Knowledge Representation Issues in Control Knowledge Learning

Ricardo Aler aler@inf.uc3m.es
Daniel Borrajo dborrajo@ia.uc3m.es
Pedro Isasi isasi@ia.uc3m.es
Departamento de Informática, Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911 Leganés, Madrid, Spain

Abstract

Knowledge representation is a key issue for any machine learning task. There have already been many comparative studies about knowledge representation with respect to machine learning in classification tasks. However, apart from some work done on reinforcement learning techniques in relation to state representation, very few studies have concentrated on the effect of knowledge representation for machine learning applied to problem solving, and more specifically, to planning. In this paper, we present an experimental comparative study of the effect of changing the input representation of planning domain knowledge on control knowledge learning. We show results in two classical domains using three different machine learning systems, that have previously shown their effectiveness on learning planning control knowledge: a pure EBL mechanism, a combination of EBL and induction (HAMLET), and a Genetic Programming based system (EVOCK).

1. Introduction

Knowledge representation is a key issue for any machine learning task. There have already been some comparative studies about knowledge representation with respect to machine learning in classification tasks. There is even a set of topics devoted to changing the input representation of the examples for improving the learning task with issues such as feature selection (Kohavi & John, 1997) or constructive induction (Donoho & Rendell, 1995). However, apart from some work done on reinforcement learning techniques in relation to state representation (Lin, 1993), very few studies have concentrated on the effect of knowledge representation for machine learning applied to problem solving, and more specifically, to planning. In (Qu & Kambhampati, 1995), the authors present results in three variations of the blocks world. They discuss differences with respect to using conditional effects, and/or universal quantifiers. But, they only compared with respect to variations of the same technique: EBL.

One of the main differences between machine learning applied to problem solving and classification is that problem solving tasks required for most systems to solve some set of problems given by an initial state and a set of goals to generate the learning examples. Changing the representation of the domain theory can change the solvability horizon of problems, and therefore, the ability to generate learning examples, given that most learning systems require finding a solution to learn. In this paper, we present an experimental comparative study of the effect of changing the input representation of domain knowledge for machine learning in planning tasks. We show results in two classical domains using three different machine learning systems, that have previously shown their effectiveness on learning planning control knowledge: a pure EBL mechanism (Minton, 1988), a combination of EBL and induction (HAMLET) (Borrajo & Veloso, 1997), and a genetic programming driven system (EVOCK) (Aler et al., 1998).

2. The Learning Task

The goal of the three systems considered in this article is to learn control knowledge for a state space planner called PRODIGY (Veloso et al., 1995). PRODIGY4.0 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 hierarchy;

Table 1. Example of a control rule for selecting the unstack operator.

```
(control-rule select-operator-unstack
     (if (and (current-goal (holding <object1>))
              (true-in-state (on <object1> <object2>))))
     (then select operator unstack))
```

- 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.

PRODIGY4.0 's planning/reasoning cycle, involves several decision points, namely:

- *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, PRODIGY4.0 might make the wrong decisions at some points, requiring backtracking and reducing planning efficiency. Table 1 shows an example of control knowledge represented as a rule to determine when the operator `unstack` must be selected. Control knowledge can be handed down by a programmer or learned automatically. The goal of this paper is to observe the effects of different representations of the same domain on learning control knowledge for that domain.

3. Description of the Learning Systems

In this Section, the three systems (EBL, HAMLET and EVOCK) involved in the experimental study will be described. As EBL is actually one of the subcomponents of HAMLET, EBL and HAMLET will be described in the same Subsection.

3.1 EBL and HAMLET

HAMLET is an incremental learning method based on EBL (Explanation Based Learning) and inductive refinement (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. HAMLET's Bounded Explanation module is the EBL system that has been used in this paper. Figure 1 shows HAMLET modules and their connection to PRODIGY.

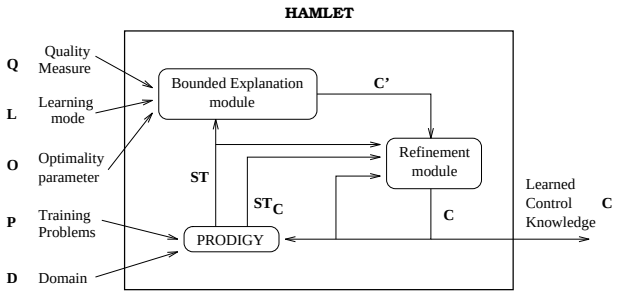

Figure 1. HAMLET's high level architecture.

The Bounded Explanation module generates control rules from a PRODIGY4.0 search tree. The details can be found in (Borrajo & Veloso, 1997). These rules will be referred to as EBL rules in the experimental Section. The EBL rules might be overly specific or overly general. HAMLET's Refinement module solves the problem of being overly specific by generalizing rules when analyzing 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). ST and $ST_{\mathcal{C}}$ are planning search trees generated by two calls to PRODIGY4.0 planning algorithm with or without control rules (respectively. $\mathcal{C}$ is the set of control rules, and $\mathcal{C}'$ is the new set of control rules learned by the Bounded Explanation module.

3.2 EVOCK

We only intend to provide a summary of EVOCK and refer to (Aler et al., 1998) for details. EVOCK is a machine learning system for learning control rules based

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

on Genetic Programming (GP) (Koza, 1992). GP is an evolutionary computation method that has been used for program induction and machine learning. GP searches in the space of computer programs, trying to find a "good enough" computer program according to some metric. GP can be seen as a kind of heuristic beam search. Initially, the beam (or population) is made up of randomly generated computer programs (or individuals). These individuals are selected according to a heuristic (or fitness) function and modified by means of the so called genetic operators.

In EVOCK, the individuals are sets of control rules that are manipulated by EVOCK's genetic operators. EVOCK's individuals are generated and modified according to a grammar that represents the language provided by PRODIGY4.0 for writing correct control rules. EVOCK's 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's guiding heuristic -the fitness function- 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 because they run faster).

4. Experiments and Results

In this section, we will first describe the variations on representation of domain theories. Then, we will present the training and testing setups, and, finally, we will present the obtained results and a discussion on those results. It is very important to bear in mind that the goal of this paper is to analyze the effect of changing the representation of a domain theory in planning for several learning systems. Therefore, we have intentionally not compared the different learning systems among them.

4.1 Domain Theories Representation

In order to analyze the impact of knowledge representation on the task of learning problem solving control knowledge, we defined the following domains relative to the blocks world:[2]

- Standard blocks world (`b`): four operators (unstack, stack, pick-up, and put-down) with a test on the ones with two arguments (unstack and stack) that checks that these arguments are not equal.
- Simplified blocks world (`sb`): two operators (put-on and new-tower) with a test on both that checks that their arguments are not equal. This version has been used in many experiments in the blocks world, and has the advantage over the previous one that there is no reasoning about the arm being empty, because movement operations on blocks are atomic; that is, they are not separated in two operations: taking the block and leaving the block.
- Reverse blocks world (`rb`): the same four operators as the standard version, but two operators (unstack and pick-up) are presented in the opposite order in the domain description file. Since PRODIGY4.0 (as most other planners) follows the order by which operators have been defined when making blind decisions, this change might affect the behavior of the learning system. For instance, learning when to select an operator might be easier than learning when to select the other one.
- Standard blocks world without functions (`bwf`): it is the standard version without the check for difference between the arguments of operators.

We also used the following versions of the logistics domain:

- Standard logistics domain (`l`): six operators (load-airplane, load-truck, unload-airplane, unload-truck, fly-airplane, and drive-truck) and predicates such as at-truck, at-airplane, at-object, inside-airplane, inside-truck. Here, we have used the standard version of the logistics domain in PRODIGY4.0 where one can define functions in the preconditions to define filters on the values of the variables of the operators. Apart from that, variables are assigned to types. For example, for unload-truck operations only trucks in the same city as the destination location of the object to be transported are selected.
- Plain logistics domain (`pl`): six operators, but we have removed the "semantics" of the predicates, leaving the above mentioned predicates in two: at and inside. This is theoretically more difficult than the previous domain, since there are more operators now that achieve those goals. For the new predicate inside, in the standard version, one would subgoal either on inside-airplane, which forces to use operator load-airplane, or on inside-truck, which forces to use operator load-truck. In

[2] We have left a version of the first two domains in `http://scalab.uc3m.es/~dborrajo/hamlet/`

this new version, since there is only one inside for both previous predicates, the planner would select both operators for any subgoal on inside. The same happens with predicate at.

- Standard logistics domain without functions (`lwf`): here, he have removed the functions calls from the preconditions, so that variables can have any value within their types (as in the case of the last version of the blocks world). This domain description is also theoretically more difficult than the standard one, given that the planner can create more bindings (substitutions) for each variable. We only report here the results of using PRODIGY, HAMLET, and EBL.

4.2 Training Setup

We used for training two randomly generated problem sets for the blocks world: the first one with 400 one goal problems and five blocks; and the second one with another 400 two goal problems and five blocks. Since representation of each domain version was different, we generated the same problem twice: one for domain versions that used the standard representation (with the arm-empty, and holding); and another for domain versions with the simplified representation (without arm-empty and holding). Since we used a random problem generator that used arm-empty and holding, we did the following to make sure that the comparison among domain versions was fair:

- Each time that arm-empty appeared in a problem, it was removed from the problem description.
- Each time that holding appeared in a problem, the block that the robot held was left on the table. Therefore, all initial configurations and goal statements had all blocks on the table or on top of other blocks, but never held by the robot. Of course, this causes some problems to become easier, because, in the previous versions, in order to hold another block, the robot first had to think on where to leave the block it was holding, while here we are leaving it directly on the table. But, it has the advantage of being fair when comparing the standard domain description with the simplified one, given that this last one does not reason about holding blocks.

In the logistics domain, we generated a training set of 400 one goal problems and five objects, airplanes, trucks, and cities. In this case, for representation reasons, the only changed we had to do in problem formulations was to change all inside-x predicates for inside, and all at-x predicates for at. We gave 200 seconds of time limit for solving each problem in both domains during learning of EBL and HAMLET.

Then, we trained the three learning systems: HAMLET, EBL, and EVOCK. HAMLET was trained as explained in previous sections. Since there are many different eays to train the learning systems, we performed the tests always using the best configuration found for each learning system. In the case of HAMLET, results using the 800 training problems were worse than using only the first 400, so we used for comparison the results of using those 400 problems. EBL learned rules from all decisions that were not the first alternative tried in each node. Then, a utility analysis was performed. According to the usual equation (Minton, 1988),

$$u(r) = s(r) \times p(r) - m(r)$$

where $u(r)$ is the utility of a rule r, $s(r)$ is the time that the rule saves when used, $p(r)$ is the probability that the rule would match, and $m(r)$ is the cost of using the rule (matching time). Therefore, we estimated the following variables:

- $s(r)$: given that each rule is learned from a node in a search tree, we set $s(r)$ to the number of nodes below the node from where it learned the rule, multiplied by the time PRODIGY4.0 takes to expand a node. A better estimate would have been computed by using a second training set of problems as Minton did.
- $p(r)$: we estimated it as the number of times that PRODIGY4.0 tried to use the rule in subsequent problems in the training phase divided by the number of times that it actually fired. As before, a better estimate would be to use a second training set.
- $m(r)$: it is estimated as the total matching time added over all times that it tried to match the rule divided by the number of times that it tried to match it.

After training, the utility $u(r)$ of each rule is computed, and rules that do not have a number of times that it fired greater than one are removed. Also, we used two utility thresholds: 0 and 0.5. So, rules that do not have a higher utility than 0/0.5 are removed. This generates two versions of EBL for each domain version. Since a threshold of zero still keeps many rules, we have only used 0.5 for the experiments reported here.

In the case of EVOCK, since it is a stochastic method, we performed around 25 experiments for each configuration. we will present the result for the best individual, and the average for all individuals and the standard deviation.

4.3 Testing Setup

For testing, we generated problems of diverse complexity with the same random procedure described before. In the blocks world, we generated 18 test sets each with ten problems. We varied the number of blocks and goals, having the following configurations of (goals,blocks): (1,5), (1,10), (1,20), (1,50), (2,5), (2,10), (2,20), (2,50), (5,5), (5,10), (5,20), (5,50), (10,10), (10,20), (10,50), (20,10), (20,20), (20,50), and (50,50). We did something similar for the logistics domain, generating also the configurations: (10,5), (20,10), and (50,20).

Time limit for test problems was set to 150 for all configurations and domains. Segre et al. (1991) discuss the effect of defining a time limit for a fair comparison, given that if one would increase that limit, the learning systems might find solutions, while the base problem solver might not. While the authors are theoretically right, we did not find experimentally a big difference by multiplying the time limit by two or three. First, we believe that, in order to find a big difference with the reported results in terms of solvability, one would have to increase in an order of magnitude (or more) the time limit. In fact, we would like to perform those experiments in the near future.

Second, suppose (which is not far from reality in most cases, at least, in ours) that the goal of using learning systems for problem solving is to attain an optimal behavior after learning: in each decision of the search tree, select the alternative that will lead directly to the best solution without exploring failure branches. If that is the goal of the research, one would have to admit as a (partial) failure (even if we do not do it) when the learning system does not learn to force that optimal behavior. Optimal behavior in the cases of all test problems that we have generated (up to 50 goals and objects/blocks), would mean in PRODIGY4.0 to expand a number of nodes which is equal to $4 \times o$, where o is the number of operators in the solution. For 50 goal problems the maximum number of operators in an optimal solution is 200 for the blocks world and 500 for the logistics domain. Given that the machines we are using expand a node in 2 milliseconds, it would need at most $4 \times 500 \times 0.002 = 4$ seconds for solving optimally each test problem. Therefore, imposing a time limit of 150 seconds for each problem is way beyond the time that we should expect an optimal learnt knowledge to solve each problem. If it does not solve the problem in that time, we will have to work harder (as it is the case, given that learning in problem solving is not a solved task).

We measured the following variables:

- number of learned rules by each learning system;
- number of solved problems: if a configuration found a solution in the time limit defined before;
- delta-time: since we wanted to compare the effect of representation in various domain descriptions for learning, we used the difference between the time that PRODIGY4.0 spent to solve each problem and the time that the learning system spent; and
- delta-solution-length: computed as the difference of the number of operators in the solutions by PRODIGYand the number of operators in the solutions by the learning systems. This is an estimate of the optimality of the solutions that the planner with the learned knowledge provides when the quality metric is the solution length.

4.4 Results

Table 2 and 3 show the number of rules learned by each learning system in each domain. In the case of EBL, the tables show the number of rules used for the experiments (after utility pruning) and, in parentheses, the original number of rules (before pruning).

Table 2. Number of learned rules in the blocks world.

System	Standard	Simplified	Reverse	Without functions
HAMLET	6	11	6	6
EBL	13 (661)	5 (328)	12 (640)	16 (635)
EVOCK	3	1	2	2

Table 3. Number of learned rules in logistics.

System	Standard	Plain	Without functions
HAMLET	32	24	1
EBL	27 (297)	33 (300)	0 (30)

Tables 4 and 5 show the number of solved problems by the different learning systems and PRODIGY4.0 in the blocks world and in the logistics domain given the time limit described before. In the case of EVOCK,

the tables show the results of the best individual, the average of all individuals and the standard deviation (σ).

Table 4. Number of solved problems over 180 in the blocks world.

System	Standard	Simplified	Reverse	Without functions
PRODIGY4.0	92	169	92	92
HAMLET	128	149	128	128
EBL	90	168	89	89
EVOCK (best)	144	169	166	143
EVOCK (ave.)	102	125	120	100
EVOCK (σ)	21	41	32	37

Table 5. Number of solved problems over 210 in logistics.

System	Standard	Plain	Without functions
PRODIGY4.0	96	96	25
HAMLET	107	104	26
EBL	103	103	

Figures 2, 3 and 4 show the accumulated delta time (difference of time between the time spent by PRODIGY4.0 and the learning system) of the solved problems by all configurations and domain of HAMLET and EBL in all domain versions of the blocks world. They were also ordered by time.

Figures 5 and 6 show the accumulated delta time (difference of time between the time spent by PRODIGY4.0 and the learning system) of the solved problems by all configurations and domain of HAMLET and EBL in all versions of the logistics domain, and ordered according to increasing difficulty for PRODIGY, measured by the time that took PRODIGY4.0 to solve the problem. In the logistics domain, there is no result for the version without functions, given that being a difficult version as discussed in the next subsection, very few rules were learnt by HAMLET and EBL and the behavior was close to that of PRODIGY. In Figure 7, we show the accumulated difference in number of operators in the solution between PRODIGY4.0 and HAMLET (we do not show these data for the blocks world, given that in that case, the difference is very close to zero).

4.5 Discussion

We can perform the following analysis from the data in the previous section:

Number of learned rules (Tables 2 and 3): the only significant difference between the various domain descriptions is in the case of the logistics domain without functions, in which HAMLET could only learn one rule,

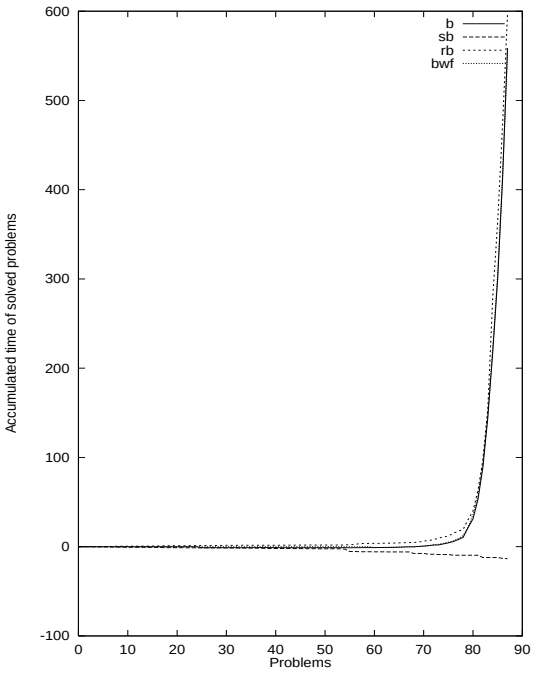

Figure 2. Accumulated delta time of solved problems by all configurations of HAMLET and PRODIGY4.0 for all variations of domain theories for the blocks world.

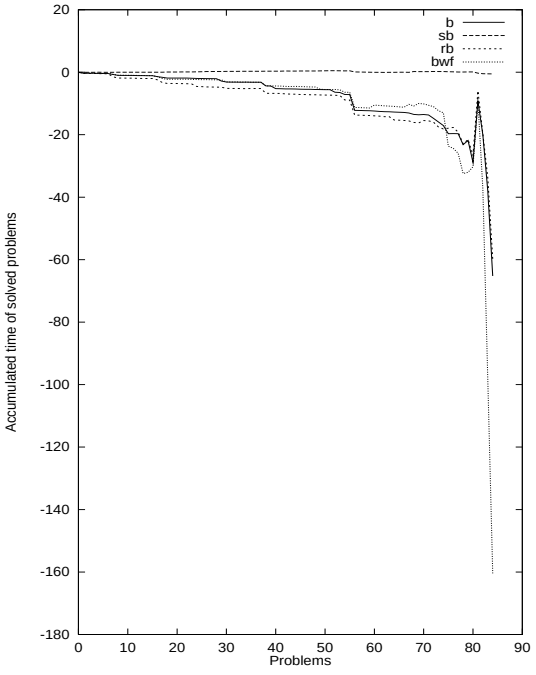

Figure 3. Accumulated delta time of solved problems by all configurations of EBL and PRODIGY4.0 for all variations of domain theories for the blocks world.

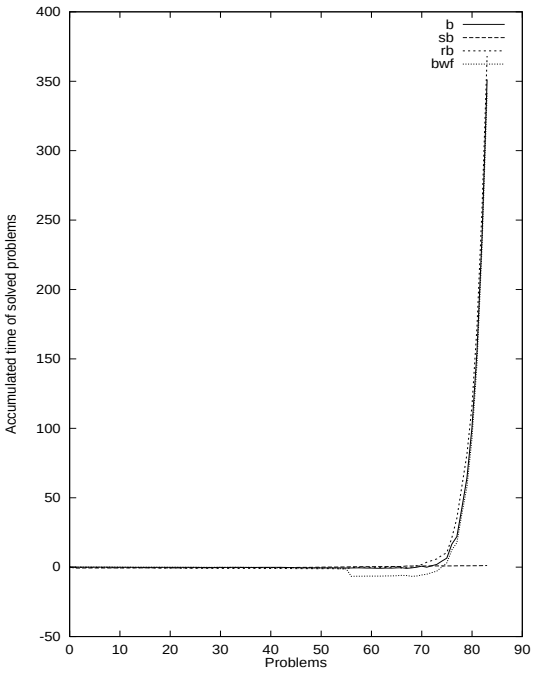

Figure 4. Accumulated delta time of solved problems by all configurations of EVOCK and PRODIGY4.0 for all variations of domain theories for the blocks world.

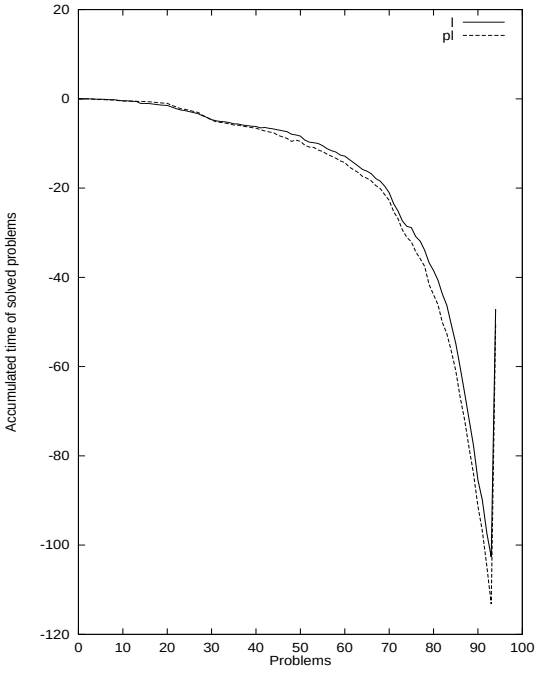

Figure 6. Accumulated delta time of solved problems by all configurations of EBL and PRODIGY4.0 for all variations of domain theories for the logistics.

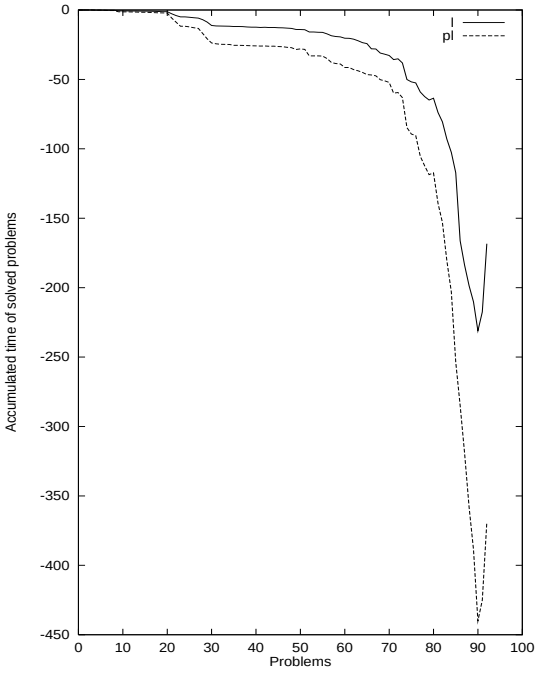

Figure 5. Accumulated delta time in the solution of solved problems by the configurations of HAMLET and PRODIGY4.0 for the best two variations of domain theories for the logistics.

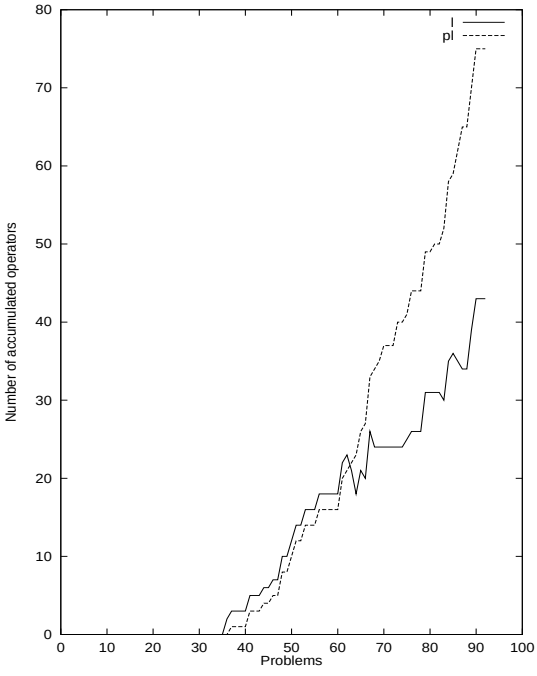

Figure 7. Accumulated difference of the number of operators in the solution of solved problems by the configurations of HAMLET and PRODIGY4.0 for the best two variations of domain theories for the logistics.

while EBL did not learn any useful rule. This is due to the difficulty that this domain poses for PRODIGY, making it hard for PRODIGY4.0 to expand the search trees. Given that both HAMLET and EBL (in this implementation) need to expand the whole tree to learn rules, they could only use very few actual training problems for learning.

Solvability (Tables 4 and 5): in the case of the blocks world, it is clear that the simplified version is easier than the rest, so all configurations solve more problems. Removing the explicit handling of the robot arm makes the domain much easier. In this domain, no learning system was able to improve over PRODIGY. Also, it is very remarkable that there is no significant difference between the rest of domain versions.

In the case of the logistics domain, the opposite happens: there is a version of the domain much harder than the rest, and it consists on removing the bindings filters implemented by functions on the preconditions of operators.

Time to solve problems (Figures 2 through 6): in the case of the blocks world, the tables clearly show that the difference between the time spent by PRODIGY4.0 and the learning systems is minimal for the simplified version, given that PRODIGY4.0 is the best option for that domain version. In the rest of versions of the domain, the differences in time to solve the problems tend to be high in absolute value. While HAMLET and EVOCK have better times than PRODIGY, EBL does worse (negative values of the difference).

In the case of the logistics domain, all configurations of learning systems use more time in the solved problems than PRODIGY. This might be caused by the fact that these systems learn more rules in the logistics domain than in the blocks world, and, therefore, the matching process takes longer. Also, with respect to the difference of solutions lengths between PRODIGY4.0 and HAMLET, HAMLET learns rules that provide better solutions (shorter in this case), specially in the case of potentially harder domain versions, such as the plain logistics domain version.

5. Conclusions

The main conclusion of this paper is that good representations (like the simplified blocks world) tend to make control knowledge learning less necessary, as far as solvavility is concerned. On the other hand, learning can alleviate the user effort for writing efficient domain theories.

In the logistics domain, one of the representations obtains very bad results, and we should expect that learning provides large gains in performance. However, if the representation is so inefficient that even simple problems cannot be solved by the base planner, then systems that learn from traces (EBL, HAMLET) will not be able to learn much. However, other systems that do not rely on previous executions of the base planner (like a simple modification of EVOCK) might overcome this problem.

References

Aler, R., Borrajo, D., & Isasi, P. (1998). Genetic programming and deductive-inductive learning: A multistrategy approach. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 10–18). Madison, Wisconsin.

Borrajo, D., & Veloso, M. (1997). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal.*, *11*, 371–405.

Donoho, S., & Rendell, L. (1995). Rerepresenting and restructuring domain theories: A constructive induction approach. *Journal of Artificial Intelligence Research*, *2*, 411–446.

Kohavi, R., & John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence.*, *97*, 273–324.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection.* Cambridge, MA: MIT Press.

Lin, L.-J. (1993). Scaling-up reinforcement learning for robot control. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 182–189). Amherst, MA: Morgan Kaufman.

Minton, S. (1988). *Learning effective search control knowledge: An explanation-based approach.* Boston, MA: Kluwer Academic Publishers.

Qu, Y., & Kambhampati, S. (1995). *Learning search control rules for plan-space planners: Factors affecting the performance* (Technical Report), Computer Science Department, Arizona State University.

Segre, A., Elkan, C., & Russell, A. (1991). Technical note: A critical look at experimental evaluations of EBL. *Machine Learning*, *6*, 183–19.

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