Learning to Improve Uncertainty Handling in a Hybrid Planning System

Jim Blythe and Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213
{jblythe,mmv}@cs.cmu.edu

Abstract

Weaver is a hybrid planning algorithm that can create plans in domains that include uncertainty, modelled either as incomplete knowledge of the initial state of the world, of the effects of plan steps or of the possible external events. The plans are guaranteed to exceed some given threshold probability of success. Weaver creates a Bayesian network representation of a plan to evaluate it, in which links corresponding to sequences of events are computed with Markov models. As well as the probability of success, evaluation produces a set of flaws in the candidate plan, which are used by the planner to improve it. We describe a learning method that generates control knowledge compiled from this probabilistic evaluation of plans. The output of the learner is search control knowledge for the planning domain that helps the planner select alternatives that have previously lead to plans with high probability of success. The learned control knowledge is incrementally refined by a combined deductive and inductive mechanism.

Introduction

We describe a machine learning technique that can be used to improve efficiency and potentially plan quality in a hybrid system for planning under uncertainty. The planning system, called Weaver (Blythe 1994; 1996), uses an AI planner (Veloso *et al.* 1995) and a probabilistic reasoner that models the probability of success of the plan. Our action model is similar to Buridan's (Kushmerick, Hanks, & Weld 1995), which is a generalisation of that of classical planning (eg (Minton *et al.* 1989)) to include uncertainty in the initial state, and in the outcomes of actions. We also model external events which can be predicted only with some probability.

Weaver iterates between a planning phase and an evaluation phase. When the planner has created a candidate plan, the probabilistic reasoning system constructs a Bayesian belief net to evaluate its probability of success. The construction uses the operator model to generate the probabilistic dependencies among the plan steps. It models the change of some aspects of the domain over the time taken to execute the plan using Markov chains, as we describe below. The output of the evaluation is also used to suggest possible ways to improve the candidate plan.

In this paper we discuss a learning technique that can be applied to hybrid planners of this form. We describe a method to learn search control rules for the planner from the experience of the probabilistic reasoner during the evaluation phase. This is a form of speed-up and quality learning, which will allow the planner to converge more quickly to plans with a high probability of success. By allowing experience gained from the probabilistic analysis of plans to be used at the stage of plan formation, the system can reduce the number of calls made to the probabilistic analysis module and greatly improve efficiency. In domains where there are limited computational resources for planning, this gain in efficiency can lead the system to produce plans of higher quality within the resource limits, and sometimes to find a plan of acceptable quality where otherwise it would not.

Since the learned knowledge is expressed in a form usable directly by the planner, the representation is no more complex than for classical planning. The basic technique can be applied to any planning system that makes use of an external critic to compile the experience of the critic into search guidance for the planner. We illustrate this learning method in a challenging planning domain that has a very large state space, real-valued as well as nominal attributes and complex interactions between goals (Desimone & Agosta 1994). This domain is of potential interest as a bench-mark for learning to act in dynamic, uncertain worlds, and we briefly discuss it in the next section.

Machine learning techniques have been studied extensively to improve the efficiency of classical AI planning systems that do not include any representation of uncertainty (Minton 1988; Veloso 1994; Borrajo & Veloso 1996; Estlin & Mooney 1996; Katukam & Kambhampati 1994) and these techniques are still useful for reducing search in domains with uncertainty. Some of the machine learning methods used in classical planners find new uses for the conditional plans now considered. Analogy, for example, can be used very efficiently for the specific problem of re-using planning effort across conditional branches of a plan (Blythe & Veloso 1996). In addition, Weaver offers several other opportunities for speed-up learning in its modules. The Bayesian net construction module, for example, re-creates certain model fragments many times in a particular domain, as specific sub-plans and event interactions recur. These

could be generalised and cached.

In contrast, the learning technique described here exploits the hybrid nature of Weaver. It produces search control knowledge for the planner by analysing the probabilistic model. The learning algorithm generates control knowledge compiled from the evaluation of the impact of the external events and the threshold probability of success. The learned control knowledge is incrementally refined by a combined deductive and inductive mechanism.

Planning in Dynamic, Uncertain Domains

In recent years there has been much research in planning under uncertainty (Kushmerick, Hanks, & Weld 1995; Blythe 1996; Collins & Pryor 1995; Boutilier, Dean, & Hanks 1995). Most of the systems developed use a probabilistic representation for the sources of uncertainty and the performance of the plan. Plans typically include sensing and conditional branches. The input to the planner is usually a probability distribution over the possible initial states, a goal to be achieved and a minimum probability with which a plan should achieve the goal. In this section we describe Weaver and the oil-spill planning domain which we use to illustrate learning in Weaver.

Modelling uncertainty

In Weaver, a planning domain P is defined by set of variables V, a set of values dom(v) for each variable $v \in V$, a set of actions A, a set of events E and a total order $<$ over the events and actions. A *state* in the domain is a complete assignment of values to the variables of V so that each variable v is assigned a value in its domain by the state s. A planning problem consists of a probability distribution over the set of states called the *initial state distribution*, a *goal description* G, which is a sentence involving the variables in V, and a *threshold probability* τ. A solution to the planning problem is a *policy* π, a partial mapping of states to actions in A, such that if the action $\pi(s)$ is taken whenever the state of the world is s, some state that satisfies G will be eventually reached with probability at least τ. The *probability of success* of a policy π with respect to a planning problem is the probability that such a state is reached. The rest of this section defines this probability by describing an underlying Markov decision process M used to define the actions and events in P.

The states of M consist of two parts:

1. an assignment to the variables Σ, corresponding to a state in the planning domain
2. a set of *pending effects* Δ.

Each element of the pending effects is a triple (t, a, σ) where t is an integer denoting time, a is an action or event and σ is a set of "effects", a partial assignment of values to variables in V. The pending effects represent the events and actions that are currently taking place in the given state of M.

We begin by defining a "successor function" $n(.)$ on states of M. Let s be a state in M and write $s = \Sigma \cup \Delta$ where Σ and Δ are as described above. To calculate the successor state $n(s)$, first find those pending effects whose time value is 1:

$$\Delta^1 = \{\delta = (1, a, \sigma) \mid \delta \in \Delta\}$$

Δ^1 can be thought of as the "imminent" pending effects, and is applied to the variables assignment Σ to give a new assignment Σ' defined by:

$$\begin{aligned} \Sigma'(v) &= \sigma(v), \text{ when } (1, a, \sigma) \in \Delta^1, \sigma(v) \text{ is defined and } a \text{ is the least action in the total order for which this is true.} \\ \Sigma'(v) &= \Sigma(v), \text{ if no such pending effect exists.} \end{aligned}$$

The remaining pending effects, $\Delta^+ = \Delta - \Delta^1$, contribute to the pending effects of the new state $n(s)$ with their time count reduced by one:

$$n(s) = \Sigma' \cup \{(t-1, a, \sigma) \mid (t, a, \sigma) \in \Delta^+\}$$

This successor function forms the basis of the MDP M describing the actions and external events in the planning domain P. As described below, changes to the world due to actions or external events are expressed by inserting pending effects into the set Δ for the state that is changed. At each time step, the successor function is performed. Note that the successor function itself is deterministic: $n(s)$ is uniquely defined for each state $s \in M$.

Actions and external events

Actions are triples $a = (\pi, d, ed(.))$, where π is a sentence over the variables V that must be satisfied in any state of P where a is applied. We refer to π as *the preconditions of* a. The *duration of* a, d, is an integer specifying the number of time units that a takes to perform. Any effects that a has on the world will not take place until d time units after the action is begun. The *effect distribution*, $ed(.)$ is a function that takes a state and returns a probability distribution of effects, that is, a finite set $ed(\Sigma) = \{(p_i, \sigma_i) \mid 1 \leq i \leq n\}$ for some n, where each σ_i is a partial assignment to variables in V, each p_i is a positive number and $\sum_{i=1}^{n} p_i = 1$.

In the special case when there are no external events, the transition function $\Phi(a, m)$ that maps an action a and a state m in M to a probability distribution of new states is defined as follows:

$$\Phi(a, m) = n(m \cup \{(d, a, \sigma_i)\}) \text{ with probability } p_i$$

Thus if the duration d of action a is 1, the effects take place in the next state after a is performed, otherwise there is a delay corresponding to d before a's effects are realised. The effect distribution is calculated from the domain-level component of the state m.

External events have a precondition, duration and effect distribution similar to actions, and also have a probability of occurrence p: $e = (\pi, d, ed(.), p)$. When the set E of external events is not empty, let $H(E, \sigma)$ denote the set of external events whose preconditions are satisfied in the state σ. Intuitively, any combination of the events in $H(E, \sigma)$ might take place when an action is performed in state σ. The probability that event h takes place and has effect $ed(\sigma)_i$ is $p \times p_i$.

For each subset $H' \subset H(E,\sigma)$, applying action a can lead to transitions with the following probabilities:

$$\begin{aligned}\Phi(a,m) &= n(m \cup \{(d,a,\sigma_i)\} \bigcup_{h \in H'} (d_h, h, ed(\sigma)_{i_h})) \\ &\quad \text{with prob } p_i \prod_{h \in H'} p_h p_{i_h} \prod_{h' \in H - H'} (1 - p_{h'})\end{aligned}$$

A simple example

The motivating example in the first section can be modelled with three variables in the set V, person, plane and taxi representing, respectively, the locations of the traveller, plane and taxi. The domain of person is $\{\mathsf{airportA}, \mathsf{airportB}, \mathsf{hotel}\}$, while the domain of taxi is $\{\mathsf{airportB}, \mathsf{hotel}\}$ and the domain of plane is $\{\mathsf{airportA}, \mathsf{airportB}\}$.

To complete the planning domain we need to specify the actions take-plane and take-taxi and the event taxi-moves. The action take-plane has the precondition $\mathsf{person} = \mathsf{plane}$, specifying that the traveller must be in the same location as the airplane. The action has a duration of 3 units, and an effect function that moves the plane and passenger on completion of the action:

$$\begin{aligned}ed(\sigma) &= \{(\{(\mathsf{plane} = \mathsf{airportB}), (\mathsf{person} = \mathsf{airportB})\}, 1)\} \\ &\quad \text{if } \sigma[\mathsf{plane}] = \mathsf{airportA} \\ ed(\sigma) &= \{(\{(\mathsf{plane} = \mathsf{airportA}), (\mathsf{person} = \mathsf{airportA})\}, 1)\} \\ &\quad \text{if } \sigma[\mathsf{plane}] = \mathsf{airportB}\end{aligned}$$

In both cases the effect distribution is a singleton set with probability 1, so the action is always deterministic. The take-taxi action is defined analogously, except that it has duration 1 and modifies the taxi variable rather than the airplane variable.

The one event, taxi-moves also has duration 1. It has a null precondition that is satisfied in every state, and an effect function that moves the taxi to its other location:

$$\begin{aligned}ed(\sigma) &= \{(\{(\mathsf{taxi} = \mathsf{airportB})\}, 1)\} \\ &\quad \text{if } \sigma[\mathsf{taxi}] = \mathsf{hotel} \\ ed(\sigma) &= \{(\{(\mathsf{taxi} = \mathsf{hotel})\}, 1)\} \\ &\quad \text{if } \sigma[\mathsf{taxi}] = \mathsf{airportB}\end{aligned}$$

In addition the event has probability $1/3$, so when an action is taken in a state, the effect that the taxi moves is added with $1/3$ probability. However, we define take-taxi $<$ taxi-moves in the total order on actions and events, so whenever a take-taxi action and a taxi-moves event complete at the same time, the value of taxi is taken from the action take-taxi. The successor function $n(.)$ is well-defined without specifying the rest of the total order, because no other pair of actions or events modify the same variable.

Evaluating plans using the underlying Markov process

Consider the initial state σ in which the traveller and the airplane are at airport A and the taxi is at airport B. Consider the plan to move the traveller to the hotel that consists of two actions taken in sequence: $\mathsf{take-plane}$ followed by $\mathsf{take-taxi}$. The probability of success of the plan is computed by examining the possible state transitions corresponding to the plan in the underlying Markov process M. Since Weaver creates plans that may be "open-loop", that is, they may contain no sensing actions, a plan does not strictly correspond to a Markov Decision Process since in some cases the plan may select different actions in the same state. A true MDP representation could be created by including the history of the plan in the state, but this is not included in M for simplicity.

In this example, there is one possible initial state m_0, in which the action $\mathsf{take-plane}$ is performed, leading to two possible successor states m_1 and m_1' as shown in Figure 1. Both of these have the Δ portion equal to $\{(2, \mathsf{take-plane}, \{\mathsf{airplane} = \mathsf{airportB}, \mathsf{person} = \mathsf{airportB}\})\}$. Since in the plan the actions take place sequentially, no action is taken for the next four time points, until successive applications of the successor function remove the $\mathsf{take-plane}$ effect from the Δ portion of the state and apply it to the Σ portion. At this point there are two possible states, m_3 and m_3' as shown in Figure 1. In only one of these, m_3, is the action $\mathsf{take-taxi}$ applicable, so the plan fails in the case that m_3' is reached, and succeeds otherwise, since m_4 satisfies the goal.

The probability of success of the plan can be computed in a number of ways, the simplest of which is to back up the values from the Markov process. Backing up the values, we successively compute the probability of success in all states that can be reached at times 4 (m_4 and m_4') and use these to compute the probability of success of all states that can be reached at time 3, and so on until the states in the initial distribution are reached. These values are shown below the states in Figure 1. In this way, the underlying Markov process M can be used to show that the probability of success of the two-step plan is $14/27 \approx 0.52$.

It is typically intractable to work directly with the underlying Markov process to evaluate plans in large domains, such as the oil-spill domain used to illustrate this paper which contains over 30 events and over 1600 literals. Rather than work with this model, our approach is to use the commitments already made in an emerging plan to restrict attention to only the sources of change in the world that are relevant to the plan. In (Blythe 1996) we show how to build reduced models of the domain that can be used to compute the correct probability of success of a given plan much more efficiently than can usually be done with the full model.

Weaver

Directly solving the underlying MDP for a given planning domain is typically intractable. Instead, the philosophy used in Weaver is to ignore many of the sources of uncertainty while forming an initial plan, which is then used to decide which sources of uncertainty are relevant for its evaluation. In order to do this, Weaver iterates between a planning phase and a plan evaluation phase. The planning system ignores external events when creating a plan and relies on the evaluator to check which events may affect it. The

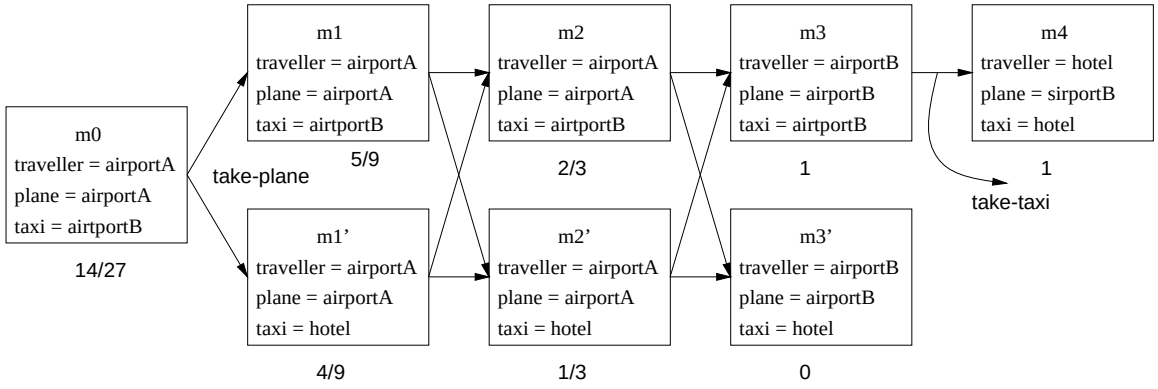

Figure 1: The states of M visited in all possible executions of the plan take-plane followed by take-taxi. Probabilities of success are backed up and shown below each state.

information that the evaluation phase provides may be used by Weaver either to improve or discard the current plan.

In either case, the information may be learned and applied as control knowledge in the planning phase to avoid creating plans which would ultimately be rejected after an evaluation phase. We describe a learning method that does this in the next section. Figure 2 shows the global architecture of the system. Weaver iterates between the classical planner and the probabilistic plan evaluator. When the planner is forced to backtrack because its plans fail to meet the threshold probability, this is a possible learning opportunity. The input to the learning module includes both the trace of the planner and of the evaluator, and the output is a set of search control rules that can be used by the planner, without reference to the evaluator, in an attempt to produce plans directly with higher probability of success.

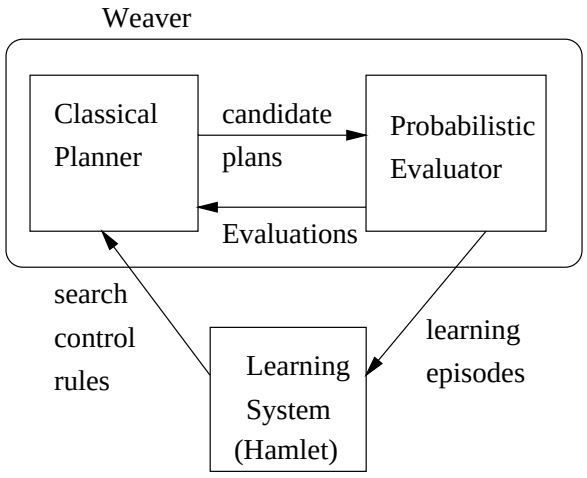

Figure 2: The global architecture for planning and learning under uncertainty.

The oil-spill domain

We illustrate the planner and the learning technique with a planning domain for dealing with oil spilled from tankers at sea. This domain was originally developed at SRI for use with the SIPE planner (Desimone & Agosta 1994), but in our version we have added information about uncertain external events that control the weather and the spread over time of spilled oil. The domain includes information about equipment that can be used to help with oil spills, including chemical dispersants, skimmers and transportation equipment, as well as information about locations in the San Francisco bay area such as sea ports, environmentally sensitive shoreline and sea sectors.

A simple initial scenario and goal from the oil-spill domain is as follows: a tanker which has begun to spill oil is located off the Santa Clara coast. Weaver is given a conjunction of 3 goals: (1) stop the flow of oil from the tanker, (2) make sure there is no spilled oil and (3) make sure there are no unprotected sensitive areas of shoreline that are threatened by the spilled oil. In the initial state, (2) and (3) are satisfied since there is no spilled oil, and the planner produces a plan to satisfy (1) in the absence of external events. In this case the chosen plan is to move a pump and a tank-barge from Richmond port, and pump the oil into the barge as it flows from the tanker. Weaver picks a pump that can transfer oil at a faster rate than it is flowing from the tanker, so the flow is effectively stopped.

The plan Weaver initially chooses to stop the flow of oil is shown in Figure 3. After the pump and barge are moved to the scene of the spill, step 3 pumps the oil. From the domain model, the action of pumping the oil requires the pump and the barge to be in the correct position, and the weather in this part of the sea to be not too rough.

The plan is passed to the probabilistic evaluation module, which first computes the duration of each step in the plan. Since the scene is 35 miles from Richmond port, and the barge's maximum speed is 2 knots, step 3 begins at least 18 hours after execution of the plan is started. Once times are assigned to the steps, a belief net is constructed for the plan that records each fact about the world that must be true

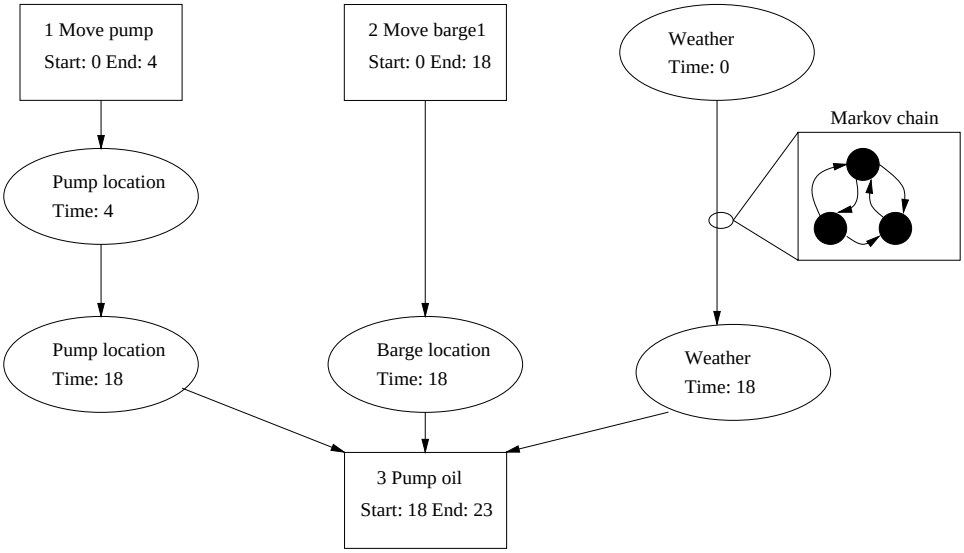

Figure 4: A portion of the belief net generated for the initial plan. Square nodes represent actions in the plan, and round nodes are state variables at given times, generated as preconditions or effects of actions, or representing persistence. The probability matrix for the arc between the two nodes for the variable "weather" is computed from a Markov model derived from the relevant external events.

1. move cargo-pump1 santa-clara-coast richmond-port utility-boat-2
2. move tank-barge1 richmond-port santa-clara-coast
3. cargo-transfer-oil-to-stabilize ss-weany santa-clara-coast tank-barge1 cargo-pump1 600 120

Figure 3: Initial plan (probability 0.51 of success)

for the world to succeed, essentially using the subgoal tree from the action models created by the planner. Since the plan is correct with respect to the actions, each such fact is established either by a previous action or in the initial state, and no actions make the fact false after it is established but before it is required. Thus several of the required conditions must hold true over a non-zero time interval, for the plan to succeed. These persistence assumptions are justified by searching for external events that can make them false, compiling the relevant events into a reduced Markov model and running the model for the time period indicated from the analysis of the plan.

In this instance, two of the persistence intervals have non-trivial Markov models: the sea state during the set-up for pumping the oil and the amount of oil spilled during the plan's execution. These models are run over the time periods required, and the evaluation module returns to the planner a low probability of success and two failure modes: that the sea conditions are too poor when the barge arrives, with probability 0.4, and that some oil has spilled from the tanker by the time the barge arrives, with probability 0.85. Part of the belief network generated to model the plan, dealing only with the preconditions of the final step, is shown in Figure 4.

The planner improves its original plan by adding a conditional branch, contingent on whether oil has spilled from the tanker. The new steps in the plan move a skimmer to the scene to skim oil from the surface of the water into the same barge. This second plan is shown in Figure 5. This plan is evaluated and found to have probability 0.6 of success, with the change in the weather being the only failure mode.

1. move cargo-pump1 santa-clara-coast richmond-port utility-boat-2
2. move tank-barge1 richmond-port santa-clara-coast
3. cargo-transfer-oil-to-stabilize ss-weany santa-clara-coast tank-barge1 cargo-pump1 600 120
IF oil-spilled santa-clara-coast
BEGIN
4. move boom-skimmer1 oakland-port santa-clara-coast
5. perform-open-water-recovery weany-spill santa-clara-coast 600
6. clean-up-spill weany-spill santa-clara-coast
END

Figure 5: The improved plan (probability 0.6 of success)

Learning Planning Control Knowledge from the Probabilistic Analysis

The example problem in the previous section also illustrates an opportunity to apply speed-up learning from the evaluation module to the planning module of Weaver. Concretely, the choice of which barge to use affects the probability of success as a function of the sea condition. In the problem scenario there are four different tank barges that the planner can choose from to pump oil into, leading to different probabilities of success for the final plan, because of their

different speeds and different worst operating conditions. The planner is unaware of these factors since it ignores the external events that cause the weather to change, and picks a barge at random. In order to find the plan that maximises the probability of success in this example problem, Weaver iteratively calls the planner again, and the planner has to backtrack over the choice of which barge to use and pick *tank-barge2* instead of *tank-barge1*, because according to the problem description, *tank-barge2* has a higher maximum sea state although it is twice as slow as *tank-barge1*.

This example illustrates the learning opportunities available in planning under uncertainty. In general, our learning method allows Weaver to learn from experience which features of the problem to pay attention to in choosing the steps in a plan. The method is based on HAMLET (Borrajo & Veloso 1996) which is a learning algorithm that combines deductive and inductive techniques to acquire control rules to improve the planning efficiency and the quality of the plans generated.[1] In this section, we briefly introduce HAMLET's learning technique and then present how we apply and

HAMLET as an inductive explanation-based learner

HAMLET is a learning algorithm that uses examples of planning episodes, i.e., search trees, to automatically generate control knowledge to guide future plan search. HAMLET was designed and developed to address situations, where: (1) example search trees are *explainable*, i.e., there is some underlying domain theory that can guide the automated explanation (interpretation) process, but (2) generating *correct* explanations from a single example is too expensive or impossible, i.e., the underlying domain theory is neither correct nor complete to the point of providing effective generalization information.[2]

Several learning algorithms applied to problem solving generate explanations for the local decisions made during the search process (e.g., (Laird, Rosenbloom, & Newell 1986; Mitchell, Keller, & Kedar-Cabelli 1986; DeJong & Mooney 1986; Minton 1988; Pérez & Etzioni 1992; Katukam & Kambhampati 1994). These *explanation-based* techniques follow a deductive approach and invest a substantial explanation effort to produce proven correct and complete control rules from a single (or few) problem solving examples and a correct and complete underlying domain theory. There has been work on learning with incomplete, or intractable theories, (e.g., (Tadepalli 1989)) applied to simple problem solving scenarios.

Alternatively to these deductive domain-theory dependent algorithms, inductive learning approaches incrementally acquire correct knowledge by observing a large set of problem solving examples. These approaches strongly depend on the particular examples seen, but can also acquire simple and useful rules (Cohen 1990; Leckie & Zukerman 1991).

HAMLET combines a deductive and an inductive approach: it generates *bounded* explanations from search trees that are *refined* inductively with more example problems. Hence, the learned knowledge becomes increasingly correct incrementally. HAMLET learns control knowledge to improve both the search *efficiency* of the planner and to improve the *quality* of the plans generated.

HAMLET is integrated with the PRODIGY planner. The inputs to HAMLET are a task domain ($\mathcal{D}$), a set of training problems ($\mathcal{P}$), a quality measure (Q), a learning mode (L), and an optimality parameter (O). $Q, L,$ and O will be explained shortly. The output is a set of control rules ($\mathcal{C}$). HAMLET has two main modules: the Bounded Explanation module, and the Refinement module. Figure 6 shows HAMLET's modules and their connection to PRODIGY.

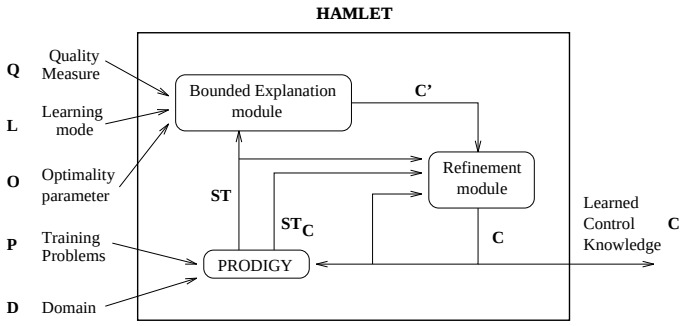

Figure 6: HAMLET's high level architecture.

The Bounded Explanation module generates control rules from a planning search tree. It explains the successful search choices by loosely following the dependencies among choices, and by selecting a bounded set of features describing the situation in which the successful choice was made. (No proof of correctness or completeness of the explanations is attempted.)

These rules might be overly specific or overly general, as their explanation procedure was *lazy*. The Refinement module generalizes rules incrementally upon analyzing new positive examples of the application of the same rules. It replaces overly general rules with more specific ones upon encountering negative examples, i.e., situations in which the learned rules lead to wrong decisions. HAMLET gradually learns and refines control rules, converging to a concise set of correct control rules (i.e., rules that are individually neither overly general, nor overly specific). In Figure 6, $\mathcal{C}$ is the current set of control rules, ST and $ST_{\mathcal{C}}$ are planning search trees generated by the planner when not using (ST) and using ($ST_{\mathcal{C}}$) the learned control rules, and $\mathcal{C}'$ is the new set of control rules learned by the Bounded Explanation module.

HAMLET has been tested in a variety of experiments involving complex planning problems. The empirical results

[1] HAMLET stands for *H*euristics *A*cquisition *M*ethod by *L*earning from s*E*arch *T*rees (Borrajo & Veloso 1994; Veloso & Borrajo 1994).

[2] In addition to the type hierarchy, set of operators and inference rules that describe the primitive problem solving action model, a complete domain theory would include also a set of domain axioms that enables the proof of the universal truth of episodic explanations.

support the effectiveness of HAMLET's learning approach, in terms of improvement in planning efficiency, in the quality of plans generated, and in its incremental convergence towards the correct knowledge (Borrajo & Veloso in press forthcoming 1996). We realize that HAMLET's learning power comes most directly from its overall lazy learning approach.

Extending HAMLET to a hybrid planner

HAMLET's incremental inductive refinement of the learned knowledge seems appropriate for compiling experience obtained from probabilistic planning episodes. We discuss how to extend HAMLET to apply to the hybrid planning scenario. Training occurs by solving simple problems, such as the earlier example from the oil-spill domain, in several different ways with different probabilities of success. Because the probabilistic model is built from events whose preconditions are known, the factors influencing the probability of success can be determined. HAMLET can use these factors to augment its partial (bounded) explanation of the preferred solution and produce control rules to be used in the planner. Three modifications are needed to use HAMLET in this way: (i) to add the plan step duration as an operational predicate in the explanation process that constructs the left hand side of the control rules; (ii) to treat the probability of success as a quality measure for plans and if necessary include it in the explanation, and (iii) to be able to handle continuous values appropriately (e.g. plan step duration is a continuous value) in both its explanation and refinement phases. Although Weaver and HAMLET are both implemented systems, these modifications are under development, so the description that follows is provided as an example of the technique but is not a trace of an already implemented algorithm.

Following up on the example of the previous section, for simplicity, consider the subgoal of stopping the discharge of oil as the top-level goal. In this situation, Weaver still produces the same plan as shown in Figure 3. This plan now has a probability of success of 0.6 as the change in weather is the only external event that matters for this subgoal. Weaver requires the planner to search for a plan with higher probability. The plan which uses tank-barge2 is found that has a probability of success of 0.84. These two solved plans are given to the learning module. HAMLET identifies the decision choices available as the learning opportunities. In particular, the point in the plan search tree where the two barge binding choices for the operator *cargo-transfer-oil-to-stabilize* is a learning opportunity. The extended HAMLET generates a partial explanation of the difference between the two plans using from their corresponding belief nets.

To generate a partial explanation for the probability of success of a plan, Weaver supplements the explanation generated by HAMLET from the plan in the normal way with literals generated from each Markov chain contributing to the plan's belief net. In each such chain, the preconditions of events that lie on a path from nodes representing desired values to nodes representing undesired values are added to the partial explanation. This explanation process is equivalent to the goal regression used by HAMLET in the sequence of planning operators. In the example the event explanation will identify the sea conditions at the scene of the oil spill in the initial state as the relevant features to be added. The duration of each step is also added to the explanation. This is computed from the bindings of the operator and so is available when the planner chooses between steps.

In this case, the set difference between the explanations of the better plan and the worse plan includes the worst sea state for each barge and the duration. HAMLET tentatively adds the control rule shown in Figure 7.

IF the chosen-operator is cargo-transfer-oil-to-stabilize
and candidate-bindings ?B1 use ?barge1
and candidate-bindings ?B2 use ?barge2
and worst-sea-state of ?barge1 is 5
and worst-sea-state of ?barge2 is 3.5
and duration of ?B1 is 37
and duration of ?B2 is 17
THEN prefer bindings ?B1 to ?B2

Figure 7: Example Control Rule Learned

Here, ?B1, ?B2, ?barge1 and ?barge2 are variables. As expected, following HAMLET's incremental learning process, the rules produced may be incorrect. In particular, the rule in Figure 7 is incorrect: it is over-general because the initial sea-state is not mentioned and also over-specific because of the explicit durations of the different instantiations of the operators. However the underlying explanations are stored so that as new cases are encountered the rule can be refined to correctly cover many examples. Durations may be generalised to ranges as more positive examples are encountered, and the initial sea state can be added to the precondition as negative examples are encountered.

Extensive empirical studies have been performed to analyze HAMLET's convergence behavior in classical domains. In the domains used, HAMLET consistently showed that the set of control rules converged to an increasingly correct set of rules (Borrajo & Veloso 1996). An equivalent empirical study as well as an analytical one in the probabilistic framework is on-going research. However this domain clearly presents a challenge to HAMLET because the learned rules are essentially trying to predict the output of one or more Markov chains in the plan's Bayesian net. Rules that simply learn conjunctions of ranges for real-valued features such as the operator duration cannot possibly learn the relation between duration, initial sea state and the best barge choice perfectly. We conjecture, however, that a few such rules will adequately cover all cases met in practice in a particular scenario such as spills in the bay area.

References

Blythe, J., and Veloso, M. 1996. Using analogy in conditional planners. Technical Report forthcoming, Computer Science Department, Carnegie Mellon University.

Blythe, J. 1994. Planning with external events. In de Mantaras, R. L., and Poole, D., eds., *Proc. Tenth Conference*

on Uncertainty in Artificial Intelligence, 94–101. Seattle, WA: Morgan Kaufmann.

Blythe, J. 1996. Decompositions of markov chains for reasoning about external change in planners. In Drabble, B., ed., *Proc. Third International Conference on Artificial Intelligence Planning Systems*. University of Edinburgh: AAAI Press.

Borrajo, D., and Veloso, M. 1994. Incremental learning of control knowledge for nonlinear problem solving. In *Proceedings of the European Conference on Machine Learning, ECML-94*, 64–82. Springer Verlag.

Borrajo, D., and Veloso, M. 1996. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Artificial Intelligence Review* in press.

Borrajo, D., and Veloso, M. in press, forthcoming 1996. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Journal of Artificial Intelligence Review*.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: structural assumptions and computational leverage. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning.* Assissi, Italy: IOS Press.

Cohen, W. W. 1990. Learning approximate control rules of high utility. In *Proceedings of the Seventh International Conference on Machine Learning*, 268–276.

Collins, G., and Pryor, L. 1995. Planning under uncertainty: Some key issues. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1567–1573. Montréal, Quebec: Morgan Kaufmann.

DeJong, G. F., and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine Learning* 1(2):145–176.

Desimone, R. V., and Agosta, J. M. 1994. Oil spill response simulation: the application of artificial intelligence planning technology. In *Simulation Multiconference*.

Estlin, T., and Mooney, R. 1996. Multi-strategy learning of search control for partial-order planning. In *Proc. Thirteenth National Conference on Artificial Intelligence*. AAAI Press.

Katukam, S., and Kambhampati, S. 1994. Learning explanation-based search control rules for partial order planning. In *Proc. Twelfth National Conference on Artificial Intelligence*. AAAI Press.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239 – 286.

Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning* 1:11–46.

Leckie, C., and Zukerman, I. 1991. Learning search control rules for planning: An inductive approach. In *Proceedings of Machine Learning Workshop*, 422–426.

Minton, S.; Carbonell, J. G.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem solving perspective. *Artificial Intelligence* 40:63–118.

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Boston, MA: Kluwer.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1:47–80.

Pérez, A. M., and Etzioni, O. 1992. Dynamic: A new role for training problems in ebl. In Sleeman, D., and Edwards, P., eds., *Machine Learning: Proceedings of the Ninth International Conference*, 367–372. San Mateo, CA: Morgan Kaufmann.

Tadepalli, P. 1989. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 694–700. San Mateo, CA: Morgan Kaufmann.

Veloso, M., and Borrajo, D. 1994. Learning strategy knowledge incrementally. In *Proceedings of the 6th International Conference on Tools with Artificial Intelligence*, 484–490.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*. Springer Verlag.