Reactive, Proactive, and Passive Learning about Incomplete Actions

Christopher Weber and **Daniel Bryce**
christopherweber@hotmail.com, daniel.bryce@usu.edu
Department of Computer Science
Utah State University

Abstract

Agents with incomplete knowledge of their actions can either plan around the incompleteness, learn by querying a domain expert, or learn through trial and error. In deciding what to learn, an agent must consider whether an incomplete action feature is relevant to achieving its goals. In deciding how to learn an action feature, the agent can i) try to execute the action and passively observe the outcome, ii) react by querying a domain expert when it fails to learn by passive observation, or iii) proactively query a domain expert prior to executing the action. The challenge is that by learning about incomplete action features an agent may determine its plan will fail and re-plan, and thus change which action features are relevant to achieving it goals. We desire agents that can ask as few questions as possible in achieving their goals.

We present a number of strategies for i) planning with incomplete knowledge of actions to identify relevant incomplete action features (preconditions and effects), ii) reasoning about plan failure explanations to identify which features will be learned passively or proactively, and iii) techniques for diagnosing action failures to reactively learn about actions when passive learning fails. We test the following configurations of our agent: i) learning only passively and asking no questions; ii) asking questions and re-planning until the plan is guaranteed to succeed; iii) planning, acting until the plan fails, diagnosing the failure, and re-planning; and iv) while diagnosing failures, proactively querying about a subset of the future action features that are likely to cause failures. We find that passive learning alone can lead to dead-ends, perfecting a plan prior to execution requires many questions, and balancing passive learning with reactive learning strikes a good balance between avoiding dead-ends and minimizing the number of questions.

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, we (Weber and Bryce 2011) showed that instead of placing effort upon making domains complete it is possible for our planner `DeFAULT` 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 than agents that ignore incompleteness when planning (Chang and Amir 2006). While we demonstrated that planning in incomplete domains can help agents passively learn about domains, we ignore cases where domain experts are available to help engineer the agent's knowledge. We extend our prior work (Weber and Bryce 2011) to consider agents that can query a domain expert, as in instructable computing (Mailler et al. 2009), but must carefully select their questions.

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). Incomplete action knowledge is unique in that plans have rich causal structure that makes questions highly coupled, and frequent re-planning can change which questions are relevant.

We seek to understand whether asking questions is at all necessary, and if so, how to select the fewest questions. Agents that passively learn by trial and error can reach scenarios where it is impossible to learn about actions that impact goal achievement without asking questions. For example, an agent might apply an action with n possible preconditions that are unsatisfied in the current state, and to know why the action failed (i.e., which of the possible preconditions are actual preconditions), it would need to apply the action again in several different states (some of which may be unreachable) to isolate the problem. Instead, the agent could reactively query the domain expert to determine the problem, or prior to executing the action proactively query the domain expert. Reactive agents take a risk that either the action will not fail (i.e., the possible preconditions are not required), and proactive agents will not risk failure.

We systematically test different approaches to planning, acting, and learning with incomplete actions that:

1. Ask no questions, but learn passively.
2. Proactively ask questions and re-plan until a plan is guar-

anteed to succeed.

3. Reactively ask questions only when learning passively insufficiently learns about an action.
4. Proactively ask about highly likely future failures and 3.

We find that the first approach can lead to dead-ends where the agent fails or because of its passive learning it is incapable of formulating an effective plan. The second technique is highly successful, but asks many questions. The third, asks fewer questions and overcomes the problems of passive learning. The fourth asks more questions but reaches dead-ends less often.

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 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 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 that can be learned from, 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 $\mathsf{f} \in F$ 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 "$\phi ? f$".

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

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

We assume that the state will remain unchanged upon executing an action whose precondition is not satisfied, 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 learns 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:

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

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 (Weber and Bryce 2011) guides its expansion of plans that are labeled with failure explanations by computing relaxed plans with failure explanations. 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 $k \geq 0$,

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

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

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

Passive Learning

A passive learner would rather act under uncertainty and ask no questions of the domain expert. Passive learning agents are potentially reckless because they apply actions whose preconditions may be unsatisfied.

Using their knowledge 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 passive learning 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 passive 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 $\phi' \models d(\pi)$.

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

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

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

Proactive Learning

Proactive learning relies on planning under uncertainty and asking about action features that are relevant to the plan. The extent to which an agent is proactive is determined by how many of the relevant questions they ask before starting to execute actions. We explore three levels of proactivity: complete, asking all questions prior to execution; partial, interleaving execution (to learn passively) and question asking; and none, asking no questions prior to executing the relevant actions. In the following, we discuss how to identify relevant questions, given a plan, and how to rank the questions so that the agent can prove a plan will fail as quickly as possible.

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 the set of relevant questions is:

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

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. We define a *diagnosis-impact* measure, where we prefer questions about the incomplete action feature 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. This measure determines the incomplete feature most likely to cause the plan to fail.

Using this measure 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.

Partial Proactivity: Asking about every relevant feature will lead to a potentially large set of questions. Agents may be able to passively learn about many of the features, so asking questions only about the most impactful features can reduce the number of questions. There are a number of methods for defining the partial set of questions, such as defining a threshold on the diagnosis impact measure or selecting the features that appear in unit cardinality diagnoses (single faults). The strategy that we evaluate in the empirical evaluation is to opportunistically ask about features that appear in a unit cardinality diagnosis of $d(\pi)$.

Reactive Learning

Agents that passively learn may fail to learn about important action features. For example, if the agent executes action a_1, which has the possible precondition q (which is unsatisfied in the current state) and the possible add effect p, but the resulting state does not change, then $\phi = (fail \wedge pre(a_1, q)) \vee \neg add(a_1, p)$. At this point, the agent is not sure that it failed, and because $\phi \not\models pre(a_1, q)$ and $\phi \not\models add(a_1, p)$ the agent cannot modify its actions prior to re-planning. If the agent re-plans (deterministically), then it will generate the same plan starting with a_1 because it did not learn definitively about a_1. The agent will continue to re-plan and fail indefinitely (i.e., it reaches a learning dead-end).

Instead, the agent can realize that it may have failed and diagnose whether it failed and why. By asking about $pre(a_1, q))$ and $\neg add(a_1, p)$, the agent can learn about the action and potentially generate a different plan. For example, if it learns that $pre(a_1, q)$ holds, then it will not plan the action because q is not satisfied in the current state. If it learns that $\neg add(a_1, p)$ holds, then the action is useless and will not be planned.

The agent can rank the questions that create ambiguity (multiple diagnoses) in ϕ in the same manner as proactive questions. Reactive agents will continue to ask questions until ϕ has a single implicant δ where $\delta \models \phi$; having a single implicant means that the agent knows if it failed or not, and if it did fail why it failed. For example, after asking about $add(a_1, p)$, the agent may know $fail \wedge pre(a_1, q) \wedge add(a_1, p)$ or $\neg add(a_1, p)$. In the first, case the agent can infer $\phi \models pre(a_1, q)$ and that a_1 will not be applicable. In the second case, the agent can infer $\phi \models \neg add(a_1, p)$ and that a_1 is irrelevant.

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: Will adding reactive learning to passive learning improve agent success?
- Q2: Does proactive learning improve reactive strategies without asking too many questions?
- Q3: Does the type of planner used by the agent affect success and number of questions across the strategies?

Domains: We use four domains in the evaluation: a modified Pathways, Bridges, a modified PARC Printer, and Barter World (Weber and Bryce 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 ten ground-truth domains selected by the simulator.

The Pathways 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 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 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 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: Tables 1 to 4 list the performance of the various strategies on the instances in each domain. Within each table, the results are listed as `DeFAULT` using different heuristics "h^{FF}/$h^{\sim FF}$" – `DeFAULT` uses best first search, so its ability to find plans that reason about incompleteness is solely directed by the heuristic. The rows in the tables correspond to the previously mentioned strategies: passive learn-

Strategy	Solved	Learning Dead-End	Physical Dead-End	Timeout
Passive Only	4110 / 4314	1053 / 588	2510 / 2251	0 / 522
Passive/Reactive	4934 / 4766	0 / 0	2732 / 2385	0 / 523
Passive/Reactive/Proactive	5439 / 5004	0 / 0	2213 / 1916	0 / 755
Proactive Only	7531 / 6537	0 / 0	22 / 63	54 / 1072

Table 1: Summary of results on 7675 instances across the domains using two heuristics ($h^{FF}/h^{\sim FF}$) within the agent. Results include the number of solved problems, number of learning dead-ends reached, number of physical dead-ends reached, and timeouts.

Strategy	Plans	Re-Plan	Acts	TotalTime	?'s
Passive Only	2.72 / 2.18	1.72 / 1.18	12.91 / 13.03	0.80 / 1.78	0 / 0
Passive/Reactive/Proactive	3.33 / 2.77	1.39 / 1.01	11.58 / 12.11	0.94 / 2.32	2.54 / 2.03
Proactive Only	6.27 / 5.47	0 / 0	10.07 / 10.50	3.14 / 8.73	5.27 / 4.47

Table 2: Domains solved by all techniques (3422 instances), with an average of 81.8 actions per domain, and an average of 24 incomplete action features.

Strategy	Plans	Re-Plan	Acts	TotalTime	?'s
Passive/Reactive	8.23 / 4.14	7.23 / 3.14	16.20 / 15.02	2.06 /4.48	2.04 / 0.75
Passive/Reactive/Proactive	6.64 / 4.03	4.70/ 2.10	13.11 / 13.18	2.26 / 5.77	6.44 / 3.81
Proactive Only	14.34 / 12.44	0 / 0	9.82 / 10.32	7.86 / 28.28	13.34 / 11.42

Table 3: Barter World instances solved by all techniques (662 instances), with an average of 99.11 actions per domain, and an average of 59.59 incomplete action features.

Strategy	Plans	Re-Plan	Acts	TotalTime	?'s
Passive/Reactive	8.87 / 6.07	7.87 / 5.07	16.13 / 16.18	1.73 / 6.29	2.41 / 1.53
Passive/Reactive/Proactive	7.09 / 4.93	5.15 / 2.96	12.86 / 13.61	1.52 / 8.26	6.72 / 4.39
Proactive Only	15.70 / 14.24	0 / 0	9.52 / 10.02	6.21 / 34.99	14.70 / 13.21

Table 4: Pathways instances solved by all techniques (310 instances), with an average of 85.05 actions per domain, and an average of 56.55 incomplete action features.

ing only; passive and reactive learning; passive, reactive, and proactive learning; and proactive learning only. The columns in Table 1 are the number of instances solved (the agent achieves the goal), the number of instances where a failure to learn prevents the agent from achieving the goal (a learning dead-end), the number of instances where the agent cannot re-plan (a physical dead-end), and the number of instances where the agent runs out of time. Table 2 to 4 list the average number of planner invocations, number of planner invocations after executing at least one action, number of actions executed, total time, and number of questions. Table 2 lists results for three strategies and includes only those instances where all three strategies were able to solve the same instance; we did not include all strategies because reactive strategies do not engage if the passive strategy succeeds. Tables 3 and 4 list respective results for Barter World and Pathways because in these two domains it is possible to have learning dead-ends (where the agent cannot successfully learn passively).

To answer Q1, Table 1 and 2 indicate that adding reactive learning to passive learning or using only proactive learning do improve upon passive learning alone. All other strategies improve upon passive learning by solving more problems, encountering no learning dead-ends, re-planning less during execution, and executing fewer actions. However, these improvements come at the cost of spending more time and potentially running out of time, generating more plans, and asking more questions. These results match our intuitions about reactive learning because we are able to diagnose action failures and avoid the same action failure when we re-plan.

The tables indicate that for Q2, yes, in conjunction with passive and reactive learning proactive learning is beneficial, solving more problems, encountering fewer dead-ends, generating fewer plans, taking less actions, and using less total time than passive and reactive strategies. Proactive strategies can avoid executing actions that will lead to dead-ends by asking about the actions early. Purely proactive strategies, while typically more successful, tend to ask nearly twice as many questions as mixed proactive, passive, and reactive strategies. Limiting the number of proactive questions and attempting to learn passively (while addressing learning dead-ends with reactive learning) seems to strike a useful balance between avoiding failure and overburdening a do-

main expert with questions.

In terms of Q3, we see that the type of planner used by the agent does have an impact, verifying our prior results (Weber and Bryce 2011). We see that using a classical planning heuristic that ignores action incompleteness can often solve more problems, but the problems that it doesn't solve are mostly due to reaching dead-ends. The heuristic that reasons about incompleteness reaches fewer dead-ends, asks fewer questions, re-plans less, and tends to fail more often because of timeouts; this suggests that improving the heuristic while still reasoning about incompleteness is a promising direction for future work.

Related Work

Our investigation is an instantiation of model-lite planning (Kambhampati 2007), and is motivated by work on instructable computing (Mailler et al. 2009). This work is a natural extension of the Garland and Lesh (2002) model for evaluating plans in incomplete domains, but our method for computing plan failure explanations is slightly different in that we actually synthesize plans in incomplete domains as well as investigate learning strategies.

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 planner that uses the FF heuristic. We have shown that incorporating knowledge about domain incompleteness into the planner can lead to a more effective agent in both execution and question asking. We also differ in that we do not assume direct feedback from the environment about action failures, we can learn action preconditions, and we can query a domain expert.

Conclusion

We have presented three techniques for learning about incomplete actions that either passively learn by execution, reactively diagnose execution failures, or proactively seek to learn about features that may cause future plan failure. The learning methods are focused by plans so that learning is goal-directed, allowing us to ignore irrelevant action features. We found that i) proactively asking to learn all relevant incomplete action features leads to a large number of questions; ii) passively learning can lead to dead-ends; iii) reactively diagnosing failures while passively learning avoids dead-ends; and iv) combining reactive, passive, and proactive learning increases success without asking prohibitively more questions.

Acknowledgements: This work was supported by DARPA contract HR001-07-C-0060.

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.

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.

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.

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

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.