Multiple Target Concept Learning and Revision in Nonlinear Problem Solving

Daniel Borrero, Manuela Veloso
Department of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-8819

1 Introduction

We have been developing a learning and revision system, HAMLET, that learns multiple target concepts in nonlinear problem solving. HAMLET learns control knowledge for individual decisions to guide the problem solver, by loosely explaining the trace of training episodes, and then refining the learned knowledge through experiencing positive and negative examples. We have described different aspects of HAMLET's procedures in [1, 2]. HAMLET is an ongoing research project. In this talk, we describe the multiple learning opportunities that HAMLET addresses in the context of nonlinear planning, and the refinement of the learned knowledge.²

2 Learning opportunities in nonlinear problem solving

As a case study, we will present HAMLET, a learning system that uses the PRODIGY architecture [3, 4]. PRODIGY, in its latest version PRODIGY 4.0, follows a means-ends analysis backward chaining search procedure reasoning about multiple goals and multiple alternative operators relevant to the goals. The inputs to the basic problem solver's algorithm are the set of operators specifying the task knowledge, and a problem is defined in terms of an initial configuration of the world, and a set of goals to be achieved. The planning reasoning cycle involves several decision points; namely: the goal to select from the set of pending goals and subgoals; the operator to choose to achieve a particular goal; the bindings to choose in order to instantiate the chosen operator; apply an operator whose preconditions are satisfied or continue subgoaling on a still unachieved goal.

In PRODIGY's nonlinear planner, dynamic goal selection from the set of pending goals enables the planner to interleave plans, exploiting common subgoals and addressing issues of resource contention. Decisions at all these choices are made based on domain-independent search heuristics and domain-dependent user-given or learned control knowledge which, when needed, overrides the default search behavior. Therefore, it is on these decisions that HAMLET learns knowledge to lead more efficiently the problem solver to an optimal solution.

3 HAMLET's architecture

HAMLET has three main modules: the Bounded-Explanation learner, the Inducer and the Refiner. The Bounded-Explanation module learns control rules from the search tree produced by PRODIGY while solving a problem. These rules may be either over-specific or over-general, so they are refined. The Induction module generalizes the rules using more positive examples. The Refinement module specializes the rules by finding negative examples of the application of the rules, i.e., situations in which the learned rules were used incorrectly. HAMLET gradually learns and refines them converging to a concise set of correct control rules.

Figure 1 shows the sketch of HAMLET's learning algorithm, where ST and ST' are search trees, L is the set of control rules, L' is the set of new control rules learned by the Bounded Explanation module, and L'' is the set learned induced from L' and L.

¹On leave from the Universidad Politecnica de Madrid.
²Although in this short abstract, and due to the lack of space, we refer to our own previous work for completeness, in the talk we will relate our framework with other theory revision and restructuring work.
Let $L$ refer to the set of learned control rules. Let $ST$ refer to a search tree. Let $P$ be a problem to be solved. Initially $L$ is empty.

For all $P$ in training problems
- $ST = $ Result of solving $P$ without any rules.
- $ST' = $ Result of solving $P$ with current set of rules $L$.
- If $\text{positive-examples} \cong p(ST, ST')$
  - Then $L' = \text{Bounded-Explanation}(ST, ST')$
  - $L'' = \text{Induce}(L, L')$
- If $\text{negative-examples} \cong p(ST, ST')$
  - Then $L = \text{Refine}(ST, ST', L'')$

Figure 1: A high level description of HAMLET's learning algorithm.

4 Multiple target concept revision in HAMLET

HAMLET learns five kinds of control rules, directly related to the kinds of choice points of the problem solver. Each rule corresponds to a generalized target concept. The generalized target concepts are: select operator <op> to achieve the goal <goal>, select goal <goal>, select bindings for operator <op> and goal <goal>, decide subgoal when the operator <op> is applicable, and decide apply operator <op>. The number of target concepts for HAMLET for a given domain is:

$$2n + P + 2n \sum_{i=1}^{n} \text{post}(O_i)$$

where $n$ is the total number of operator schemas in the domain, $P$ is the number of predicates of the domain, and $\text{post}(O_i)$ is the number of postconditions of the operator $O_i$. HAMLET generates a set of rules for each target concept, each one with a conjunctive set of preconditions, so it learns a DNF description of the target concept, as it can learn several rules for the same target concept.

The target concepts might be over-general. The negative example allows the refinement module to modify the erroneous target concepts by specializing them. The following is HAMLET's definition of a negative example.

A negative example for HAMLET is a situation in which a control rule was applied, and the resulting decision led to either a failure, or a worse solution than the best one for that decision.

Each target concept keeps a set of the negative examples found for its rules. Each new rule generated is tested against that set. If a rule covers a negative example, then HAMLET finds the set of conditions which, when added to the preconditions of the rule, make it not cover the negative examples. If the rule was created by the induction module, it picks up in turn the conditions from the rules that generated that rule. HAMLET selects the preconditions according to their potential amount of information with respect to the specific target concept, $T_j$:

$$\text{Information-Content}(I_i, T_j) = \sum_{i=1}^{L} p(I_i, T_j) \log p(I_i, T_j)$$

where $L$ is the literal $i$, $L$ is the number of predicates in the domain, and $p(I_i, T_j)$ is the probability that a literal $L_i$ belongs to a positive example of target concept $T_j$. The data for the formula are taken from a table for each target concept that relates the number of positive and negative examples of the target concept and the literals of the domain.

If the rule was created by the Bounded Explanation module, it selects the conditions to add from the whole set of literals true in the state when it created the rule.
References


