state space-based heuristic search that reasons about a domain's incompleteness to avoid potential plan failure. We have shown that `DeFault` performs best by counting prime implicants (failure diagnoses) rather than propositional models. Further, we have shown that by reasoning about incompleteness in planning (as opposed to ignoring it), `Goalie` fails and replans less often, and executes fewer actions. Finally, we have shown that goal-directed knowledge acquisition –

prioritizing questions based on plan failure diagnoses – leads to fewer questions, lower overall planning and replanning time, and higher success rates than approaches that naively ask many questions or learn by trial and error.

REFERENCES

Bertoli, P.; Botea, A.; and Fratini, S. 2009. Introduction. In *Proceedings of ICKEPS, ICAPS'09*.

Blum, A., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *Proceedings of IJCAI'95*, 1636–1642.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of AIPS'00*, 52–61.

Boutilier, C. 2002. A pomdp formulation of preference elicitation problems. In *AAAI/IAAI*, 239–246.

Bryant, R. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–691.

Bryce, D.; Kambhampati, S.; and Smith, D. 2008. Sequential monte carlo in probabilistic planning reachability heuristics. *AIJ* 172(6-7):685–715.

Bryce, D. 2011. Planning in incomplete domains. Technical Report 001, Utah State University. Available at: http://www.cs.usu.edu/∼danbryce/papers/USU-CS-TR-11-001.pdf.

Chang, A., and Amir, E. 2006. Goal achievement in partially known, partially observable domains. In *Proceedings of ICAPS'06*.

Choudhary, A.; Datta, A.; Bittner, M. L.; and Dougherty, E. R. 2006. Intervention in a family of boolean networks. *Bioinformatics* 22(2):226–232.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *JAIR* 17:229–264.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *AIJ* 32(1):97–130.

de Kleer, J.; Mackworth, A. K.; and Reiter, R. 1992. Characterizing diagnoses and systems. *AI* 56(2-3):197–222.

Domshlak, C., and Hoffmann, J. 2007. Probabilistic planning via heuristic forward search and weighted model counting. *JAIR* 30:565–620.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of AAAI'71*, 608–620.

Garland, A., and Lesh, N. 2002. Plan evaluation with incomplete action descriptions. In *Proceedings of AAAI'02*.

Gervasio, M.; Yeh, E.; and Myers, K. 2011. Learning to ask the right questions to help a learner learn. In *Proceedings of the IUI'11*.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

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

Kambhampati, S. 2007. Model-lite planning for the web age masses. In *Proceedings of AAAI'07*.

Mailler, R.; Bryce, D.; Shen, J.; and Orielly, C. 2009. Mable: A framework for natural instruction. In *Proceedings of AAMAS'09*.

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

Oates, T., and Cohen, P. R. 1996. Searching for planning operators with context-dependent and probabilistic effects. In *AAAI/IAAI, Vol. 1*, 863–868.

Roth, D. 1996. On the hardness of approximate reasoning. *AIJ* 82(1-2):273–302.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. Cambridge, MA, USA: The MIT Press.

Vahidi, A. 2011. JDD: Java BDD package. `http://javaddlib.sourceforge.net/jdd/`.

Wu, K.; Yang, Q.; and Jiang, Y. 2007. ARMS: An automatic knowledge engineering tool for learning action models for AI planning. *K. Eng. Rev.* 22(2):135–152.

APPENDIX

April 15, 2011

I, Daniel Morwood, co-author of the paper Goal-Directed Knowledge Acquisition, do hereby give my permission to Christopher H. Weber to use this paper as part of his Utah State University Masters of Science thesis entitled Planning, Acting, and Learning in Incomplete Domains.

Yours,

Daniel Morwood
Utah State University
Department of Computer Science
danmorwood@gmail.com