# Using Previous Experience for Learning Planning Control Knowledge

**Susana Fernández** **Ricardo Aler** **Daniel Borrajo**
Universidad Carlos III de Madrid
Avda. de la Universidad, 30
28911 Leganés (Madrid), España
sfarregu,aler@inf.uc3m.es, dborrajo@ia.uc3m.es

**Abstract**

Machine learning (ML) is often used to obtain control knowledge to improve planning efficiency. Usually, ML techniques are used in isolation from experience that could be obtained by other means. The aim of this paper is to determine experimentally the influence of using such previous experience or prior knowledge (PK), so that the learning process is improved. In particular, we study three different ways of getting such experience: from a human, from another planner (called FF), and from a different ML technique. This previous experience has been supplied to two different ML techniques: a deductive-inductive system (HAMLET) and a genetic-based one (EVOCK).

## Introduction

Planning in non-trivial applications is a PSpace-hard task that has to be guided by human or machine generated knowledge in order to efficiently solve problems. In the context of machine learning, several approaches have been used to supply that guidance, varying from Case-Based Reasoning, as in (Kambhampati 1989; Veloso 1994), to pure EBL as in (Kambhampati 1999; Minton 1988; Qu & Kambhampati 1995) or macrooperators (Fikes, Hart, & Nilsson 1972). More recent approaches combine techniques, as reinforcement learning with ILP (Dzeroski, Raedt, & Driessens 2001). Other approaches combine deductive and inductive methods (Estlin & Mooney 1997; Huang, Selman, & Kautz 2000).

One of the first approaches that combined deductive learning techniques, as EBL, with inductive learning techniques was HAMLET (Borrajo & Veloso 1997). Later, we developed another learning technique that could be considered as relational learning, and used a genetic programming approach, EVOCK (Aler, Borrajo, & Isasi 2002). Both approaches generated control knowledge in terms of control rules to be used by the PRODIGY planner (Veloso *et al.* 1995). We showed that the behaviour of EVOCK depends very much on the prior (background) knowledge (PK) that we provided to it, in the form of an initial population (Aler, Borrajo, & Isasi 2002), or an auxiliary population that could be used for the crossover operators (Aler, Borrajo, & Isasi 2001). 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 (Aler & Borrajo 2002).

In this paper, we wanted to deepen in the study of the effect of providing PK outlined in the previous works, by placing all the results together for comparison purposes and adding a new form of introducing PK, such as the use of another planner. We have selected for this study the two relational learning systems, HAMLET and EVOCK, which use very different learning biases, so that results can be generalized. Therefore, they have been supplied previous experience by means of three different strategies: a human provides the PK in terms of a set of control rules; one 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.

In the context of non-linear planning, some learning systems have opted to provide previous experience for learning one way or another. For instance, Q-RRL (Dzeroski, Raedt, & Driessens 2001) 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 PK (Estlin & Mooney 1996). 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.

The rest of the paper is organized as follows. The next section overviews the planner, and the machine

learning techniques that we have used for the experiments (they have been extensively covered in previous works). The second section describes how to incorporate PK into each machine learning technique. The third section presents the experiments that we have performed and their outcome. And, finally, last section draws some conclusions from the work.

## 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.

### Prodigy

PRODIGY is a nonlinear planning system that follows a means-ends analysis (Veloso *et al.* 1995). 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 planning standard PDDL2.1 (Fox & Long 2002).

From a control knowledge acquisition perspective, PRODIGY planning/reasoning cycle, involves several decision points. Figure 1 shows an schematic view of a generic search tree with all those decisions. 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 (Veloso *et al.* 1995) 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 is usually required, thus reducing planning efficiency.

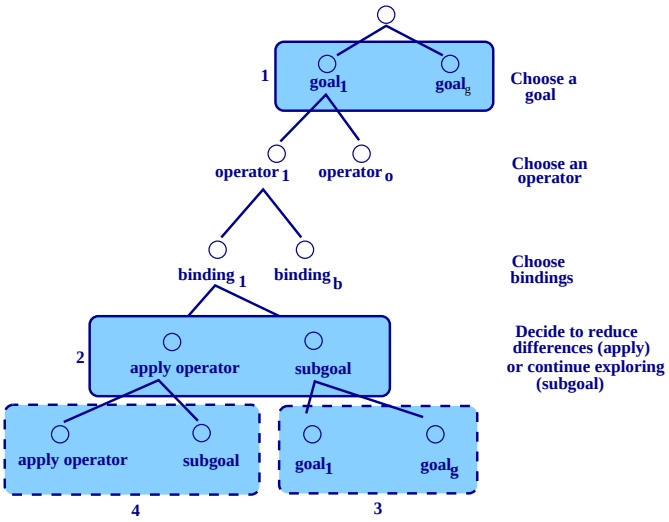


Figure 1: Generic PRODIGY search tree.

### Hamlet

HAMLET is an incremental learning method based on EBL (Explanation Based Learning) and inductive refinement of control rules (Borrajo & Veloso 1997). 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, a control 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 (Borrajo & Veloso 1997)). These EBL like 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 of decisions of the same type. 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).

Figure 2 shows an example of control rule generated by HAMLET for the logistics domain. In the logistics domain, packages (objects) have to be moved from one location to another among cities. There are two types of carriers: airplanes, that can transport packages be-

[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.

tween two airports; and trucks, that can only move packages among locations in the same city. The rule determines when the `unload-airplane` operator must be selected for achieving the goal of having an object in an airport of another city where it is now.

```
(control-rule select-operators-unload-airplane
   (if (current-goal (at <object> <location1>))
       (true-in-state (at <object> <location2>))
       (true-in-state (loc-at <location1> <city1>))
       (true-in-state (loc-at <location2> <city2>))
       (different-vars-p)
       (type-of-object <object> object)
       (type-of-object <location1> airport))
   (then select operator unload-airplane))
```

Figure 2: Example of a control rule for selecting the `unload-airplane` operator in the logistics domain.

## EVOCK

We only intend to provide here a summary of EVOCK and refer to (Aler, Borrajo, & Isasi 2002) for details. EVOCK is a machine learning system for learning control rules based on Genetic Programming (GP) (Koza 1992). 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 (each set is an individual) that can be either randomly generated, or initialised with some PK. Figure 3 shows an example of an individual in the blocksworld domain.

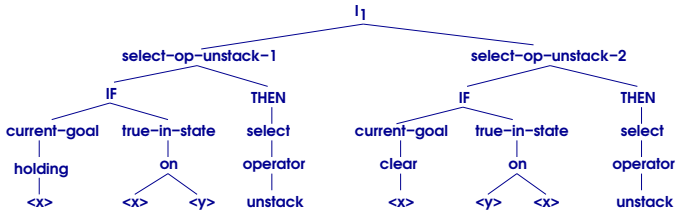


Figure 3: Example of an EVOCK individual.

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 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 standard 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.

## 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.

### Providing prior knowledge to HAMLET

We have devised two different ways of providing PK 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. These rules cannot be specialised given that they do not come from a bounded explanation, but were given directly as they are by an external source. HAMLET's refinement module 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 (Hoffmann & Nebel 2001) 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 in order to learn something out of it. 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 (a search tree corresponding to this solution). Basically, it provides the order based 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.

Using this new set of rules, PRODIGY generates a solution path corresponding to the solution provided by the other planner. Afterwards, we let PRODIGY continue searching for different solutions, so that HAMLET can generate control rules from the decisions that led to better solutions if any was found within the learning time limit. Figure 4 shows a schema of the approach.

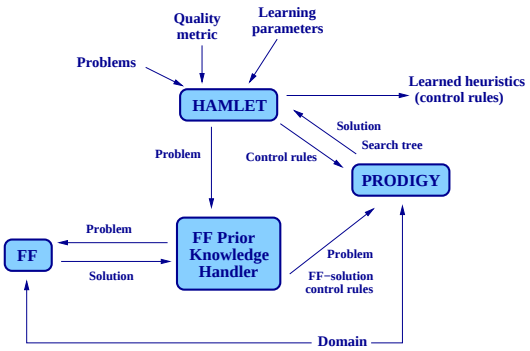


Figure 4: Providing prior knowledge to HAMLET by using another planner, FF.

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 adding them into the control rules conditions.

### 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 another machine learning technique. 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.

Figure 5 shows how HAMLET rules are used as PK 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.

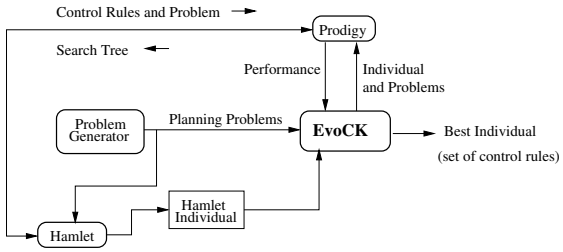


Figure 5: Prior knowledge for EVOCK.

## Experiments and results

For the experiments, we have used two commonly used domains in previous planning competitions: logistics (Bacchus 2001) and depots (from the 2002 competition). We have already described the logistics domain, and the depots domain is a similar one, where crates should be moved among depots and distributors. There are trucks for movement, and hoists to lift crates from places and trucks. Crates can also be stacked one top of another, as in the blocksworld. The depots domain is a specially hard domain as it can be seen in the results of the competition.

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. The training problems must be of small size to guarantee that the planner finds all the

possible solutions to provide the positive and negative decision points made in the search tree necessary for the learning. Then, we provided PK in the different forms that we explained in the previous section to each learning system, and tested against randomly generated test problems. In the logistics, we used 120 test problems ranging from 1 to 5 goals. In the depots, we used 60 test problems also ranging from 1 to 5 goals. The time limit used in both domains was 15 seconds, which is a rather small one. We have verified experimentally that increasing this time limit does not contribute to any significant improvement. For problems of this size, if a solution is not reached within very few seconds, the probability of finding a solution with a time limit an order of magnitude larger is very small.

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 time devoted to hand-generated control knowledge was about 8 hours in the depots domain. In the logistics domain, it is difficult to determine because the expert has been working with that domain for about 10 years. We would like to study in the future the impact that people with different expertise have on the definition by hand of such knowledge.

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

| | Logistics | | Depots | |
|---|---|---|---|---|
| System | Solved | Nº rules | Solved | Nº rules |
| PRODIGY | 21% | | 12% | |
| EVOCK | 33% | 7 | 52% | 2 |
| HAMLET | 36% | 32 | 0% | 4 |
| Human | 100% | 37 | 55% | 5 |

The main conclusion from the analysis of the results 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 PK might be 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 PK comes from: EVOCK control rules, HAMLET control rules, FF solutions, and Human control rules. In the rest of columns, results are shown for each of the systems with and without PK. In the case of providing the output of EVOCK to HAMLET as PK, since they use different representation languages for control rules, it is not always easy for HAMLET to use these rules. HAMLET requires that input rules follow a template that cannot always be guaranteed by EVOCK. This could be achieved in the logistics domain but not in depots. We have not devise yet ways of allowing FF to serve as PK source for EVOCK.

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

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

In the table we see that in the logistics domain, providing PK to both EVOCK and HAMLET improves over the no PK situation, with any of the alternatives except when HAMLET provides the PK. EVOCK goes from 33% without PK, up to 83% when knowledge is supplied by the human, and HAMLET goes from 36% without PK, up to 48% when it is supplied by FF and 88% when it is supplied by the human. However, we also see that the learning does not improve the human rules. For instance, the human rules solve 100%[2] (as Table 1 showed) and, after learning from that PK, EVOCK and HAMLET actually worsen results (83% and 88%, respectively). Actually, the only case where PK can be considered useful is when FF is used to improve HAMLET, because both PK and HAMLET are improved.

In the depots domain, we see again that using PK allows HAMLET to improve its behaviour from 0% to 57% (EVOCK), 43% (FF), and 55% (human). In the case of EVOCK, it does not benefit 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%. The human rules are not improved (results remain constant at 55%). Again, there are only two cases where both PK and learning system are improved: using EVOCK and FF as PK for HAMLET.

## Conclusions

We have presented in this paper three ways of providing PK to 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 previous experience in terms of a solution. We have compared these three approaches with not using PK, and we have seen that in most cases, PK in the form of an initial set of control rules or another type of information (solution to

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

a planning problem) can improve the behaviour of the learning system. However, there are only a few cases where both the PK and the learning system are improved. Specifically, we have shown that using a solution as PK is a consistent way of improving control rule learning.

The aim of this paper is to perform a preliminary study of the effect of providing PK for learning and we've done it by counting the number of problems solved. However, in the future it would be interesting to use different metrics in comparing systems such as the average planning time, number of operators in the solutions or any other quality measure.

## References


Aler, R., and Borrajo, D. 2002. On control knowledge acquisition by exploiting human-computer interaction. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, 141–151. Toulouse (France): AAAI Press.

Aler, R.; Borrajo, D.; and Isasi, P. 2001. Learning to solve planning problems efficiently by means of genetic programming. *Evolutionary Computation* 9(4):387–420.

Aler, R.; Borrajo, D.; and Isasi, P. 2002. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence* 141(1-2):29–56.

Bacchus, F. 2001. AIPS'00 planning competition. The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems. *AI Magazine* 22(3):47–56.

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning* 11(1-5):371–405.

Dzeroski, S.; Raedt, D.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43:7–52.

Estlin, T. A., and Mooney, R. J. 1996. Multi-strategy learning of search control for partial-order planning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume I, 843–848. Portland, Oregon: AAAI Press/MIT Press.

Estlin, T. A., and Mooney, R. J. 1997. Learning to improve both efficiency and quality of planning. In Pollack, M., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1227–1232. Morgan Kaufmann.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

Fox, M., and Long, D. 2002. *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. University of Durham, Durham (UK).

Hoffmann, J., and Nebel, B. 2001. The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Journal of Artificial Intelligence Research* 14:253–302.

Huang, Y.-C.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In Langley, P., ed., *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*.

Kambhampati, S. 1989. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. Ph.D. Dissertation, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD.

Kambhampati, S. 1999. Improving Graphplan's search with EBL & DDB techniques. In Dean, T., ed., *Proceedings of the IJCAI'99*, 982–987. Stockholm, Sweden: Morgan Kaufmann Publishers.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

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

Qu, Y., and Kambhampati, S. 1995. Learning search control rules for plan-space planners: Factors affecting the performance. Technical report, Arizona State University.

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. 1994. *Planning and Learning by Analogical Reasoning*. Springer Verlag.