Goal-Directed Knowledge Acquisition

Paper ID: 441

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.

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, Bryce and Weber (2011) 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 re-plan less often, achieving their goals in less overall time than agents that ignore incompleteness when planning (Chang and Amir 2006). While Bryce and Weber (2011) demonstrate that planning in incomplete domains can help agents learn about domains, they ignore cases where domain experts are available to help engineer the agent's knowledge. We extend the work of Bryce and Weber (2011) (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 problems 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 rich causal structure that makes questions highly coupled, frequent re-planning 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.

Background & Representation

Incomplete STRIPS 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 un-

bounded. 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 correspond to each precondition and effect.

Incomplete STRIPS Domains: An incomplete STRIPS domain D defines the tuple $(P,\ A,\ I,\ G,\ F)$, where: P is a set of propositions, 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 $pre(a,p)$, $add(a,p)$, and $del(a,p)$, each indicating that p is a respective possible precondition, add effect, or 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.

Incomplete STRIPS Plans: A plan π for D is a sequence of actions, that when applied, *can lead* to a state where 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. While it is possible that in the true domain r is not an add effect of a, 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.

Belief Maintenance & 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 is 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 $\mathtt{f} \in F$ 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 "$\phi ? f$".

Filtering Observations: An agent that acts in incomplete STRIPS domains will start 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:

$$\begin{aligned}
\texttt{filter}(\phi, f) &= \phi \wedge f \\
\texttt{filter}(\phi, \neg f) &= \phi \wedge \neg f \\
\texttt{filter}(\phi, o(s,a,s)) &= \phi \wedge ((fail \wedge o^-) \vee o^+) \\
\texttt{filter}(\phi, o(s,a,s')) &= \phi \wedge o^+
\end{aligned}$$

where

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

We assume that the state will remain unchanged upon executing an action whose precondition is not satisfied, and because the state is observable, $\texttt{filter}(\phi, o(s,a,s))$ references the case where the state does not change and $\texttt{filter}(\phi, o(s,a,s'))$, the case where it changes. 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}).

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 where 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:

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

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

where $d(a_{-1}) = \bot$. The intuition behind the label propagation is that an action will fail in the domain interpretations $d(a_t)$ where 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 where 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 $d(\pi) = d(a_{n-1}) \vee \bigvee_{p \in G} d(p_n)$ (i.e., 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 $d(\pi) = pre(a, r) \vee del(a, p) \vee (del(b, q) \wedge pre(c, q))$.

Incomplete Domain Relaxed Plans: The `DeFAULT` planner (Bryce and Weber 2011) 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 the 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 $(\mathcal{P}_t, \mathcal{A}_t, ..., \mathcal{A}_{t+m}, \mathcal{P}_{t+m+1})$. The relaxed planning graph built for a state 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 an optimistic semantics for action effects by adding possible add effects to proposition layers, but, 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 $\hat{d}(p_t)$ affecting a proposition are defined such that $\hat{d}(p_t) = d(p_t)$, and for $k \geq 0$,

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

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

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

The relaxed planning graph expansion terminates at the level $t+k+1$ where 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 $d(\hat{\pi}) = \bigvee_{p \in G} \hat{d}(p_{t+m+1})$.

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.

Uninformed QA: Uninformed QA (UQA) is the strategy taken by an agent that cannot or will not plan or act

vlined 1 Non-QA(s, G, A)

state s, goal G, actions A
$\phi \leftarrow \top$; $\pi \leftarrow \texttt{plan}(s, G, A, \phi)$ $\pi \neq ()$ and $G \not\subseteq s$ $a \leftarrow \pi.first()$; $\pi \leftarrow \pi.rest()$ $pre(a) \subseteq s$ and $\phi \not\models d(\pi)$ $s' \leftarrow Execute(a)$ $\phi \leftarrow \texttt{filter}(\phi, o(s, a, s'))$ $s \leftarrow s'$ $\phi \leftarrow \phi \wedge fail$
$\phi \models fail$ $\phi \leftarrow \exists_{fail}\phi$ $\pi \leftarrow \texttt{plan}(s, G, A, \phi)$

under uncertainty and thus 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 will ask a set of questions $Q_F = F$ about every feature in F, formulate a classical plan, and execute the plan.

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.

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. Algorithm 1 is the strategy used by the Non-QA agent. The algorithm involves initializing the agent's knowledge and plan (line 1), and then while the plan is non-empty and the goal is not achieved (line 2) the agent proceeds as follows. The agent selects the next action in the plan (line 3) and determines if it can apply the action (line 4). If it applies the action, then the next state is returned by the environment/simulator (line 5) and the agent updates its knowledge (line 6) and state (line 7), otherwise the agent determines that the plan will fail (line 9). If the plan has failed (line 11), then the agent forgets its knowledge of the plan failure by projecting over $fail$ (line 12) and finds a new plan using its new knowledge (line 13).

For example, the non-QA agent might observe the state transition $o_1 = o(\{p, q\}, a, \{p, q\})$ upon executing a, and $\phi' = \texttt{filter}(\phi, o_1) = \neg del(a, r)$. The agent must re-plan because $\phi' \models d(\pi)$.

Goal Directed QA:Goal Directed QA agents plan under uncertainty, similar to the Non-QA agent, but 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 re-plan until it finds a plan that is guaranteed to succeed. The strategies include: asking about all incomplete features related to actions in a plan, asking about only those features that can cause failure, and ranking the questions based on the diagnoses of plan failure to "fail fast".

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 so that the set of questions is defined:

$$\begin{aligned} Q_\pi = &\{pre(a_t, p) | \phi ? pre(a_t, p), a_t \in \pi\} \cup \\ &\{add(a_t, p) | \phi ? add(a_t, p), a_t \in \pi\} \cup \\ &\{del(a_t, p) | \phi ? del(a_t, p), a_t \in \pi\} \end{aligned}$$

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 re-plan afterward and determine if the plan contains actions with incomplete features.

The example plan would lead to $Q_\pi = F$ because 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.

Plan Failure Diagnosis Relevant: 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 defines $Q_{d(\pi)} = \{pre(a, r), del(a, p), del(b, q), pre(c, q)\}$ because each feature appears in a diagnosis.

Ranking Relevant Questions: The features in smaller cardinality diagnoses have more 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

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

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

$$\begin{aligned} \mathsf{f} = &\underset{\mathsf{f} \in Q_{d(\pi)}}{\text{argmin}} - \sum_{\mathsf{f} \in \{\mathsf{f}, \neg\mathsf{f}\}} p_\mathsf{f} \log p_\mathsf{f} \\ p_\mathsf{f} = &|M(\texttt{filter}(\phi, \mathsf{f}) \wedge d(\pi))| / 2^{|F|} \end{aligned}$$

and $M(\cdot)$ denotes 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 where 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 re-plan; when computing p_f and the

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 1: FF/DeFAULT Performance Ratio (Num Solved/Num Steps/Time/Questions).

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 2: Extreme Strategy Average Performance. Bold indicates best performance. (Num Solved/Num Steps/Time (s)/Questions).

plan fails, we compute a new plan π' given the knowledge $\phi' = \texttt{filter}(\phi, \texttt{f})$ and replace $d(\pi)$ with $d(\pi')$. If no plan π' exists, we define $p_{\texttt{f}} = 1$ to denote that the absence of a plan indicates failure.

For example, both $pre(a, r)$ and $del(a, p)$ as equally preferred questions because their entropy is 0.16, whereas both $del(b, q)$ and $pre(c, q)$ have entropy 0.32.

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 would like to answer include:

- Q1: Does reasoning about incompleteness during planning 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?

Domains: We use four domains in the evaluation: a modified Pathways, Bridges, a modified PARC Printer, and Barter World (Bryce and Weber 2011). 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. With these variations of the domains, the instances include up to ten thousand incomplete features each. All results are taken from ten random instances (varying F) of each problem and three ground-truth domains selected by the simulator.

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 because they are an active research topic in biology.

The Bridges (Br) domain consists of a traversable grid and the task is to find a different treasure at each corner of the grid. In Bridges, a bridge might be required to cross between some grid locations (a possible precondition), many of the bridges may have a troll living underneath that will take all the treasure accumulated (a possible delete effect), and the corners may give additional treasures (possible add effects). Grids are square and vary in dimension (2-16).

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, paper size becomes a possible precondition to actions using the module.

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

Test Setup: The tests were run on a Linux machine with a 3 Ghz processor, with a 2GB memory limit and 60 minutes time limit for each instance. All code was written in Java and run on the 1.6 JVM. The DeFAULT planner uses a greedy best first search with deferred heuristic evaluation and a dual-queue for preferred and non-preferred operators (Helmert 2006).

Results: To answer Q1, Table 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 by Bryce and Weber (2011)), and that the total time and num-

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: Goal-directed KA Average Performance using `DeFAULT`. Bold indicates best performance. (Num Solved/Num Steps/Time (s)/Questions)

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

Table 4: Plan versus Relaxed Plan Ratio in SE.

ber 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 2 and 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, 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 re-planning episodes to compute the entropy.

To answer Q3, and follow up on the high cost of SE, Table 4 lists the performance ratios of using SE with actual plans versus relaxed plans when re-planning is required. We note that the performance is nearly identical, with the exception of much lower total times – which makes SE competitive with DI.

Conclusion

We have found that reasoning about incompleteness planning leads to more effective trial-and-error agents and KA agents. The plans help maintain a goal-directed focus on knowledge acquisition and lower the overall strain upon a domain expert to answer many questions. We found that methods for prioritizing questions are able to reduce the number of questions required to synthesize successful plans and that relaxed plans serve as a reasonable substitute for actual plans.

References

Bertoli, P.; Botea, A.; and Fratini, S., eds. 2009. *ICKEPS*.

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

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

Bryce, D., and Weber, C. 2011. Planning and acting in incomplete domains. In *Proceedings of ICAPS'11*.

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

de Kleer, J.; Mackworth, A. K.; and Reiter, R. 1992. *Characterizing diagnoses and systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 54–65.

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 *AAMAS'09*.

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

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.