On Control Knowledge Acquisition by Exploiting Human-Computer Interaction

Ricardo Aler and Daniel Borrajo
Universidad Carlos III de Madrid
28911 Leganés (Madrid), España
aler@inf.uc3m.es, dborrajo@ia.uc3m.es
+34-1-624-9418, +34-1-6249459

Abstract

In the last decade, there has been a strong and increasing interest on building fast planning systems. From it, we have seen an enormous improvement in relation to the solvability horizon of current planning techniques. Most of these techniques rely on the translation of the predicate logic description of domains into propositional representations of them. Then, a fast search procedure is applied for obtaining solutions to planning problems. While this is an important step towards solving the planning problem, we believe older planners that are based on predicate logic computation can still be competitive when they are combined with learning capabilities. This is so, because they can acquire and use more abstract representations of control knowledge (CK) which are easier to describe and maintain by humans and/or automatic systems than the ones based on propositional logic.

In this paper, we present the results we have obtained with a relatively "old" planner, Prodigy4.0, powered with CK that has been acquired using a mixed human-computer collaboration. We advocate for the initial generation of CK by a learning system, and its later refinement by a human. In fact, we can iterate this process until the desired result has been achieved. We show results in the logistics domain used at AIPS'00 that are at the same level when compared with other techniques in the Track on hand-tailored planning systems (Track2).

Introduction

In the last decade, there has been a strong and increasing interest on building fast planning systems. From it, we have seen an enormous improvement in relation to the solvability horizon of current planning techniques. Most of these techniques rely on the translation of the predicate logic description of domains into propositional representations of them. Then, a fast search procedure is applied for obtaining solutions to planning problems. While this is an important step towards solving the planning problem, we believe older planners that are based on predicate logic computation can still be competitive when they are combined with learning capabilities. This is so, because they can acquire and use more abstract representations of control knowledge (CK) which are easier to describe and maintain by humans and/or automatic systems than the ones based on propositional logic.

In this paper, we present the results we have obtained with a relatively "old" planner, Prodigy4.0, powered with CK that has been acquired using a mixed human-computer collaboration. We believe and show that this is a competitive solution for solving planning problems in comparison to modern approaches.

This is so, because humans and computers can benefit from the other advantages, while overcoming the other disadvantages. On one hand, manually defining from scratch the needed CK for a given domain and planner is a hard knowledge engineering task. But, humans are very good at tuning already defined CK. On the other hand, automatic acquisition of perfect CK by a learning system is a very hard task. This is specifically so in some domains where solving one problem by the underlying planning system is a very difficult task. However, generating an acceptable first CK description is a relatively easy task for them.

Therefore, we advocate for the initial generation of CK by a learning system, and its later refinement by a human. In fact, we can iterate this process until the desired result has been achieved. We show results in the logistics domain used at AIPS'00 that are at the same level when compared with other techniques in the Track on hand-tailored planning systems (Track2).

In this paper we use two learning systems: HAMLET, and EVOCK. HAMLET inductively refines CK initially obtained by means of EBL techniques (Borrajo & Veloso 1997). EVOCK is a Genetic Programming based system which is able to evolve CK (Aler, Borrajo, & Isasi 1998). EVOCK has proven useful to evolve and correct CK knowledge generated by HAMLET, and it will be used in that way in this article. Therefore, we will show learning sequences like HAMLET-human-EVOCK-human.

Planning and learning systems

This section describes the planner we have used in this article (PRODIGY4.0) and the two learning systems used to obtain and correct CK (HAMLET and EVOCK).

PRODIGY4.0

In this work, we have used a state space planner called PRODIGY4.0 (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;
- 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:

- *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 PRODIGY4.0. In this paper it is enough to see the planner as a program with several decision points that can be guided by CK. If no CK is given, PRODIGY4.0 might make the wrong decisions at some points, requiring backtracking and reducing planning efficiency. Figure 1 shows an example of CK represented as a rule to determine when the operator `unstack` must be selected. CK can be handed down by a programmer or learned automatically.

```
(control-rule select-operator-unstack
     (if (and (current-goal (holding <object1>))
              (true-in-state (on <obj1> <obj2>))))
     (then select operator unstack))
```

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

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. Figure 2 shows HAMLET modules and their connection to PRODIGY4.0.

The Bounded Explanation module generates control rules from a PRODIGY4.0 search tree. The details can be found in (Borrajo & Veloso 1997). The rules might be overly specific or overly general. HAMLET's Refinement module

[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.

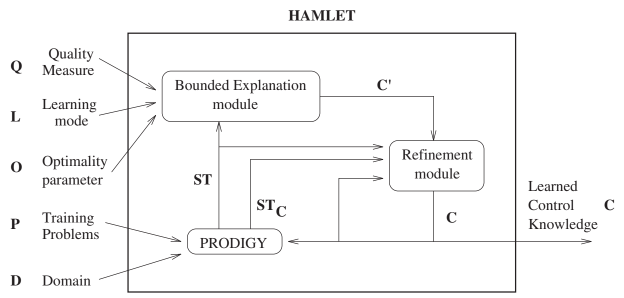

Figure 2: HAMLET's high level architecture.

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.

EVOCK

We only intend to provide a summary of EVOCK and refer to (Aler, Borrajo, & Isasi 1998) for details. EVOCK is a machine learning system for learning control rules based on Genetic Programming (GP) (Koza 1992). GP can be seen as a kind of heuristic Beam Search with its own idiosincracies. 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).

Experiments and Results

We have conducted two experiments. The goal of the first experiment is to show several ways of generating control knowledge for planning and how they can be effectively combined. We will use a base planner, PRODIGY4.0, together with the learning systems that we have discussed in previous sections, HAMLET and EvoCK. Also, we will complement these automatic generation techniques with a manual process of refinement of the control knowledge to explore other alternatives. We have used this approach in the logistics domain.

The goal of the second experiment was to explore the same idea, applying it to a temporal version of the logistics domain used in (Hasslum & Geffner 2001). This version is very similar to the previous one, but, in this case, trucks can move between cities, and each operator in the domain has a cost in terms of time.

First Experimental Setup

At the initial step, our goal was to study the possibility that a general purpose "classical" planner, such as PRODIGY4.0, could compete in the AIPS planning competitions. Obviously, by itself this was not possible. In some domains, such as the version of the logistics that was used in the competitions, PRODIGY4.0 could not solve any problem (under the very strict time bounds that we imposed). In others, such as the blocksworld, or the gripper, the results (in terms of consumed time and solvability) were far from what others achieved. Therefore, the only chance to make it competitive was to rely on the generation of control knowledge. This was not very far fetched, given that, in the last competition and in the next one, there is a special track for hand-tailored systems, which allows planners with control knowledge defined to participate.

The second step was to generate such control knowledge. There are two approaches mainly for coding such knowledge: manual or automatic. We wanted to explore in this study a mixed approach, in which control knowledge would be generated first by an automatic (machine learning) technique and then refined by a human. Also, we wanted to explore the combination of two different machine learning techniques to exploit the best of each technique. So, we tried a machine learning system, HAMLET, to generate a first version of the control knowledge. The problem that we found is that this system needs that the planner solves at least the training problems. In fact, the best HAMLET configuration requires that all the search tree has been explored. This is possible when solving "easy" problems. However, in this domain representation, even the theoretically easy problems (one and two goals problems) were hard for base PRODIGY4.0 to generate the whole search tree. This was already discussed in (Aler, Borrajo, & Isasi 2000).

Therefore, we employed a different strategy: use a more efficient representation of this domain to learn from, and then translate the generated control knowledge in this domain representation into the harder representation of the domain. The more efficient version is the standard one in the PRODIGY4.0 distribution for the logistics domain. Instead of having the predicate `at`, we have a predicate for each type of package and/or carrier. So, the predicates are: `at-object`, `at-airplane`, and `at-truck`. The same happens with the `inside` predicate. There are also some differences in the way the assignment of a truck to a city is performed, and some other minor (from this paper's point of view) changes. Another issue that allows this more efficient version of the logistics domain to explore complete search trees is that it uses lisp functions to constrain the values of variables when instantiating them in the operators. We have shown elsewhere that this allows a very rich representation of domains, which can be used to reason about time, resource or quality constrains (Borrajo, Vegas, & Veloso 2001; Rodríguez-Moreno, Borrajo, & Meziat 2001).

We trained HAMLET with 1200 problems of one, two and three goals to up to five packages. It generated a set of 32 control rules. This first set of control knowledge translated (very easily) from the efficient representation of the domain into the less efficient representation will be called HAMLET. Once we have this first control knowledge, we tested and saw that it could not solve the Track2 problems (see Results Subsection). So, the second step was to use a human to start with this learned knowledge and manually refine it. This task was rather easy for someone knowledgeable on how control knowledge works in PRODIGY4.0 as also happens with other hand-tailored systems (Bacchus & Kabanza 2000). It took two evenings of one person.

We used two strategies to perform this refinement. The first one consisted on manually inspecting the control rules to find erroneous preconditions of those rules. The second strategy consisted on letting those rules solve easy problems (one-two goals problems) and detecting when the solution path (sequence of search nodes that go from the root to the first solution found by the planner) was not the first path explored.[2] This means that the planner, using that control knowledge, had to backtrack; so there is still room for improvement. Ideally, the optimal planner would go directly (without backtracking) to the best solution. In case we found a search node in which the solution path was not the first one, we would manually create a control rule for that decision, and would run all problems again to see another instance of this phenomenon. We did this analysis with a set of 100 easy problems (one and two goals) until PRODIGY with the learned CK solved them all optimally. This second version of the control knowledge will be called HAMLET-HUMAN. After using this knowledge, the results showed that its performance was comparable to that of the systems entering in the hand-tailored track at AIPS'00.

Then, we wanted to explore a further step that would be an automatic refinement of this knowledge by another machine learning system that is able to explore areas of the search space (of sets of control knowledge) that could have been ignored by the human. Also, it could happen that not all rules in the set of control knowledge were used, or were appropriate. So, we used EvoCK to perform a GP-based search for this refinement, by seeding the initial population with the HAMLET-HUMAN CK. This will be called the HAMLET-HUMAN-EVOCK version. In the case of using EvoCK, we used three approaches: letting EvoCK evolve the complete set of control rules (HAMLET-HUMAN-EVOCK-1 configuration); letting it only generate new control rules by fixing the set generated in the HAMLET-HUMAN configuration

[2]Trying to do so for harder problems is a difficult task, since one has to know what the solution is and test at each decision step whether the planner was on the track of a solution path or not.

(HAMLET-HUMAN-EVOCK-2 configuration); and a similar configuration to the first one in which we used a different fitness function (HAMLET-HUMAN-EVOCK-3 configuration). Since this process depends on a random search, we performed several searches (experiments), at least 15 from each configuration, and selected the best individuals from all searches according to a validation set, different to the training and Track2 sets.

The GP settings were:

- Training problems: 100 one goal problems and 100 two goals problems.
- Population size: 2 (this population size amounts to some sort of hill climbing. In previous studies we have shown that this is enough for our purposes (Aler, Borrajo, & Isasi 1998)).
- Number of evaluations: 100,000

Finally, to study another refinement step, we manually analysed one of the EVOCK generated individuals (CK) in order to produce a new version called HAMLET-HUMAN-EVOCK-HUMAN.

First Results

The results of the experiments are shown in Table 1. Our experiments were performed in a Pentium II 400 Mhz, with 384 Mb of RAM. We set a time bound of 100 seconds. In the table, we also show the results of the best hand-tailored systems of the AIPS'00 competition, having in mind that a better machine was used for this competition (500MHz Pentium III with 1GB of RAM):

- TALPLANNER is an "ad-hoc" version which was built at Linkoping Univ. (Sweden). It is based on a temporal logic planner developed by Bacchus and Kabanza (Bacchus & Kabanza 2000).
- PBR was built at ISI (Univ. Southern California) and is based on the planning by rewriting technique (Ambite & Knoblock 2001)
- SHOP was built at Univ. of Maryland and is based on an HTN planner (Nau *et al.* 1999)
- R was built at The Hong Kong University of Science & Technology and it is based on the compilation on a set of propositional theories from a domain description (Lin 2000).

We show the number of solved problems of the Track2 in the logistics domain, the total time (when a planner was not able to solve a problem in the time bound, we have added our time bound to the total time, 100 seconds), the length of the solution, and the number of rules in the configuration. We will refer to quality as the length of the solution, though in the future, this aspect will improve with user-defined quality functions to be incorporated into the planners (Borrajo, Vegas, & Veloso 2001).

Second Experimental Setup

In the second experiment we wanted to explore the same idea when a quality measure was defined. In this case, we used a simple version of time, by which each operator had a different number of execution time steps: both loading and unloading take 1 step; flying airplanes takes 3 steps; driving trucks between cities takes 11 steps; and driving trucks within a city takes 3 steps. This was used in (Hasslum & Geffner 2001) to show how HSP*, a version of HSP (Bonet & Geffner 2001), could handle cost metrics. We could have used a more complex definition of time that could be dependent on the type of truck and/or airplane (Borrajo, Vegas, & Veloso 2001).

The sequence was similar to the previous one. We first trained HAMLET with 200 problems in this domain with the quality parameter set to `time` instead of the default one (plan length). It learned a set of 29 control rules that we will call HAMLET-200. Then, we refined these rules by hand, generating a new set of 5 control rules that we will call HAMLET-200-HUMAN. Then, we performed the same two operations using 200 more problems, resulting in other sets of 144 HAMLET generated rules, HAMLET-400, and 12 human refined rules, HAMLET-400-HUMAN. Also, the HAMLET-200-HUMAN and HAMLET-400-HUMAN versions were seeded into EVOCK, generating 4 HAMLET-200-HUMAN-EVOCK control rules and 4 HAMLET-400-HUMAN-EVOCK. Finally, the human refined again the control rules generated by GP in both configurations, leading to 3 HAMLET-200-HUMAN-EVOCK-HUMAN control rules and 4 HAMLET-400-HUMAN-EVOCK-HUMAN control rules (actually, in the later case, the human judged EVOCK control rules correct). We show the results in the next section.

Second Results

In Tables 2 and 3, we show the results using the above explained configurations. Table 2 compares PRODIGY4.0 in turn against each HAMLET configuration. Table 3 compares each HAMLET configuration against its next configuration. For instance, HAMLET-200-HUMAN is compared against HAMLET-200-HUMAN-EVOCK. A special case is HAMLET-200 which has two successors: HAMLET-200-HUMAN and HAMLET-400.

We used the same 80 test problems in (Hasslum & Geffner 2001) of 3 to 4 cities, and 4 to 5 packages. Also, we generated 40 more complex problems with the same random generator (the one built by Patrick Hasslum). These new problems had 10 and 20 goals, and 10 and 20 packages. We gave 100 seconds as time bound to all problems.

The columns on time, solution length, and solution quality (temporal measure) are measured by using only the problems solved by the corresponding configuration and PRODIGY4.0. This is why we have different figures for PRODIGY4.0 results in these three columns.

We also compare against the TP4 configuration of HSP*. In this case, we do not print the quality of solutions of both configurations, given that HSP* measures it as the makespan, while we measure it as the sum of the costs (in time) of all operators in the solution. This is so because PRODIGY4.0 generates total order solutions, while HSP* generates partial ordered solutions.

Table 1: Results for several configurations in the AIPS'00 logistics domain.

Planning System	**Solved problems**	**Time**	**Solution quality**	**Number of rules**
PRODIGY4.0	0	3600	-	-
HAMLET	0	3600	-	32
HAMLET-HUMAN	36	268.12	2604	39
HAMLET-HUMAN-EVOCK-1	36	84.27	3111	18
HAMLET-HUMAN-EVOCK-2	36	255.41	2585	40
HAMLET-HUMAN-EVOCK-3	36	79.79	3051	18
HAMLET-HUMAN-EVOCK-HUMAN	36	74.39	3051	18
TALPLANNER	36	9.53	2329	
PBR	29	1089.58	2409	
SHOP	36	14.93	2236	
R	36	39.22	4133	

Related Work

There has been relatively very little work in the field of learning for obtaining good quality solutions as an explicit goal of the learning system. Moreover, very few have concentrated on the interaction between a human and a machine learning system for acquiring control knowledge for quality. However, there is an increasing interest now through the use of resources and time by the planners (Hasslum & Geffner 2001; Nareyek 2001; Ghallab & Laruelle 1994). Also, there is a strong relation to the whole field of scheduling, given that schedulers handle time and resources (Smith, Frank, & Jonsson 2000).

An earlier system, QUALITY system (Pérez & Carbonell 1994) also used the PRODIGY4.0 planner. While QPRODIGY defines a cost function for each operator, QUALITY needs as input an evaluation function of a whole plan. With respect to learning, QUALITY compares the search trees produced by two different solutions to the same problem (one possibly generated by a human)[3] in order to learn control knowledge to *prefer* one solution to another. The problem with using *preference* control knowledge instead of using *selection* control knowledge (as is the case of HAMLET) is that when the planner has to backtrack, the alternatives are still there, and search can be much less efficient. An advantage of QUALITY is that it is able to transform the knowledge described in the evaluation functions into operative knowledge in terms of control knowledge. Another difference is that QUALITY does not refine the rules once they have been learned.

Other approaches define plan quality in terms of quality goals (Iwamoto 1994), carry out a rewriting process for optimizing the plan (Ambite & Knoblock 1998), or perform quality-based planning, without using learning as in PYRRHUS (Williamson & Hanks 1996). Within learning systems, others do not have the specific goal of improving solution quality; they obtain good solutions when they learn search control knowledge as a side effect. An example of such work are the first versions of the SCOPE system (Estlin & Mooney 1996), that uses a variation of FOIL (Quinlan 1990) to learn control knowledge for UCPOP (Penberthy & Weld 1992). They bounded the set of conditions to add to a control rule by using the information from the search trees. A newer version of this system was also able to obtain good solutions by learning, but they used only the "solution length" as their quality metric (Estlin & Mooney 1997).

[3]Therefore, it is difficult to compare its performance against HAMLET.

Others employ a different strategy for learning, such as STEPPINGSTONE (Ruby & Kibler 1992), that learns cases for achieving good solutions, or reinforcement learning systems that acquire numerical information about the expected values of applying actions to states (Watkins & Dayan 1992). Reinforcement handles planning in a different way, since usually there is no explicit/declarative representation of operators. Learning relates to modifying numerical quantities associated to expected values of applying actions to states.

There has also been some work on planning using predefined notions of quality, such as PYRRHUS (Williamson & Hanks 1996), where optimal solutions were found by a version of the branch-and-bound technique, but there was no learning involved.

The work reported in (Ambite & Knoblock 1998) describes the planning by rewriting approach that allows to optimize solutions after a basic planning process has ended. In this case, instead of learning control knowledge, they allow the user to specify a set of rewriting rules for optimizing a generated plan. But, there is no learning, so it would be equivalent to allowing the user to define his/her own control rules.

Conclusions

We can draw the following conclusions from the analysis of the results:

- The combination of automatic and manual generation of control knowledge is an appropriate approach, that is better than most hand-tailored current systems (only TALPLANNER and SHOP are better than our approach under both time and quality measures). To perform a fairer comparison, we would have to run all systems in the same machine. In the case of PBR, this system focuses on the quality of the solutions, so it is not strange that it is better

Table 2: Comparison between PRODIGY4.0 and each HAMLET configuration in the temporal logistics domain.

Number of problems	Solved problems	Time	Solution length	Solution quality	Solved problems	Time	Solution length	Solution quality
	PRODIGY				HAMLET-200			
80	71	8.9	943	4261.0	35	668.3	881	4117.0
120	103	8.9	943	4261.0	35	668.3	881	4117.0
	PRODIGY				HAMLET-200-HUMAN			
80	71	15.3	2068	9102.0	68	11.9	1623	7081.0
120	103	416.1	6577	28318.0	99	205.3	4459	19079.0
	PRODIGY				HAMLET-200-HUMAN-EVOCK			
80	71	13.9	2150	9470.0	71	10.0	1637	7013.0
120	103	385.1	6765	29133.0	103	146.5	3981	17171.0
	PRODIGY				HAMLET-200-HUMAN-EVOCK-HUMAN			
80	71	14.0	2150	9470.0	71	10.1	1637	7013.0
120	103	407.1	6765	29133.0	103	127.7	3981	17171.0
	PRODIGY				HAMLET-400			
80	71	10.5	1696	7514.0	57	655.9	1340	6826.0
120	103	10.5	1696	7514.0	57	655.9	1340	6826.0
	PRODIGY				HAMLET-400-HUMAN			
80	71	19.0	2130	9384.0	70	15.7	1416	4656.0
120	103	483.4	6745	29047.0	102	152.1	3531	11350.0
	PRODIGY				HAMLET-400-HUMAN-EVOCK			
80	71	15.0	2150	9470.0	71	6.8	1150	6039.0
120	103	363.5	6765	29133.0	103	99.7	2784	14777.0

than ours with this respect. However, it could not solve seven problems in 100 seconds (though it could solve them when time bound was set to 1000 seconds). With respect to the R system, it is faster than our solutions, but the quality is worse. Other systems that competed in the same track are not as good as any of these reported systems.

- There are domains in which machine learning alone is not enough unless we dedicate a long time to this activity. In most domains, this is a feasible solution given that once the control knowledge has been correctly acquired, it will be used for many problems solutions. When this is not feasible, an alternative consists on defining a "simpler" domain representation, learn from that domain, and then translate to the more difficult version the learned knowledge. Since we will never be sure that the learning system has captured all required knowledge, we can complement this automatic step with a manual adjustment of the knowledge. This process can be repeated until we are satisfied with the result.
- EVOCK can be effectively used to compact the control knowledge (reductions to half the number of rules are frequent) and make it faster, while preserving and improving some of the properties of the initial configurations (seeds).
- Another aspect relates to the fact that all HAMLET-HUMAN* configurations always go directly to the first solution (without any backtracking) in the first experiment.
- In the case of the temporal version, we can observe that plain HAMLET (HAMLET-200 and HAMLET-400) is able to improve the quality and solution length of the solutions. However, the number of solved problems is much less than the one by PRODIGY4.0. This is usually caused by a few rules (in the extreme only one) that prune the

Table 3: Comparison between each HAMLET configuration in the temporal logistics domain.

Number of problems	Solved problems	Time	Solution length	Solution quality	Solved problems	Time	Solution length	Solution quality
		PRODIGY				HAMLET-200		
80	71	8.9	943	4261.0	35	668.3	881	4117.0
120	103	8.9	943	4261.0	35	668.3	881	4117.0
		HAMLET-200				HAMLET-200-HUMAN		
80	35	666.9	831	3893.0	68	5.5	711	3136.0
120	35	666.9	831	3893.0	99	5.5	711	3136.0
		HAMLET-200-HUMAN				HAMLET-200-HUMAN-EVOCK		
80	68	11.9	1623	7081.0	71	9.6	1572	6738.0
120	99	205.3	4459	19079.0	103	145.2	3865	16679.0
		HAMLET-200-HUMAN-EVOCK				HAMLET-200-HUMAN-EVOCK-HUMAN		
80	71	10.0	1637	7013.0	71	10.1	1637	7013.0
120	103	146.5	3981	17171.0	103	127.7	3981	17171.0
		HAMLET-200				HAMLET-400		
80	35	666.7	834	3906.0	57	245.5	710	3667.0
120	35	666.7	834	3906.0	57	245.5	710	3667.0
		HAMLET-400				HAMLET-400-HUMAN		
80	57	652.2	1325	6750.0	70	13.8	1112	3759.0
120	57	652.2	1325	6750.0	102	13.8	1112	3759.0
		HAMLET-400-HUMAN				HAMLET-400-HUMAN-EVOCK		
80	70	15.7	1416	4656.0	71	6.7	1137	5985.0
120	102	152.1	3531	11350.0	103	99.7	2771	14723.0
		HAMLET-200-HUMAN-EVOCK-HUMAN				HSP*		
80	71	4.0	729	-	37	526	979	-
120	103	4.0	729	-	37	526	979	-
		HAMLET-400-HUMAN				HSP*		
80	70	3.4	606	-	37	499	945	-
120	102	3.4	606	-	37	499	945	-
		HAMLET-400-HUMAN-EVOCK				HSP*		
80	71	3.1	514	-	37	526	979	-
120	103	3.1	514	-	37	526	979	-

solutions paths from the search tree. By human intervention, we can achieve a similar (or better) number of solved problems to that of PRODIGY. It would have been difficult for a person to program all the rules that achieve this effect, but with the help of HAMLET, this process is made much easier. Also, the comparison with HSP* is very positive for our system.

In the current approach, the human in the loop must know something about how the planner works. In the future, we would like to evaluate how much knowledge the human actually needs and how much effort novice PRODIGY4.0users must undergo to generate and correct control rules. It would also be interesting to characterize the domains in which the human is needed, as opposed to simpler domains where a machine learning algorithm is enough.

Acknowledgements

This work was partially supported by a grant from the Ministerio de Ciencia y Tecnología through project TAP1999-0535-C02-02. The authors would also like to thank Patrick Hasslum for his help on using HSP*.

References

Aler, R.; Borrajo, D.; and Isasi, P. 1998. Genetic programming and deductive-inductive learning: A multistrategy approach. In Shavlik, J., ed., *Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98*, 10–18.

Aler, R.; Borrajo, D.; and Isasi, P. 2000. Knowledge representation issues in control knowledge learning. In Langley, P., ed., *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, 1–8.

Ambite, J. L., and Knoblock, C. A. 1998. Flexible and scalable query planning in distributed and heterogeneous environments. In Simmons, R.; Veloso, M.; and Smith, S., eds., *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, 3–10. Pittsburgh, PA: AAAI Press.

Ambite, J. L., and Knoblock, C. A. 2001. Planning by rewriting. *Journal of Artificial Intelligence Research* 15:207–261.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*.

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.

Borrajo, D.; Vegas, S.; and Veloso, M. 2001. Quality-based learning for planning. In *Working notes of the IJCAI'01 Workshop on Planning with Resources*, 9–17. Seattle, WA (USA): IJCAI Press.

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.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *Proceedings of the 2nd International Conference on AI Planning Systems*.

Hasslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In Cesta, A., and Borrajo, D., eds., *Preprints of the Sixth European Conference on Planning (ECP'01)*, 121–132.

Iwamoto, M. 1994. A planner with quality goal and its speed-up learning for optimization problem. In Hammond, K., ed., *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS94)*, 281–286.

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

Lin, F. 2000. From causal theories to successor state axioms and STRIPS-like systems. In *Proceedings of the AAAI-2000*.

Nareyek, A., ed. 2001. *Working notes of the IJCAI'01 Workshop on Planning with Resources*. Seattle, WA (USA): IJCAI Press.

Nau, D.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the IJCAI-99*, 968–973.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, 103–114.

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*, 323–328. Chicago, IL: AAAI Press, CA.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239–266.

Rodríguez-Moreno, M. D.; Borrajo, D.; and Meziat, D. 2001. An artificial intelligence planner for satellites operations. In *ESA Workshop on On-Board Autonomy*, 233–240. AG Noordwijk (The Netherlands): European Space Agency. Accepted.

Ruby, D., and Kibler, D. 1992. Learning episodes for optimization. In *Proceedings of the Ninth International Conference on Machine Learning*, 379–384. Aberdeen, Scotland: Morgan Kaufmann.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

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.

Watkins, C. J. C. H., and Dayan, P. 1992. Technical note: Q-learning. *Machine Learning* 8(3/4):279–292.

Williamson, M., and Hanks, S. 1996. Flaw selection strategies for value-directed planning. In Drabble, B., ed., *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS96)*, 237–244.