# On providing prior knowledge for learning relational search heuristics

Ricardo Aler, Daniel Borrajo and Susana Fernández

Universidad Carlos III de Madrid
Avda. de la Universidad, 30
28912 Leganés. Madrid (Spain)
email: aler@inf.uc3m.es, dborrajo@ia.uc3m.es, sfarregu@inf.uc3m.es **

**Abstract.** In this paper, we propose the use of two relational learning systems, HAMLET and EVOCK, for acquiring useful search control heuristics in the context of automated task planning. In particular, we discuss the influence of different ways of providing prior background knowledge to such systems. We compare the results of providing initial information by means of a human-centered approach against two automated approaches. The first automated one consists of using the output of HAMLET as input to the learning process of EVOCK, and viceversa. The second automated approach consists of using another planner for providing guidance towards solutions of problems.

## 1 Introduction

Planning in non-trivial applications is an exponential process, that has to be guided by human or machine generated knowledge. In the context of machine learning, several approaches have been used to supply that guidance, but varying from Case-Based Reasoning, as in [13, 22], to pure EBL-macrooperators as in [10, 14, 16, 18]. More recent approaches combine techniques, as reinforcement learning with ILP [7]. Other approaches combine deductive and inductive methods that rely one way or another on relational learning techniques, as in the use of FOIL [19] integrated with different planners [9, 12].

One of the first approaches that combined deductive learning techniques, as EBL, with inductive learning techniques based on relational learning, was HAMLET [6]. We presented in [5] a relational learning description of the learning technique, as well as a comparison with using FOIL and PROGOL [17]. Later, we developed another learning technique that could be considered as relational learning, and used a genetic programming approach, EVOCK [3]. We showed that the behaviour of this new learning technique depends very much on the prior (background) knowledge that we provided to it, in the form of an initial population, or an auxiliary population that could be used for the crossover operators [2]. Both approaches generated control knowledge in terms of control rules

** This work has been partially supported by the Spanish MCyT under project TIC2001-4936-E.

to be used by the PRODIGY planner [23]. We have also explored in the past the usefulness of a mixed initiative approach (collaborative work of a human and an automated system) to control knowledge (heuristics) generation in the context of planning [1].

In the context of non-linear planning, many relational based learning systems have opted to provide prior knowledge for learning one way or another. For instance, Q-RRL [7] uses predicates such as `numberofblockson` to learn control knowledge for the blocksworld domain. SCOPE, that modified FOIL to learn control rules for UCPOP, also required human supplied prior knowledge [8]. The reason for this is that it is very difficult for a machine learning system to find the right conditions that have to appear in the left hand side of control rules. These conditions (called meta-predicates in our case) vary from checks on the literals that are true in the current state of the planning process, to checks on the things that are true in the meta-state of the search process: on which goal the planner is working, which operator it has selected to set as true a given literal, or what is the initial goal that introduced a given sub-goal in the search tree.

There has already been some work on the effect of prior knowledge in some ILP learning tasks, as in [20, 21], but it has been for more classical ILP tasks. In this paper, we wanted to study the effect of providing prior knowledge for learning in planning. We have selected for this study the two relational learning systems, HAMLET and EVOCK, and have supplied prior knowledge by means of three different strategies: a human provides the prior knowledge in terms of a set of control rules; one relational learning system provides the set of control rules as prior knowledge to the other; and another planner provides knowledge on how to solve a given planning problem by generating one solution to the problem. The experiments compare the three different approaches in two relatively difficult domains for machine learning in planning: logistics and depots.

Section 2 overviews the planner, and the machine learning techniques that we have used for the experiments (they have been extensively covered in previous works). Section 3 describes how to incorporate prior knowledge into each machine learning technique. Section 4 presents the experiments that we have performed and their outcome. And, finally, Section 5 draws some conclusions from the work.

## 2 Planner and machine learning techniques used

In this paper, we intend to study the effects of using prior knowledge on planning control knowledge learners. First, the planner itself (PRODIGY) will be introduced and then each one of the two machine learning techniques: HAMLET and EVOCK.

### 2.1 PRODIGY

PRODIGY is a nonlinear planning system that follows a means-ends analysis. The inputs to the problem solver algorithm are:

– Domain theory, $\mathcal{D}$ (or, for short, domain), that includes the set of operators specifying the task knowledge and the object types hierarchy;

– Problem, specified in terms of an initial configuration of the world (initial state, $\mathcal{S}$) and a set of goals to be achieved ($\mathcal{G}$); and
– Control knowledge, $\mathcal{C}$, described as a set of control rules, that guides the decision-making process.

A planning operator is the specification of an action that informs how the world changes when the operator is applied. PRODIGY uses a specific domain description language whose representation capabilities are better, in some issues, than the current standard PDDL2.1, but very similar to the base ideas.

PRODIGY planning/reasoning cycle, involves several decision points. The types of decisions made are:

– *select a goal* from the set of pending goals and subgoals;
– *choose an operator* to achieve a particular goal;
– *choose the bindings* to instantiate the chosen operator;
– *apply* an instantiated operator whose preconditions are satisfied or continue *subgoaling* on another unsolved goal.

We refer the reader to [23] for more details about PRODIGY. In this paper it is enough to see the planner as a program with several decision points that can be guided by control knowledge. If no control knowledge is given, backtracking might be required, thus reducing planning efficiency.

### 2.2 HAMLET

HAMLET is an incremental learning method based on EBL (Explanation Based Learning) and inductive refinement of relational formulae (control rules) [6, 5]. The inputs to HAMLET are a task domain ($\mathcal{D}$), a set of training problems ($\mathcal{P}$), a quality measure ($Q$)[1] and other learning-related parameters. The output is a set of control rules ($\mathcal{C}$). HAMLET has two main modules: the Bounded Explanation module, and the Refinement module.

The Bounded Explanation module generates control rules from a PRODIGY search tree by finding examples of right decisions (lead to a good solution instead of a failure path). Once a right decision is found, the rule is generated by extracting the meta-state, and performing a goal regression for finding which literals from the state were needed to be true to make this decision (the details can be found in [6]). The EBL rules might be overly specific or overly general. HAMLET Refinement module solves the problem of being overly specific by generalising rules when analysing new positive examples. It also replaces overly 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, in an attempt to converge to a concise set of correct control rules (i.e., rules that are individually neither overly general, nor overly specific).

HAMLETbe considered as a relational learner as it learns relational control rules.

[1] A quality metric measures the quality of a plan in terms of number of operators in the plan, execution time (makespan), economic cost of the planning operators in the plan or any other user defined criteria.

### 2.3 EVOCK

We only intend to provide a summary of EVOCK and refer to [3] for details. EVOCK is a machine learning system for learning control rules based on Genetic Programming (GP) [15]. GP is an evolutionary computation method that has been used for program induction. Instead of complete programs, EVOCK tries to induce control knowledge. EVOCK starts from a population of sets of control rules (called individuals) that can be either randomly generated, or initialised with some prior knowledge.

Then, it follows a kind of heuristic beam search to find a good enough set of control rules. During the search process, individuals are modified by the so called genetic operators. Only syntactically correct control rules are always generated by means of a grammar. EVOCK genetic operators can grow (components of) rules, remove (components of) rules and cross parts of rules with parts of other rules, just like the GP crossover operator does. EVOCK also includes some tailor made operators for modifying control rules. EVOCK search is guided by the fitness function, which measures individuals according to the number of planning problems from the learning set they are able to solve, the number of nodes expanded, and the size of the individual (smaller individuals are preferred). EVOCK can be considered also as a relational learner as it learns relational control rules.

## 3 Introduction of prior knowledge

The goal of this paper is an empirical study of the effect of providing prior knowledge to a relational learning system in the planning framework. We will describe in this section the different options that we have used to supply such knowledge for the HAMLET and EVOCK systems.

### 3.1 Providing prior knowledge to HAMLET

We have devised two different ways of providing prior knowledge to HAMLET for this work. The first one consists of providing HAMLET an initial set of control rules. This set can be generated by a human, or by a previous learning process of another learning technique. In this case, HAMLET can use this initial set of control rules to generalise some of them, if it thinks it is needed, or remove some of them if negative examples of their use are found. They cannot be specialised given that they do not come from a bounded explanation, but were given directly as they are by an external source. Therefore, it would not know what meta-predicates (conditions) to add to the control rule.

The second method for incorporating knowledge into HAMLET consists of receiving a solution to a planning problem by another planner. The reason that lead us to use this approach was that, in some domains, it might be the case that the planner we were using, could not solve some planning problems. HAMLET assumes that the planner is able to solve planning problems, in order to provide positive and negative examples of decisions that were made in the search tree.

If the planner cannot generate even one solution, the learning system cannot generate those instances. Therefore, we used another planner, FF [11] in the case of these experiments, when our planner could not solve a problem in a reasonable learning time. The approach is general and any other planner could have been used. The generation of instances to learn from is performed in two steps:

- In the first step, FF is given the same planning problem, and it generates a solution to the problem. If it finds a solution, this solution does not have to be the best one in terms of plan length or solution quality, but when the domain is a difficult one, generating one solution might be enough. Then, the problem consists of how the solution to a problem can help PRODIGY to generate instances to learn from, since HAMLET needs a search tree in which there is, at least, one solution path, and, possibly, several dead ends.
- So, the second step consists of artificially generating a search tree from the FF solution. This is not a trivial task, given that a solution to a planning problem does not incorporate all the needed rationale to reproduce a problem solving episode. It provides, basically, the order on which a set of instantiated operators have to be executed. But, there are potentially many search trees that can generate this order. We have opted to use a set of control rules (independent of the ones that are being learned, and of the problem within a domain), that select as valid search nodes, the ones that use any of the instantiated operators in the solution. There are two types of control rules in this set:
  + Select operator: one control rule selects those operator names that are member of the FF solution. If the solution incorporates all operators in the domain, then this control rule does not help in following the solution provided by FF given that, for each goal, it would select all relevant operators (those that can achieve that goal).
  + Select bindings of an operator: there is one control rule for each operator in the domain. The bindings control rule for an operator $O$ would select all bindings for its variables that correspond to instantiations of $O$ in the solution. As an example, if a solution in the logistics domain is:

    load-airplane(package1,airplane1,airport1)
    fly-airplane(airplane1,city1,city2)
    unload-airplane(package1,airplane1,airport2)
    fly-airplane(airplane1,city2,city3)

    the bindings control rule for the `fly-airplane` operator would select in all bindings selection nodes of the search tree in which the `fly-airplane` operator has been previously selected the following bindings (substitution):

    ((<airplane . airplane1) (<city-from> . city1) (<city-to> . city2))
    ((<airplane . airplane1) (<city-from> . city2) (<city-to> . city3))

  Using this new set of rules, it generates a solution path corresponding to the solution provided by the other planner. Afterward we let PRODIGY continue searching for different solutions, so that HAMLET can generate control rules

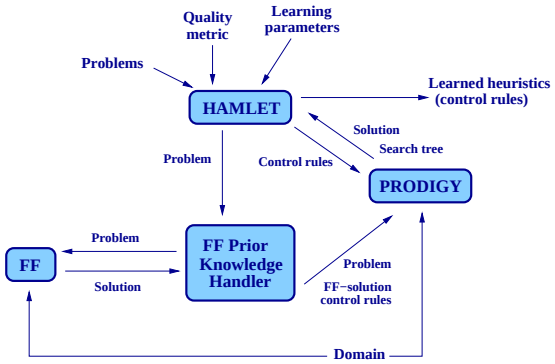


**Fig. 1.** Providing prior knowledge to HAMLET by using another planner, FF.

from the decisions that led to better solutions. Figure 1 shows a schema of the approach.
In some cases, even with this guided process the planner is not able to generate a solution. This is so, because in those cases the set of possible search paths that can be generated out of this scheme is very large, and the solution path might not be found. We will explore better ways of generating a solution path out of a solution provided by another planner (or even a human) in the future.

A third way of providing knowledge into HAMLET that we have not explored in this paper, and it might be interesting to study in the future, consists of the definition of a set of domain-dependent functions that could be used by HAMLET for generating the control rules conditions. This would be equivalent to standard definition of background knowledge in other ILP systems.

### 3.2 Providing prior knowledge to EVOCK

Instead of starting from a random population, EVOCK can accept prior knowledge from different sources. In particular, it is possible to seed EVOCK initial population with control rules generated by either a human or other machine learning techniques. These rules provide a starting point, that might be difficult to get at by purely evolutionary means. These seeding rules also focus the search in the set of control rules space. EVOCK can also use EBL rules (or HAMLET Bounded Explanation module) by means of the Instance Based Crossover [2], although we will not experiment with it in this paper.

Figure 2 shows how HAMLET rules are used as prior knowledge by EVOCK. First, HAMLET is run to learn from a set of randomly generated training problems. HAMLET uses the search trees returned by PRODIGY after solving each of the training problems. Then, HAMLET control rules are used to seed EVOCK initial population, along with other randomly generated individuals. Control rules are evaluated by EVOCK by loading them into PRODIGY. Then, the PRODIGY+control

rules system is run and performance data such as whether the learning problems were solved or not, or the time required to solve them, is returned to EVOCK, so that the fitness of individuals can be determined.

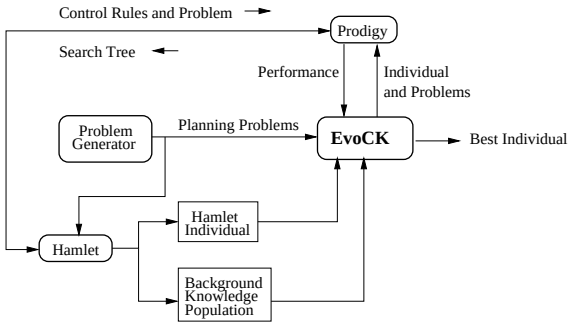


**Fig. 2.** Prior knowledge for EVOCK.

## 4 Experiments and results

For the experiments, we have used two commonly used domains in previous planning competitions: logistics [4] and depots (from the 2002 competition). The depots domain is a specially hard one.

In both domains, we trained separately both HAMLET and EVOCK with 400 randomly generated training problems of 1 and 2 goals in the logistics, and 200 randomly generated training problems also of 1 and 2 goals in the depots domain. Then, we provided prior knowledge in the different forms that we explained in previous section to each learning system, and tested against randomly generated test problems. In the logistics, we used 120 problems ranging from 1 to 5 goals. In the depots, we used 60 problems also ranging from 1 to 5 goals.

Table 1 displays the results for all the systems working autonomously. Results for human (an expert on planning and control knowledge definition) generated control rules are also shown. The main conclusion is that both domains are quite hard for PRODIGY and for the learners. The human gets mixed results: 100% problems solved in the logistics domain, but only 55% in the depots domain. It is also noticeable that HAMLET cannot solve any problem in the depots domain. The reason is that PRODIGY cannot fully expand the search tree for any of the training problems, which is required for HAMLET to learn rules. In any case, none of the learners does too well. Thus, it seems that prior knowledge is needed if results are to be improved.

Table 2 shows the results of providing prior knowledge to each one of the learners in terms of test problems solved in the logistics and depots domains. In the first column, we have shown where the prior knowledge (PK) comes from: EVOCK control rules, HAMLET control rules, FF solutions, and Human control

**Table 1.** Results for PRODIGY, EVOCK, and HAMLET with no prior knowledge.

| | Logistics | | Depots | |
|---|---|---|---|---|
| System | Solved | Number of rules | Solved | Number of rules |
| PRODIGY | 21% | | 12% | |
| EVOCK | 33% | 7 | 52% | 2 |
| HAMLET | 36% | 32 | 0% | 4 |
| Human | 100% | 37 | 55% | 5 |

rules. In the case of providing the output of EVOCK to HAMLET as prior knowledge, since they use different representation languages for control rules, it is not always easy for HAMLET to use those rules. HAMLET requires that input rules follow a template that cannot always be guaranteed by EVOCK.

**Table 2.** Results for EVOCK, HAMLET, and Human with prior knowledge in the Logistics and Depots domain. Results are percentage of problems solved.

| | Logistics | | | | Depots | | | |
|---|---|---|---|---|---|---|---|---|
| PK source | | EVOCK | HAMLET | Human | | EVOCK | HAMLET | Human |
| No PK | | 33% | 36% | 100% | | 52% | 0% | 55% |
| EVOCK | 33% | - | - | - | 52% | - | 57% | - |
| HAMLET | 36% | 33% | - | 98% | 0% | 43% | - | 55% |
| FF | - | - | 48% | - | - | - | 43% | - |
| Human | 100% | 83% | 88% | - | 55% | 55% | 55% | - |

In the table we see that in the logistic domain, providing prior knowledge to both EVOCK and HAMLET improves the individual results, with any of the alternatives to supply such knowledge except when HAMLET provides the prior knowledge. EVOCK goes from 33% without prior knowledge, up to 83% when knowledge is supplied by human, and HAMLET goes from 36% up to 48% and 88%. However, we also see that the behaviour of the control rules after the learning process is worse or equal to each prior knowledge. For instance, the human rules solve 100%[2] and, after learning with that prior knowledge, EVOCK solves only 83%, and HAMLET 88%. This shows that the overall learning task is a hard one, even for sophisticated systems. It is also possible that the prior knowledge is by itself a local maxima, or close to, in the space of prior knowledge.

Another issue that we have elaborated further at [1] is the fact that even if the human achieves a 100% of solved problems, the effort of obtaining such set of control rules was much bigger than the effort of polishing the rules that

[2] Actually, we tested the human rules in some harder problems (10 to 50 goals) and it solved 198 out of 210.

HAMLET generated before (even if the percentage of solved problems was less, 88%).

In the depots domain, we see again that using prior knowledge allows HAMLET to improve its behaviour from 0% to 57% (EVOCK), 43% (FF), and 55% (human). However, in the last case, the human PK knowledge does all the work and HAMLET cannot improve his results. In the case of EVOCK it is not always the case. EVOCK does not benefit much from using PK: with no PK it solves 52% problems, whereas with HAMLET as PK source it decreases to 43% and with the human, it increases slightly to 55%. FF cannot serve as PK source for EVOCK.

## 5 Conclusions

We have presented in this paper three ways of providing prior knowledge to relational learning systems in planning tasks: using another learning system output as initial seed of the learning task; using a human to supply also an initial state for the search of sets of control rules; and using another planner for providing knowledge in terms of a solution, instead of an initial description of the target concept. We have compared these three approaches with not using prior knowledge, and we have seen that in most cases, prior knowledge in the form of an initial target concept definition (set of control rules) or another type of information (solution to a planning problem) can indeed improve the behavior of the learning system. However, prior knowledge itself is not usually improved, which shows that the learning task is very hard.

## References


1. Ricardo Aler and Daniel Borrajo. On control knowledge acquisition by exploiting human-computer interaction. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pages 141–151, Toulouse (France), 23-17 April 2002. AAAI Press.
2. Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Learning to solve planning problems efficiently by means of genetic programming. *Evolutionary Computation*, 9(4):387–420, Winter 2001.
3. Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, 141(1-2):29–56, October 2002.
4. Fahiem Bacchus. AIPS'00 planning competition. The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems. *AI Magazine*, 22(3):47–56, Fall 2001.
5. Daniel Borrajo, David Camacho, and Andres Silva. Multistrategy relational learning of heuristics for problem solving. In *Proceedings of the Nineteenth SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, Cambridge, UK, Diciembre 1999.
6. Daniel Borrajo and Manuela Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 11(1-5):371–405, February 1997.

7. Saso Dzeroski, De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.
8. Tara A. Estlin and Raymond J. Mooney. Multi-strategy learning of search control for partial-order planning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume I, pages 843–848, Portland, Oregon, August 1996. AAAI Press/MIT Press.
9. Tara A. Estlin and Raymond J. Mooney. Learning to improve both efficiency and quality of planning. In Martha Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1227–1232. Morgan Kaufmann, 1997.
10. Richard E. Fikes, P. E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.
11. Jörg Hoffmann and Bernhard Nebel. The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
12. Yi-Cheng Huang, Bart Selman, and Henry Kautz. Learning declarative control rules for constraint-based planning. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, Stanford, CA (USA), June-July 2000.
13. Subbarao Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. PhD thesis, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD, 1989.
14. Subbarao Kambhampati. Improving Graphplan's search with EBL & DDB techniques. In Thomas Dean, editor, *Proceedings of the IJCAI'99*, pages 982–987, Stockholm, Sweden, July-August 1999. Morgan Kaufmann Publishers.
15. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
16. Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, MA, 1988.
17. Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
18. Yong Qu and Subbarao Kambhampati. Learning search control rules for plan-space planners: Factors affecting the performance. Technical report, Arizona State University, February 1995.
19. J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, August 1990.
20. A. Srinivasan, R.D. King, and M.E. Bain. An empirical study of the use of relevance information in inductive logic programming. *Machine Learning Research*, 4(369-383), 2003.
21. Marcel Turcotte, Stephen Muggleton, and Michael J.E. Sternberg. The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, 14(1-2):81–96, 2001.
22. Manuela Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.
23. Manuela Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, 7:81–120, 1995.