From: AAAI Technical Report FS-94-01. Compilation copyright © 1994, AAAI (www.aaai.org). All rights reserved.

Incremental Learning of Control Knowledge for Improvement of Planning Efficiency and Plan Quality *

Daniel Borrajo
Facultad de Informática
Universidad Politécnica de Madrid
28660 Boadilla del Monte
Madrid, Spain
dborrajo@fi.upm.es

Manuela Veloso
Department of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891
veloso@cs.cmu.edu

Abstract

General-purpose planners use domain-independent search heuristics to generate solutions for problems in a variety of different domains. However, as heuristics they are, there are situations in which these heuristics do not produce the expected effective guidance, and the planner performs inefficiently or obtains solutions of poor quality. Learning from experience can help to identify the particular situations for which the domain-independent heuristics need to be overridden. In this paper, we present a system, HAMLET, that learns control knowledge and incrementally refines it, allowing the planner not only to solve efficiently complex problems, but also generate solutions of good quality. We claim that incremental learning of control knowledge and consideration of the quality of the solutions are two fundamental research directions towards the goal of applying planning techniques to real-world problems. We show empirical results in a complex domain that show the promise of our approach to support our claims.

Introduction and Related Work

Most systems that learn strategic knowledge in problem solving have been applied to problem solvers with the linearity assumption, such as the ones applied to Prolog or logic programming (Quinlan 1990; Zelle & Mooney 1993), special-purpose (Langley 1983; Mitchell, Utgoff, & Banerji 1983), or other general-purpose linear problem solvers (Etzioni 1993; Leckie & Zukerman 1991; Minton 1988; Pérez & Etzioni 1992). These problem solvers are known to be incomplete and unable of finding optimal solutions (Rich 1983; Veloso 1989).

If we remove the linearity assumption, we are dealing with nonlinear problem solvers. This kind of problem solvers are needed to address real world complex problems. Some nonlinear planners search the plan space by using partially-ordered plans (Chapman 1987; McAllester & Rosenblitt 1991). Others remove the linearity assumption by fully interleaving goals, searching in the state space, and using totally-ordered plans (Veloso 1989; Warren 1974). Most of the approaches do backward chaining, although some use forward chaining (Bhatnagar 1992; Laird, Rosenbloom, & Newell 1986). In general, there have been only a few learning approaches applied to nonlinear problem solving (Bhatnagar 1992; Kambhampati & Kedar 1991; Laird, Rosenbloom, & Newell 1986; Pérez & Carbonell 1994; Ruby & Kibler 1992; Veloso 1992).

In this paper we show that nonlinear problem solving offers new learning opportunities where domain-dependent control knowledge may be used to further improve not only the problem solver performance but also the quality of the solutions produced. Both issues are needed for scaling up the kinds of domains and problems that planners can solve. Constructing correct explanations of the nonlinear problem solver successes and failures from a single example may be computationally very expensive, as the generalization phase to generate provably correct knowledge would have to consider a large number of possible combinations of planning situations. To alleviate this effort, we developed a new approach, and implemented it in HAMLET,[1] where control knowledge for individual decisions is incrementally acquired through experience. HAMLET is integrated with PRODIGY4.0, the current nonlinear problem solver of the PRODIGY architecture for planning and learning (Carbonell *et al.* 1992). HAMLET learns local control rules by first lazily explaining the decisions made during problem solving from individual examples. Upon finding new positive and negative examples of the use of its learned rules, HAMLET incrementally induces and refines its control knowledge (Borrajo & Veloso 1993; 1994a). A similar lazy approach can be found in (Tadepalli 1989), where LEBL (Lazy Explanation Based Learning) is presented. The main difference is that while Tadepalli refines the knowledge introducing exceptions, HAMLET mod-

*This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory or the U.S. Government. This work was initiated while Borrajo was at Carnegie Mellon University on leave from the Universidad Politécnica de Madrid supported by grants from the Ministerio de Educación y Ciencia and from Comunidad de Madrid.

[1] "HAMLET" stands for *H*euristics *A*cquisition *M*ethod by *L*earning from s*E*arch *T*rees.

ifies the control rules themselves adding or removing their applicability conditions. Also, Tadepalli applies it to game playing, while we use it for general task planning.

While improving problem solving performance has been largely studied, learning to improve solution quality has only been recently pursued by some researchers, including (Pérez & Carbonell 1994; Ruby & Kibler 1992). We differ from and Pérez's work in the fact that HAMLET performs inductive refinement of the control rules, and in the way positive examples are generated. Ruby and Kibler's approach differs in the knowledge representation of the learned control knowledge, since it is a case-based learner. HAMLET combines the two kinds of optimization, by learning control rules, that allow not only to do more effective search, but also to achieve better solutions.

In the paper, we discuss what are learning opportunities for problem solving, and how we extended previous EBL work. We present HAMLET describing the main features of its deductive, inductive, and refinement modules. Finally, the paper shows empirical results on this work and draws conclusions.

Learning Opportunities

In order to efficiently solve problems in real world applications, general-purpose problem solvers must use efficient domain-independent heuristics to improve its search performance. In this section we show that additional domain-dependent control knowledge may be used to further improve not only the problem solver performance but also the quality of the solutions produced. To illustrate our points, we use PRODIGY4.0, but the reasons we highlight that make learning needed can, in general, improve any problem solver's performance.

The current nonlinear problem solver in PRODIGY, PRODIGY4.0, follows a means-ends analysis backward chaining search procedure reasoning about multiple goals and multiple alternative operators relevant to the goals. PRODIGY4.0 is a successor of the previous linear PRODIGY2.0 (Minton *et al.* 1989) and the first nonlinear and complete NOLIMIT (Veloso 1989). The inputs to the basic problem solver algorithm are the set of operators specifying the domain knowledge, and a problem specified in terms of an initial configuration of the world, and a set of goals to be achieved. Table 1 shows the skeleton of PRODIGY4.0's planning algorithm.

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. Default decisions at all these choices can be directed by explicit control knowledge.

Although PRODIGY can use a variety of powerful domain-independent heuristics (Stone, Veloso, & Blythe 1994), it is very difficult and costly to determine in general which of these heuristics are going to succeed or fail. Therefore, learning can be used for automatically acquiring control knowledge to *override the default behavior* of a particular domain-independent search heuristic to drive the planner more efficiently to a solution. Note that this need to learn when particular domain-independent search strategies do not produce desirable results is common to any planner (Veloso & Blythe 1994).

1. Terminate if the goal statement is satisfied in the current state.
2. Compute the set of *pending goals* $\mathcal{G}$, and the set of *applicable operators* $\mathcal{A}$. A goal is pending if it is a precondition, not satisfied in the current state, of an operator selected to be in the plan to achieve a particular goal. An operator is applicable when all its preconditions are satisfied in the state.
3. Choose a goal G from $\mathcal{G}$ or select an operator A from $\mathcal{A}$.
4. If G has been chosen, then
   - *Expand goal* G, i.e., get the set $\mathcal{O}$ of *relevant instantiated operators* that could achieve the goal G,
   - Choose an operator O from $\mathcal{O}$,
   - Go to step 1.
5. If an operator A has been selected as directly applicable, then
   - *Apply* A,
   - Go to step 1.

Table 1: A skeleton of PRODIGY4.0's planning algorithm and choice points.

Another reason for the need of learning relates to the issue of *optimality of the solutions* obtained by the problem solver. Our current measure of *optimality* is the length of the solution.[2] There are many domains in which the necessary knowledge to select the optimal solution is not explicit in the definition of the domain knowledge or it is costly to do a breadth-first search for finding it. In those cases, learned control knowledge can direct the search to those optimal solutions, as also pointed out by (Pérez & Carbonell 1994).

Extending Previous Work

HAMLET extends the EBL methods used with the linear planning algorithm of PRODIGY2.0 (Etzioni 1993; Minton 1988; Pérez & Etzioni 1992) to apply to the nonlinear PRODIGY4.0. This extension is needed along several aspects in order to address new problems raised by the decisions on multiple goal interleaving choices combined with multiple operator and binding choices. We identify new learning opportunities related to PRODIGY's ability of postponing planning commitments for efficiency and quality purposes. We introduce new language primitives for describing the learned control rules, in order to capture the information related to these new choices.

HAMLET reduces the explanation effort by generating partial ("bounded") explanations of branching decisions made during the search for a solution. HAMLET's inductive learning module assures the incremental correctness of every

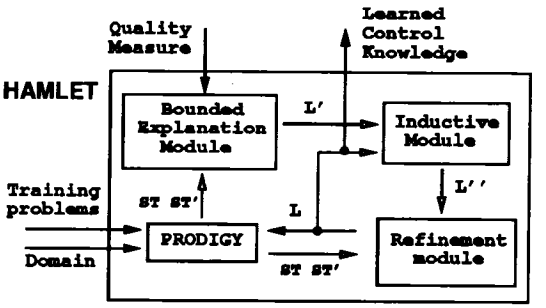
[2] Our method is not dependent of this particular metric, and can use any operational optimality measure.

deduced control rule. The use of inductive learning eliminates the need for an axiomatic domain theory to support the correct generalization of an episodic explanation, as required in EBL applied to PRODIGY2.0.

HAMLET's Architecture

The inputs to HAMLET are a domain specified as a set of planning operators, a set of training problems, and a quality measure. The output is a set of control rules. HAMLET has three main modules: Bounded-Explanation, Induction, and Refinement. The Bounded-Explanation module generates control rules from a PRODIGY search tree. These rules might be over-specific or over-general. The Induction module addresses the problem of over-specificity by generalizing rules when analyzing positive examples. The Refinement module replaces over-general rules with more specific ones when it finds 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 over-general, nor over-specific.[3]

Figure 1(a) shows HAMLET's modules and their connection to PRODIGY, and Figure 1(b) presents an outline of HAMLET's algorithm. Here ST and ST′ are search trees generated by the PRODIGY planning algorithm, 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 of rules induced from L and L′ by the Inductive module. We describe next the main features of the three modules of HAMLET.



(a)

```
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.
Let Q be a quality measure.
Initially L is empty.
For all P in the set of training problems
    ST = Result of solving P without any rules.
    ST' = Result of solving P with current set of rules L.
    If positive-examples-p(ST, ST',Q)
    Then L' = Bounded-Explanation(ST, ST',Q)
         L''= Induce(L,L')
    If negative-examples-p(ST, ST',Q)
    Then L=Refine(ST, ST',L'')
Return L
```

(b)

Figure 1: (a) HAMLET's high level architecture; (b) A high-level description of HAMLET's learning algorithm.

Bounded Explanation

The Bounded-Explanation module learns control rules by identifying important decisions made during the search for a solution and extracting the information that justifies these decisions from the search space. This explanation process consists of four phases as follows.

Labeling the decision tree HAMLET traverses the search tree bottom-up, starting from the leaf nodes. It assigns three kinds of labels to the leaf nodes of the tree: *success*, if the node corresponds to a correct solution plan; *failure*, if the node is a dead end in the search space; and *unknown*, if the planner did not expand the node. After labeling the leaf nodes, HAMLET propagates the labels to all other search tree nodes, using the algorithm described in (Borrajo & Veloso 1994a), which is similar to the one used by other systems, such as LEX (Mitchell, Utgoff, & Banerji 1983). The algorithm labels a node as: *success* if any of its children is a *success* and it does not have any *unknown* child; *failure* if all its children are labeled as *failure*; and *unknown* if any of its children is labeled as it unknown.

Credit Assignment The credit assignment is the process of selecting important branching decisions that are responsible for the successes and failures of a particular search branch. At these decision points learning occurs. Credit assignment is done while labeling the search tree. HAMLET decides that a branching decision is important, and therefore a learning opportunity, if this decision (1) leads to one of the optimal solutions and (2) differs from the default decision made by domain-independent heuristics.

Generation of control rules At each decision choice to be learned, HAMLET has access to information on the current state of the world and on the meta-level planning information, such as the goals that have not been achieved, the goal the planner is working on, and the possible applicable operators. This information is used by the generation module to create the applicability conditions, i.e. the *if*-parts, of the control rules. The set of features used in the *if*-parts of control rules can be found in (Borrajo & Veloso 1994a). Examples of features are: *true-in-state* that checks whether a predicate is true in the current search state; *candidate-goal* that checks whether a given goal is the one the planner is currently trying to achieve; or *some-candidate-goals* that checks the set of alternative goals that have not been yet achieved. The relevant features of the current state are selected using *goal regression* similarly to foot-printing in (Veloso 1992).

HAMLET learns four kinds of *select* control rules, corresponding to PRODIGY's decisions, as presented in Table 1, which are generalized target concepts. HAMLET generates a set of rules for each target concept, where the *if*-part of

[3] We have empirical evidence for this convergence phenomena, but one of our current research efforts is the formal study of the convergence of HAMLET's learning algorithm.

Test sets		Unsolved problems		Solved by both configurations (271 problems)					
				Better solutions		Solution length		Nodes explored	
Number of Goals	Number of Problems	without rules	with rules	without rules	with rules	without rules	with rules	without rules	with rules
1	100	5	0	0	11	327	307	2097	1569
2	100	15	6	0	25	528	479	3401	2308
5	100	44	23	0	29	789	708	4822	3472
10	100	68	35	1	22	731	640	3309	2846
20	75	62	45	0	8	348	322	1516	1360
50	25	24	18	0	0	34	34	143	141
Totals	500	218	125	1	95	2757	2490	15288	11696

Table 2: Empirical results on increasingly complex problems in the logistics domain.

each rule is described as a conjunctive set of predicates. As HAMLET can learn several rules for the same target concept, the set of all rules can be viewed as the disjunction of conjunctive rules.

Parameterization After a rule is generated, HAMLET replaces specific constants inherited from the episodic planning situation with variables of corresponding types. Distinct constants are replaced with differently named variables, and when the rule is applied, different variables must always be matched with distinct constants. This latter heuristic may be relaxed in the process of inductive generalization of the learned rules.

Inductive Generalization and Refinement

The rules generated by the bounded explanation method may be over-specific as also noticed by (Etzioni & Minton 1992). To address this problem, we use the Inductive Learning module, which generalizes the learned rules by analyzing new examples of situations where the rules are applicable. We have devised methods for generalizing over the following aspects of the learned knowledge: state; subgoaling structure; interacting goals; and type hierarchy (Borrajo & Veloso 1994a).

HAMLET may also generate over-general rules, either by inducing or by doing *goal regression* when generating the rules. Therefore, the over-general rules need to be refined. There are two main issues to be addressed: how to detect a negative example, and how to refine the learned knowledge according to it. A negative example for HAMLET is a situation in which a control rule was applied, and the resulting decision led to either a failure (instead of the expected success), or a worse solution than the best one for that decision.

When a negative example is found, HAMLET tries to recover from the over-generalization by refining the wrong rule. The goal is to find, for each over-general rule, a larger set of predicates that covers the positive examples, but not the negative examples. For each generalization operator, HAMLET has the inverse specialization operator. For instance, if the generalization procedure intersected the *if*-parts of two rules, the *Recover-intersection* specialization operator backtracks on each preceding rule and adds conditions from the *if*-part of the preceding rule to the *if*-part of the more general rule, until they do not cover the negative examples of the target concept(Borrajo & Veloso 1994b).

Empirical Results

We have carried out experiments on several domains, and the results we report in this section were done on a transportation domain used first in (Veloso 1992).[4] In this domain, packages must be delivered to different locations in several different cities. Packages are carried within the same city in trucks and across cities in airplanes. At each city, there are several locations, such as post offices and airports. This domain poses many interesting problems of interleaving of goals, and quality of the solutions, and it is a close approximation to a real world logistics-transportation domain with multiple routing alternatives.

We trained HAMLET with 400 randomly generated problems of one and two goals, and up to three cities and five packages. HAMLET generated 26 control rules. Then, we randomly generated 500 testing problems of increasing complexity. Table 2 shows the results of those tests. We varied the number of goals in the problems from 1 up to 50, and the maximum number of packages from 5 up to 50. Notice that the problems with 20 and 50 goals are really very complex especially in terms of finding good or optimal solutions, involving solutions of over 100 steps.

In the experiments we compare two configurations, namely PRODIGY not using and using the rules learned by HAMLET. In both situations, PRODIGY was given a CPU running time limit that varied with the number of goals of the problem according to the formula $\text{Time\_bound} = 150 * (1 + \text{mod}(\text{number\_of\_goals}, 10))$ in seconds. The results show a very significant decrease in the number of unsolved problems when PRODIGY used the learned rules. To compare the solution quality, we can only account for the problems solved by both configurations. Table 2 shows that, for a large number of those problems, the solution generated using the rules is of better quality than the one generated without the rules. This shows that the learned rules drive the planner to find solutions of quality in addition to doing it efficiently, which supports our research goals underlying HAMLET's learning algorithm. The overall running times also decreased using the rules, but not significantly. We did not find empirically with our learned rules that the time spent solving the problem degraded so much to consider it a utility problem (Minton 1988). However, we are currently

[4] We are currently working on applying HAMLET to other real world tasks.

developing efficient methods for organizing and matching the learned control rules. We consider this organization essential and part of the overall learning process (Doorenbos & Veloso 1993).

Conclusions

We have presented HAMLET, a system that learns the situations in which there is a need to override the default search behavior of a planner, to improve its performance, and also the quality of the plans generated. HAMLET first bounds the explanation of nonlinear problem solving decisions generating rules that might be over-specific or over-general, as it does not use an axiomatic domain theory to guarantee the correctness of the learned knowledge. The rules converge to correctness by inductive generalization and refinement through experience. Empirical results show that HAMLET enables the planner to solve very complex problems. HAMLET produces solutions of high quality to complex problems with significant efficiency.

References

Bhatnagar, N. 1992. Learning by incomplete explanations of failures in recursive domains. In *Proceedings of the Machine Learning Conference 1992*, 30–36.

Borrajo, D., and Veloso, M. 1993. Bounded explanation and inductive refinement for acquiring control knowledge. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, 21–27.

Borrajo, D., and Veloso, M. 1994a. 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. 1994b. Multiple target concept learning and revision in nonlinear problem solving. In *Working notes of the ECML/MLNet Workshop on Theory Revision and Restructuring*.

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Pérez, A.; Reilly, S.; Veloso, M.; and Wang, X. 1992. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, SCS, Carnegie Mellon University.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32:333–378.

Doorenbos, R. B., and Veloso, M. M. 1993. Knowledge organization and the utility problem. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, 28–34.

Etzioni, O., and Minton, S. 1992. Why EBL produces overly-specific knowledge: A critique of the prodigy approaches. In *Proceedings of the Ninth International Conference on Machine Learning*, 137–143.

Etzioni, O. 1993. Acquiring search-control knowledge via static analysis. *Artificial Intelligence* 62(2):255–301.

Kambhampati, S., and Kedar, S. 1991. Explanation based generalization of partially ordered plans. In *Proceedings of AAAI-91*, 679–685.

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.

Langley, P. 1983. Learning effective search heuristics. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 419–421.

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

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI-91*, 634–639.

Minton, S.; Knoblock, C. A.; Kuokka, D. R.; Gil, Y.; Joseph, R. L.; and Carbonell, J. G. 1989. PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University.

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

Mitchell, T. M.; Utgoff, P. E.; and Banerji, R. B. 1983. Learning by experimentation: Acquiring and refining problem-solving heuristics. In *Machine Learning, An Artificial Intelligence Approach*. Palo Alto, CA: Tioga Press. 163–190.

Pérez, M. A., and Carbonell, J. G. 1994. Control knowledge to improve plan quality. In *Proceedings of the Second International Conference on AI Planning Systems*.

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

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

Rich, E. 1983. *Artificial Intelligence*. McGraw-Hill, Inc.

Ruby, D., and Kibler, D. 1992. Learning episodes for optimization. In *Proceedings of the Machine Learning Conference 1992*, 379–384. San Mateo, CA: Morgan Kaufmann.

Stone, P.; Veloso, M.; and Blythe, J. 1994. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, 164–169.

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 Blythe, J. 1994. Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the Second International Conference on AI Planning Systems*, 170–175.

Veloso, M. M. 1989. Nonlinear problem solving using intelligent casual-commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University.

Veloso, M. M. 1992. *Learning by Analogical Reasoning in General Problem Solving*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. Available as technical report CMU-CS-92-174. A revised version of this manuscript will be published by Springer Verlag, 1994.

Warren, D. 1974. WARPLAN: A system for generating plans. Technical report, Department of Computational Logic, University of Edinburgh. Memo No. 76.

Zelle, J., and Mooney, R. 1993. Combining FOIL and EBG to speed-up logic programs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*.