The Knowledge Engineering Review, Vol. 00:0, 1–24. © 2004, Cambridge University Press
DOI: 10.1017/S000000000000000 Printed in the United Kingdom

PLTOOL. A Knowledge Engineering Tool for Planning and Learning

Susana Fernández[1], Daniel Borrajo[1], Raquel Fuentetaja[1], Juan D. Arias[1] and Manuela Veloso[2]

[1]*Departamento de Informática, Universidad Carlos III de Madrid*
Avda. de la Universidad, 30. 28911 Leganés (Madrid). Spain
E-mail: dborrajo@ia.uc3m.es, {sfarregu,rfuentet,jdarias}@inf.uc3m.es
[2] *Computer Science Departament, Carnegie Mellon University*
Pittsburgh PA 15213-3890, USA
E-mail: veloso@cs.cmu.edu

Abstract

AI planning solves the problem of generating a correct and efficient ordered set of instantiated activities, from a knowledge base of generic actions, which when executed will transform some initial state into some desirable end-state. There is a long tradition of work in AI for developing planners which make use of heuristics which are shown to improve their performance in many real world and artificial domains. The developers of planners have chosen between two extremes when defining those heuristics. The domain-independent planners use domain-independent heuristics, which exploit information only from the "syntactic" structure of the problem space and of the search tree. Therefore, they do not need any "semantic" information from a given domain in order to guide the search. From a Knowledge Engineering (KE) perspective, the planners that use this type of heuristics have the advantage that the users of this technology need only focus on defining the domain theory and not on defining how to make the planner efficient (how to obtain "good" solutions with the minimal computational resources).

On the other hand, the domain-dependent planners require users to manually represent knowledge not only about the domain theory, but also about how to make the planner efficient. This approach has the advantage of using either better domain-theory formulations or using domain knowledge for defining the heuristics, thus potentially making them more efficient. However, the efficiency of these domain-dependent planners strongly relies on the KE and planning expertise of the user. When the user is an expert on these two types of knowledge, domain-dependent planners clearly outperform domain-independent planners in terms of number of solved problems and quality of solutions.

Machine-learning (ML) techniques applied to solve the planning problems have focused on providing middle-ground solutions as compared to the aforementioned two extremes. Here, the user first defines a domain theory, and then executes the ML techniques that automatically modify or generate new knowledge with respect to both the domain theory and the heuristics. In this paper, we present our work on building a tool, PLTOOL, to help users interact with a set of machine-learning techniques, and planners. The goal is to provide a KE framework for mixed-initiative generation of efficient and good planning knowledge.

1 Introduction

Planning is a problem solving task that consists of a given domain theory (a set of states and operators) and a problem (an initial state and a set of goals) to obtain a plan (a set of operators and a partial order of execution among them) such that when executed this plan transforms the initial state into a state where all the goals are achieved. Planning has been shown to be a difficult computational task (Bäckström 1992), whose complexity increases if we also consider trying to obtain "good" quality solutions (according to a given user-defined quality metric). In fact, it is PSPACE-complete (Bylander 1994). Therefore, redefining the domain theory and/or defining heuristics for planning is necessary if we want to obtain solutions to real world problems.

Planning technology has experienced a big advance in the last decade. New planning algorithms and techniques have been developed, including new totally-ordered and partially-ordered planners, planning based on planning graphs, planning based on SAT resolution, heuristic search planners, HTN planning (Hierarchical Task Networks) and planning with uncertainty and with time and resources (Ghallab, et al. 2004). The development of GRAPHPLAN (Blum & Furst 1995) and the definition of domain-independent heuristics based on relaxed domains as in HSP (Heuristic Search Planner) (Bonet & Geffner 2001) or FF (Fast Forward) (Hoffmann & Nebel 2001), has had a great impact in the field. But, even if these heuristics are very informative, and efficiently computed, they fail in many problems and domains. This can be especially noted when trying to obtain *good* plans according to different quality metrics (e.g. execution time, solution cost, and resource usage among others).

There is a parallel approach which aims to develop planners that require a domain-dependent manual definition. In contrast to fully domain-independent planners, which compute the heuristics they use automatically, this second type of planners rely on the manual design of their heuristics. Nevertheless, the planner engine usually is domain-independent. The manual design of heuristics is a time-consuming and error-prone process that requires good enough expertise by the user about the domain, the planning techniques and the specific planner employed. One representative example of languages used by these planners is temporal logic approaches, such as TLPLAN (Bacchus & Kabanza 2000). Heuristics in TLPLAN are expressed in a form of temporal logic, and introduced in the domain-theory formulation. Another example of such planners are HTN planners like SHOP2 (Nau, et al. 2003) and O-PLAN (Currie & Tate 1991). They allow defining the domain theory using a hierarchical decomposition of tasks. This kind of representation of tasks is quite adequate for many real world problems. However, defining hierarchical domain models usually requires more effort than defining the corresponding "flat" models, given that they also incorporate some kind of heuristic knowledge.

As discussed in more detail in (McCluskey, et al. 2003a), one of the main drawbacks of using current planning techniques in real world applications is the lack of good KE tools. There has been little work on helping the user define domain models. Examples of previous work in this area can be found in (Currie & Tate 1991, Myers & Wilkins 1997, Joseph 1989). Recently, there has been some interesting work on building such tools, as shown by the GIPO tool (McCluskey, et al. 2003b). GIPO allows a user to define a model using several types of representations of the domain theory: classical, hierarchical, with durative actions, or state-machines representations. It also includes some learning modules, as one that induces operators models.

In this paper, we propose more exploration in the use of ML techniques to help in the knowledge acquisition process. Following this idea, we present here a tool, PLTOOL (Planning and Learning Tool), able to automatically extract patterns of good behaviour from previous problem-solving episodes, present them to the user in such a way that s/he is not only able to understand the generated knowledge, but also is able to evaluate its validity. Then, the user can either accept the knowledge as it is to be incorporated into the domain-theory, modify it appropriately, or delete it. One of the advantages of this approach over the domain-independent planners is that it enables a user to define the heuristics semi-automatically by using domain-dependent knowledge. We could, in fact, use as a starting point a planner that already incorporates domain-independent heuristics with the goal of improving these heuristics. Also, it has the advantage over the domain-dependent planners in that knowledge is semi-automatically acquired, removing part of the KE burden from the user. In particular, the manual definition of heuristics is usually a difficult task which requires a lot of expertise of the user and can be alleviated by using the proposed tool.

In the past, the closest type of tools to the one we propose here used machine-learning technology without a mixed-initiative approach. They range from macro-operators acquisition (Fikes, et al. 1972, Korf 1985), case-based reasoning (Veloso 1994, Kambhampati 1989), rewrite rules acquisition (Ambite, et al. 2000, Upal & Elio 2000), generalized policies (Khardon 1999), deductive approaches of heuristics learning (EBL) (Minton 1988, Qu & Kambhampati 1995, Kambhampati 2000), learning domain models (Wang 1994, Yang, et al. 2005), to inductive approaches (ILP based) (Borrajo & Veloso 1997, Estlin & Mooney 1997, Aler, et al. 2002, Huang, et al. 2000). The reader can see (Zimmerman & Kambhampati 2003) for a general overview of the field. A classification of these techniques separates these techniques into the ones whose goal is acquiring a domain model (theory), and the ones that acquire heuristics (usually, in planning, heuristics are also named control knowledge). So, for instance, macro-operators modify the domain theory including new operators, while EBL or ILP techniques usually generate control knowledge.

So far, all these learning systems in planning are executable code lacking any type of user interaction. However, we believe that a better KE solution for the deployment of this technology in real world applications consists of a mixed-initiative approach to acquire domain and control knowledge (Aler & Borrajo 2002, Cortellesa & Cesta 2006). More specifically, we present here a tool, PLTOOL, that is able to plan, and then learn macro-operators and heuristics, in the form of control rules. The user can interact with PLTOOL at any stage of planning to explore the search tree, modify the learned knowledge, or add new knowledge. The learning of control rules is based on two approaches: a purely deductive approach, like EBL (Minton 1988, Mitchell, et al. 1986), and an inductive-deductive approach, based on HAMLET (Borrajo & Veloso 1997). Macro-operators learning consists of using a straightforward implementation of STRIPS macro-operators (Fikes et al. 1972) with some variation.

This tool works on top of the IPSS integrated planner and scheduler (Rodríguez-Moreno, et al. 2004c), which is based on PRODIGY (Veloso, et al. 1995) and QPRODIGY (Borrajo, et al. 2001). The main goal of the PRODIGY project, which spans almost 20 years, has been the study of ML techniques on top of a problem solver. Over the years, many different learning techniques have been implemented within this architecture (see (Veloso et al. 1995) for a review). However, little has been done within the PRODIGY architecture, and also outside it in the rest of the learning approaches applied to planning, to provide tools (including user interfaces) that would integrate several learning mechanisms and that users could employ for generating domain theories and heuristics. Some exceptions exist, such as the work on integrating learning abstraction levels and EBL (Knoblock, et al. 1991) or the work on graphical acquisition of domain theories (Joseph 1989). AI planning and scheduling complement each other, and overlap to some extent, but tackle different problems. Scheduling is applied to the problem of efficiently, or sometimes optimally, allocating time and other resources to an already-known sequence of actions (plan). Scheduling can therefore be seen as selecting among the various action sequences implicit in a partial-order plan in order to find the one that meets efficiency or optimality conditions with respect to time constraints, and filling in all the resource details as to the point at which each action can be executed.

This paper intends first to review the past work on building some of the learning components of the PRODIGY architecture, as well as the current planner. Second, the goal is to describe PLTOOL, which provides an integrated tool for the user to perform mixed-initiative planning and learning. The best way of taking advantage of different learning and planning techniques is integrating all of them in the same tool with a graphical interface that gives the user an interactive monitoring system. The third goal is to provide some experiments and results that show the usefulness of automatically learning knowledge and also on using a mixed-initiative approach.

Section 2 provides an integrated view of PLTOOL. Then, we describe its components in the following sections. So, Section 3 describes the planners that we are currently using: IPSS and FF. Section 4 presents three current learning components of PLTOOL: HAMLET, EBL, and macro-operators. Section 5 shows an example of interaction between the user and PLTOOL. Section 6 describes some experimentation performed with the tool to analyze its adequacy. And, finally, in Section 7 we draw some conclusions and outline future work.

2 PLTOOL architecture

Our goal is to build an integrated environment for planning, that also provides tools for helping in the KE process, especially from a ML perspective. Figure 1 shows a high level view of the architecture of one such environment, PLTOOL. The architecture includes several planning modules, several ML modules, and a GUI. The user can run any planner, execute any learning technique, and access/modify the learned knowledge.

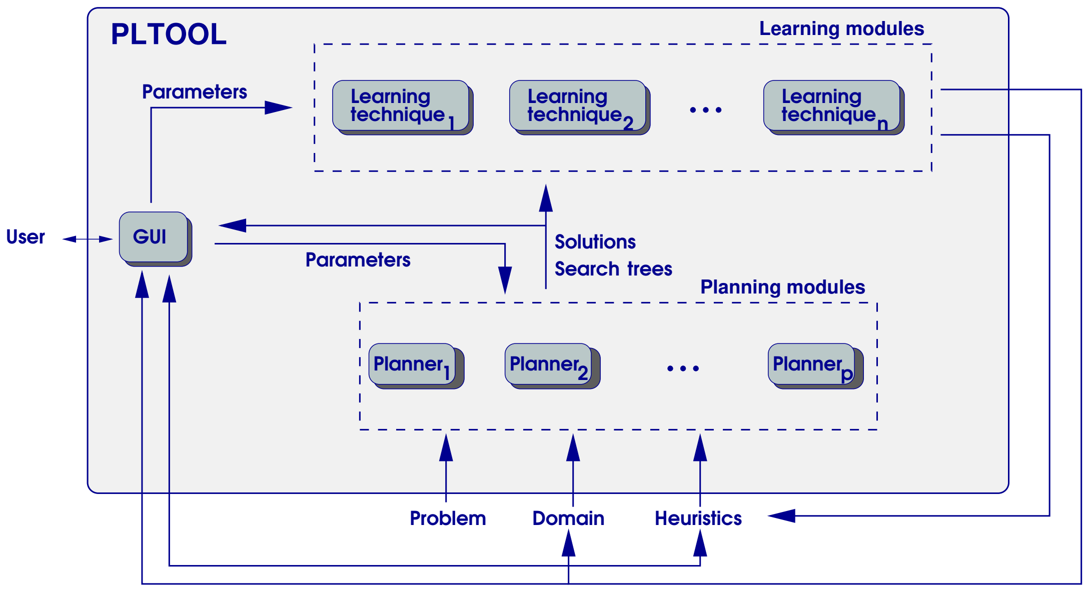


Figure 1 High level architecture of PLTOOL.

Currently PLTOOL is composed of five modules: two planner modules, IPSS and FF; three learning modules, HAMLET, EBL, and macro-operators; and one GUI. Figure 2 shows a high level view of the current status of PLTOOL. The IPSS planning module is explained in more detail in Section 3.1. The user can interact with the planner by providing IPSS its inputs and analysing its outputs. As described in the next section in more detail, the inputs are a domain description, a problem to be solved, a (possibly empty) set of heuristics (in the form of control rules), and a set of planning parameters. The output is a set of solutions for the problem (that includes some information on each solution and the performed search), and an explicit search tree. Currently, the other planning module, FF, is only used for some learning aspects related to the HAMLET learning module, that are explained in Section 4.1. Its inputs are a domain theory in PDDL, and a problem, and the output is a solution (we do not use its search tree for now).

In relation to the learning modules, the inputs to the HAMLET and EBL modules are a domain theory, a set of training problems, a search tree for each problem, and a set of parameters. The output is a set of heuristics in the form of control rules that can be refined by the user through the GUI (or by any other learning system, as described in (Fernández, et al. 2004)). The inputs to the macro-operators learning module are a domain, a solution, and some parameters. This module

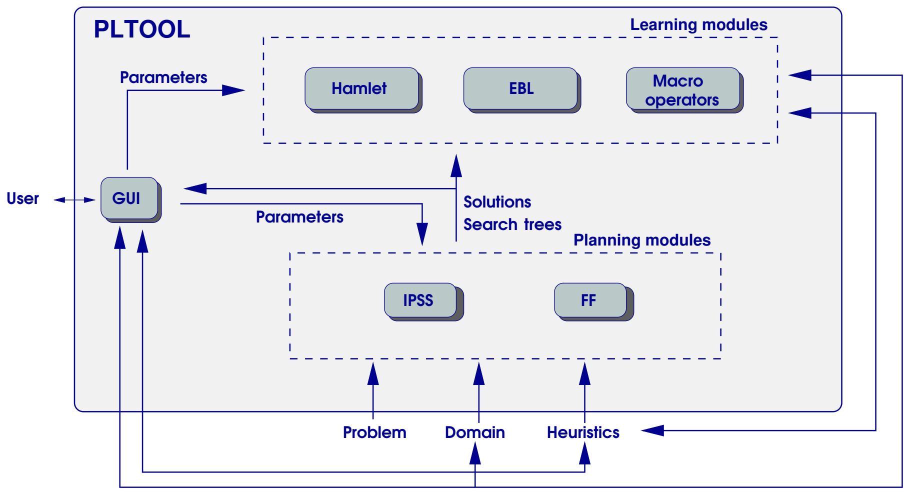


Figure 2 Current implementation of PLTOOL architecture.

generates as output a macro-operator each time it is called, that can also be further refined by the user through the GUI.

The GUI is written in CommonLisp using commonly used open source tools that help by defining and coding user interfaces, such as GLADE[1] and GTK[2].

The use of the tool starts with the user requirement of either using a planning module, or a learning module. If s/he wants to plan, then the user will specify the needed parameters (such as domain theory, problem to be solved, time bound, etc.) and the tool will call the appropriate planner to solve the problem. Then, PLTOOL can show the plan(s) and/or the search tree to the user for analysis. If the user wants to use the learning modules, s/he will also specify which learning technique to use, some parameters and PLTOOL will act accordingly as explained in later sections. In doing so, PLTOOL will use the planning modules to provide solutions to problems, and, therefore, be able to learn from them. Then, the user can inspect the learned knowledge, modify it and continue interacting with PLTOOL.

3 Planner modules of PLTOOL

So far, PLTOOL incorporates two planners, IPSS and FF, described in this section.

3.1 The IPSS planner

IPSS is an integrated tool for planning and scheduling (Rodríguez-Moreno et al. 2004c, Rodríguez-Moreno, et al. 2004b) that provides sound plans with temporal information (if run in integrated-scheduling mode). It uses QPRODIGY as the planner component, one version of the PRODIGY planner that is able to handle different quality metrics. The planner is integrated with a constraints-based scheduler (Cesta, et al. 2002) which reasons about time and resource constraints. We have not yet used the learning tools in combination with the scheduler, so we will focus now only on the planning component of IPSS. The planning component is a nonlinear planning system that follows a means-ends analysis (see (Veloso et al. 1995) for details on the planning algorithm). It performs a kind of bidirectional depth-first search (subgoaling from the goals, and executing operators from the initial state), combined with a branch-and-bound technique when dealing with quality metrics. The inputs to IPSS are the standard ones for planners (except for the heuristics, which are not usually given as explicit input):

[1]http://glade.gnome.org
[2]http://www.gtk.org

- domain theory: represented as a set of operators and a set of types;
- problem: represented as an initial state, a set of goals to be achieved, and a set of instances of the domain types;
- heuristics or control knowledge: a set of control rules that specify how to make decisions in the search tree; and
- parameters: such as time bound, node bound, depth bound, or how many solutions to generate.

The output of the planner alone is a valid total-ordered plan; a sequence of instantiated operators such that, when executed, transform the initial state of the problem into a state in which the goals are true (achieved). When using the complete IPSS, with its scheduling component, it returns a more general representation of a plan: a partially-ordered plan with temporal information (temporal intervals of valid execution of actions).

All knowledge is specified using the PDL language (PRODIGY DESCRIPTION LANGUAGE (Minton, et al. 1989, Carbonell, et al. 1992)) which is similar in terms of representation power to the ADL version of PDDL2.2 (including axioms, called inference rules in PDL).[3] This language provides some enhancements, such as: the possibility of calling LISP functions to specify values of operators, variables or different quality metrics for each operator; the definition of functions called handlers that are called by the planner every time a node is expanded; the possibility of having predicates with more than one numeric argument; or the declarative representation of heuristics. These enhancements provide a reasonable framework for building applications, given that one can plug in other computational tools that can work in parallel to the planner (Rodríguez-Moreno, et al. 2004a, Arias, et al. 2005). This can be achieved by calling other tools from either the operators preconditions, or the handlers.

From a perspective of heuristics acquisition, IPSS planning cycle, involves several decision points (all can lead to backtracking):

- *select a goal* from the set of pending goals; i.e. preconditions of previously selected operators that are not true in the current state (by default, it selects the order in which to work on goals according to their definition in the problem file);
- *choose an operator* to achieve the selected goal (by default, they are explored according to their position in the domain file);
- *choose the bindings* to instantiate the previously selected operator (by default, it selects first the bindings that lead to an instantiated operator that has more preconditions true in the current state);
- *apply* an instantiated operator whose preconditions are satisfied in the current state (default decision), or continue *subgoaling* on another pending goal.

Suppose that we are working on the well-known (in planning literature) logistics domain (Veloso 1994). In this simplification of a real world domain, packages have to be transported from their initial locations (either post offices or airports) to their final destination, by using either trucks (cannot move among cities) or airplanes (can only fly between city airports). Operators in the domain can load a package on a truck or an airplane, unload a package from a truck or airplane, and drive trucks or fly airplanes between two locations.

Figure 3 shows a control rule (heuristic) that determines when the `unload-airplane` operator has to be selected. It says that if the planner is trying to achieve the goal of having a package (`<package>`[4]) in an airport (`<location1>`) of another city (`<city1>`) where it currently is (`<city2>`), it should unload the package from an airplane instead of unloading it from a truck, given that trucks cannot move packages from one city to another one. This heuristic refers only

[3]We have defined translators working both directions, PDL to PDDL and vice-versa, covering almost all representation features.
[4]Variables appear between brackets.

to the decision on how to select an operator, and not on how to select an instantiation of the operator, given that these are two different kinds of decisions in IPSS.

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

Figure 3 Example of a hand-crafted control rule for selecting the `unload-airplane` operator in the logistics domain.

The conditions (left-part) of the control rules refer to facts that can be queried as to the meta-state of the search. These queries are called meta-predicates. PRODIGY already provides the most usual ones, such as knowing whether: some literal is true in the current state (`true-in-state`), some literal is a pending goal (`current-goal`) or some instance is of a given type (`type-of-object`). But, the language for representing heuristics also admits coding user-specific functions. As an example, we have defined the `different-vars-p` meta-predicate that checks whether all different variables of the rule are bound to different values.

The planner does not incorporate powerful domain-independent heuristics, given that the main goal of PRODIGY was not building the most efficient planner, but the study of ML techniques applied to problem solving. The idea was that the architecture would incrementally learn knowledge (such as heuristics, cases, domain models, . . .) through the interaction between the planner and the learning techniques, so that this knowledge would make PRODIGY efficient.

The reasons to choose IPSS are manifold. Among them, we can highlight the possibility of defining and handling different quality metrics (Borrajo et al. 2001), reasoning about multiple criteria, flexibility to easily define new behaviours (through handlers), capability to represent and reason about several numeric variables in the same predicate, definition of constraints on variable values in preconditions of operators, explicit definition of control knowledge as well as its automatic acquisition through different machine-learning techniques, and explicit rationale of each problem-solving episode through the search tree. This last feature is needed for building learning techniques, since we need to access decisions made and their rationale after a search has been performed.

3.2 The FF *planner*

The Fast Forward (FF) planner is a domain-independent heuristic planner developed by Jörg Hoffmann (Hoffmann & Nebel 2001). The inputs to FF are the standard ones for planners: a domain theory, a problem, and some parameters. The output, is a valid total-ordered plan; a sequence of instantiated operators. FF can handle classical STRIPS as well as ADL planning tasks, to be specified in PDDL2.2.

Its main heuristic principle was originally developed by Blai Bonet and Hector Geffner for the HSP (Heuristic Search Planner) system (Bonet & Geffner 2001). This principle consists of obtaining the heuristic estimate, by relaxing the planning task P into a simpler task P' by ignoring the delete lists of all operators. While HSP employs a technique that gives a rough estimate for the solution length of P', FF extracts an explicit solution to P' by using a GRAPHPLAN-style algorithm. The number of actions in the relaxed solution is used as a goal distance estimate.

These estimates control a local search strategy called Enforced Hill-Climbing (EHC); a hill-climbing procedure that, in each intermediate state, uses breadth-first search to find a strictly better, possibly indirect, successor.

The relaxed plans can also be used to prune the search space: usually, the actions that are really useful in a state are contained in the relaxed plan, so one can restrict the successors of any state to those produced by members of the respective relaxed solution. FF employs a slightly more elaborated form of this heuristic, called helpful actions pruning. If the local search fails, then FF switches to a best-first algorithm.

4 Learning modules of PLTOOL

Even if many learning techniques have been implemented within the PRODIGY architecture, we are only using three now for the current implementation of PLTOOL. Some of the techniques worked in the first linear version of PRODIGY, such as the pioneering work of Minton (Minton 1988) on EBL and macro-operator learning, so we have reimplemented some of these ideas on PLTOOL. Other ML techniques will be shortly implemented within PLTOOL, as ANALOGY (Veloso 1994). In this section, we will describe the three selected ML techniques we are currently using: HAMLET, EBL and macro-operators.

4.1 The HAMLET *module*

HAMLET is an incremental learning method based on EBL (Explanation Based Learning) and inductive refinement of control rules (Borrajo & Veloso 1997), that can also be seen as an ILP (*Inductive Logic Programming*) technique (Borrajo, et al. 1999). The inputs to HAMLET are a task domain ($\mathcal{D}$), a set of training problems ($\mathcal{P}$), a quality measure (Q) and other learning-related parameters. Internally, HAMLET calls IPSS and also receives as input the search tree expanded by the planner, in order to decide what to learn. HAMLET's output is a set of control rules ($\mathcal{C}$) that potentially guide the planner towards *good* quality solutions. HAMLET has two main modules: the Analytical learning (EBL) module, and the Inductive learning (ILP) module. Figure 4 shows HAMLET modules and their connection to the planner.

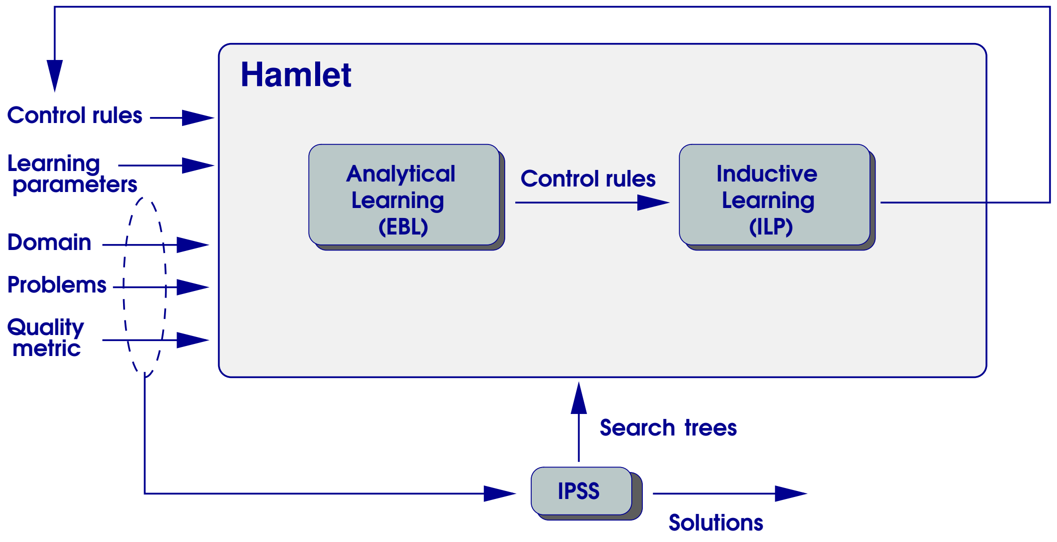


Figure 4 HAMLET high level architecture.

The Analytical learning module generates control rules from an IPSS search tree by finding positive examples of decisions; that is, decisions that lead to the best solution, instead of a worse solution, or a failure path. Once a positive example of a decision is found, a control rule is generated by performing an EBL step, modified to take into account the specifics of non-linear planning, such as considering the effect of the other pending goals. Therefore, HAMLET extracts the meta-state of the search, and performs a goal regression for finding which literals from the current 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 (Etzioni & Minton 1992) or overly general. Most work previous to HAMLET focused on this problem as a utility problem (learning many control rules degrades the performance of the planner when using them) (Minton 1988). However, HAMLET was one of the first ML techniques that focused it through an inductive solution. A similar approach was proposed in (McCluskey 1989). If rules were overly specific, HAMLET would solve more training problems and find more positive examples similar to the previously learned ones. Thus, HAMLET could generalize from the overly-specific examples. Similarly, if rules were overly general, when solving new problems, the control rules would fire when they should not, guiding IPSS towards failure paths or bad solutions. Then, HAMLET would generate negative examples of the control rules. HAMLET would then specialize the overly-general control rules, so that they do not longer cover the negative example. The generalization and specialization processes are performed by the Inductive learning module, that performs a kind of ILP process. 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 5 shows an example of a rule learned by HAMLET in the logistics domain for selecting the `unload-airplane` operator. As it is, the control rule says that if the goal is to have a package in a given location, `<location1>`, and the package is currently inside an airplane, then IPSS should use the `unload-airplane` instead of the `unload-truck` that also achieves the same goal (having a package in a given location). HAMLET learns this rule by first generating a rule that would require the airplane to be in another airport from <`location1`> (conditions computed from the goal regression), then learning another similar rule from a problem in which the airplane is at the same <`location1`>, and finally inducing a new more general rule from these other two (and removing the other two more specific rules).

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

Figure 5 Example of a control rule learned by HAMLET for selecting the `unload-airplane` operator in the logistics domain.

We have used the same deductive approach for automatically acquiring control rules for other types of planners, such as hybrid HTN-POP planners as HYBIS (Castillo, et al. 2001),[5] though we have not yet implemented its inductive counterpart (Fernández, et al. 2005). Within the general framework of learning heuristics for different types of planners, we are currently generating a common language for describing heuristics for different planning paradigms, independently of the planner, such as GRAPHPLAN or FF based planners. The main difficulty is that each planner generates a different type of search tree, and, therefore, decision points with different semantics. However, we have found that, in most cases, we have used the same language for capturing the important features of the meta-state of the search.

[5]This domain-independent planner has been used for industrial manufacturing domains.

Given that IPSS is not able to solve problems in some difficult domains, HAMLET can use FF (calling the FF planning module) to try to solve those problems (a kind of bootstrapping the learning process). The FF solution of a problem is automatically translated into an equivalent IPSS search tree, so that HAMLET can then learn from it. Figure 6 shows a scheme of the approach. FF is given the same unsolved planning problem and it generates a solution to the problem. Then, the problem consists of how the solution to a problem can help IPSS 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). It provides, basically, the order in 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), that select as valid search nodes, the ones that use any of the instantiated operators in the solution. Using this new set of rules, IPSS generates a solution path corresponding to the solution provided by the other planner. Afterwards, we let IPSS 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. The interested reader can find more details in (Fernández et al. 2004).

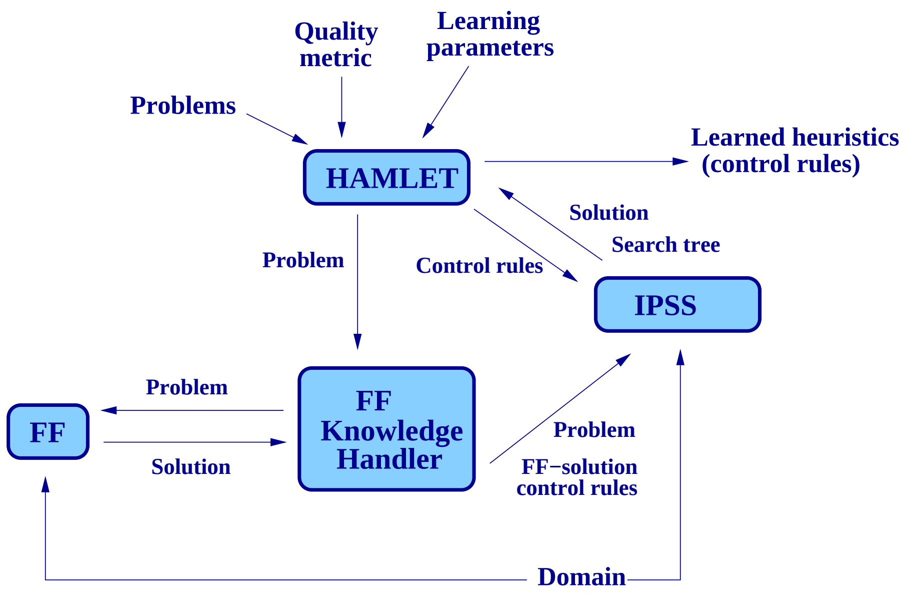


Figure 6 Providing prior knowledge to HAMLET by using another planner, FF.

HAMLET uses some parameters for which it provides reasonable default values (defined after the extensive experimentation performed in the past by the authors). A non-expert would use those default values, but advanced users might like to play with the parameters. Here we describe the most important ones:

- Quality metric: given that the user can define a set of specific quality metrics for each domain, HAMLET's input, Q, refers to the metric to be used in the search. Therefore, the learned heuristics will refer to that metric, and the heuristics are usually different if a different metric is chosen. The default quality metric is the number of operators in the plan (solution length), but the user can choose to learn heuristics to minimize execution time (makespan), economic cost of the planning operators in the plan or any other user defined criteria.

- Cost-bound: when HAMLET computes quality measures, the user can specify an initial bound for the branch-and-bound search of the planner. From an application perspective, users can then provide constraints on solutions.
- Initial set of heuristics: HAMLET can also start with an initial set of control rules that can be either learned by HAMLET or another ML technique, or defined by the user. Therefore, HAMLET can also be used for theory refinement. In (Fernández et al. 2004), we present some experiments that show that HAMLET results can be improved if we start the learning process from the output of another learning process (such as using a genetic programming approach (Aler et al. 2002)) or a previous manual generation of control rules.
- *eager-mode*: there are two dimensions related to this parameter. On the one hand, we can specify where HAMLET will learn from. It can be:

  – Lazy: does not learn control rules from decisions that were the default ones of the planner. So, if the first alternative that the planner tried was the best one, we do not learn anything. This assumes a deterministic planner, as IPSS is, given that the planner will also select that alternative first in the future.
  – Eager: HAMLET learns also from default decisions.

  On the other hand, we can control whether HAMLET would learn even if it did not expand the whole search tree, exploring all alternatives in all nodes. Given that positive examples are taken from the best solutions, we cannot be sure that an alternative is the best one for a node if we do not expand all alternatives. So, we can have two values:

  – Total: HAMLET requires the expansion of the whole search space including finding all possible solutions. This implies that training problems have to be simple, but informative enough.
  – Partial: IPSS expands the search space up to the time limit. When the domain is hard and IPSS cannot expand the whole search tree, this option makes learning lazier also.

  Then, we have four modes for the eager-mode parameter:

  – Lazy-total: learning from non-default alternatives only and need of expanding the whole search space (very lazy);
  – Lazy-partial: learning from non-default alternatives and search space can be partially explored (lazy);
  – Eager-partial: learning from default alternative also and search space can be partially explored (eager); and
  – Eager-total: learning also from default alternatives and need of expanding the whole search space (very eager).

  In (Borrajo & Veloso 1997), the reader can find more information on the impact of different levels of laziness in the learning process. The main conclusion is that the lazier the behaviour of HAMLET is, the better the results are obtained. Given that HAMLET learns fewer rules (only from non-default alternatives) and requires a lot of knowledge to learn (expansion of the whole search space), it is harder to find a negative example of the learned heuristics.
- *learn-time-bound*: maximum allowed time (in seconds) to solve each learning problem.
- *learning-type*: controls what kind of general learning scheme we are using. It can be:

  – `interactive`: it will ask for each problem to learn from, solve it, and learn. This process will be repeated until the user stops it.
  – `deduction`: only learning by deduction (Section 4.2).
  – `deduction-induction`: deduction followed by a fast induction. First, HAMLET makes a deduction on all the training set (it only generates EBL-type rules). Then, HAMLET tries to generalize the learned rules only using the ones learned (no specialisation process).
  – `dynamic`: for each problem in the training set, HAMLET generates rules applying EBL, and HAMLET tries to generalize or specialize them before solving the next problem.

- `ebl`: learning only by deduction, and also computing utility. Later, rules that do not have a higher utility than a given threshold are removed. This will be explained in more detail in Section 4.2.
- `active`: HAMLET performs a kind of active learning, that consists of the automatic generation of *good* problems for learning.

As any machine-learning inductive tool, HAMLET needs a set of training problems to be given as input. The standard solution from the ML perspective is that the user provides as input a set of relevant planning problems (similar to the ones that IPSS would need to solve in the future). However, in most real world domains, these problems are not available. So, an alternative solution consists of defining a generator of random problems. Then, one can assume (ML theory assumes) that if HAMLET sees enough random problems covering the space of problems reasonably well, and it learns from them, the learned knowledge will be reasonably adapted to the future. But, this solution requires building one such generator for each domain. Another solution consists of automating this task as described in (McCluskey & Porteous 1997). Finally, another alternative consists of using active learning schemes, so that HAMLET selects at each step what are the characteristics that the next learning problem should have in order to improve the learning process. Then, HAMLET can call a problem generator with these characteristics as input, so that the problem generator returns a problem that is expected to improve learning. Examples of these characteristics in the logistics domain might be the number of packages, the number of goals, or the number of cities. HAMLET has an active learning module that implements this solution, and can be used to improve the learning process.

Finally, another aspect that obviously influences the behaviour of HAMLET, as in any other computational system, is the representation of the domain theory. Very few people have studied the impact of domain theory representation on the learning processes in planning. In (Aler, et al. 2000), we described some initial experiments on this aspect that we would like to expand on in the future.

4.2 The EBL *module*

Given that HAMLET incorporates an EBL module without the utility computation, it required little programming effort to expand it with such a utility-based analysis. In order to run the EBL module, the user only has to provide the value `ebl` to the `learning-type` parameter. In this mode, HAMLET only uses its Analytical learning module with the value `eager-total` of the `eager-mode` parameter (EBL learns rules from all decisions, including the first alternative tried in each node). So, in the previous example of the logistics, EBL would learn and keep the two more specific rules, instead of performing the inductive (generalization) step. Then, EBL performs a utility analysis performed for each learned rule. According to the equation by Minton (Minton 1988), the utility of a rule r can be computed as:

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

where $u(r)$ is an estimation of the utility of r, $s(r)$ is an estimation of the search time that the rule saves when used, $p(r)$ is the estimation of the probability that the rule matches, and $m(r)$ is an estimation of the cost of using the rule (matching time). We estimate these functions as:

- $s(r)$: given that each rule is learned from a node in a search tree, we estimate $s(r)$ as the number of nodes below the node from where it learned the rule, multiplied by the time IPSS takes to expand a node. A better estimate can be computed by using a second training set of problems as Minton did, and computing the average number of nodes every time the rule was used.
- $p(r)$: we estimate it as the number of times that r actually fired in the training problems divided by the number of times that IPSS tried to use the rule. As before, a better estimate can be obtained by using 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 whose utility is less than a user-defined threshold can be removed. Other solutions for estimating the control rules utility can be found in previous work (Gratch & DeJong 1992). The EBL module provides an access to the needed information, so they can potentially be easily implemented here also.

4.3 Macro-operators learning

The tool also provides a utility for automatically building macro-operators that are incorporated into the domain theory. This provides some kind of abstraction that can be used by the user to refine the domain theory, creating more efficient versions of it, as has been shown by MACRO-FF (Botea, et al. 2005). Perhaps, the main drawback that learning macro-operators has is the utility problem (Minton 1988, McCluskey & Porteous 1997). One way of solving it is using a utility analysis: macro-operators are generated with a set of learning problems. Then, they are used in another set of learning problems to assess their utility and remove not useful macro-operators (using similar schemes as the one presented in the EBL section). Our implementation does not incorporate a utility analysis, but we already have some code for computing utility of control knowledge that can easily be reused for this purpose. Another way of solving the utility problem is by analyzing carefully which macro-operators are learned, as in MACRO-FF.

We propose here yet another way of solving the utility problem by integrating their generation with control-rule learning, as PLTOOL offers this possibility. Once macro-operators are generated, control rules can be learned that tell the planner when they are useful. Classically, as a heuristic, macro-operators are always selected as first alternatives given that, otherwise, the planner can always find a solution without them (by using base domain operators). In the IPSS planner, this solution implies placing them at the beginning of the domain file, which ensures that the macro-operator is going to be chosen as the first option. Nonetheless, this is not always the best choice. For example, there could be situations in which it is not necessary to execute the entire sequence of operators included in the macro-operator, and doing so will consume more resources than necessary or even avoid finding a solution (increasing search time and explored nodes).

Our solution is to include the macro-operator at the end of the domain file. But, when doing so, IPSS will never select it, because the base operators will solve the problem first without the need of macro-operators. Thus, the behaviour of the planner would be equal or worse than without the macro-operator. It would be worse in the case of backtracking, given that if IPSS does not solve the problem with the base operators, neither it will solve the problem with the macro-operators. Thus, we have to combine this with control rules that select the macro-operator only when appropriate. And these control rules can be automatically learned by HAMLET. Section 6.6 shows some experimental results of combining control-rule learning and macro-operators in the satellite domain.

5 Example of interaction

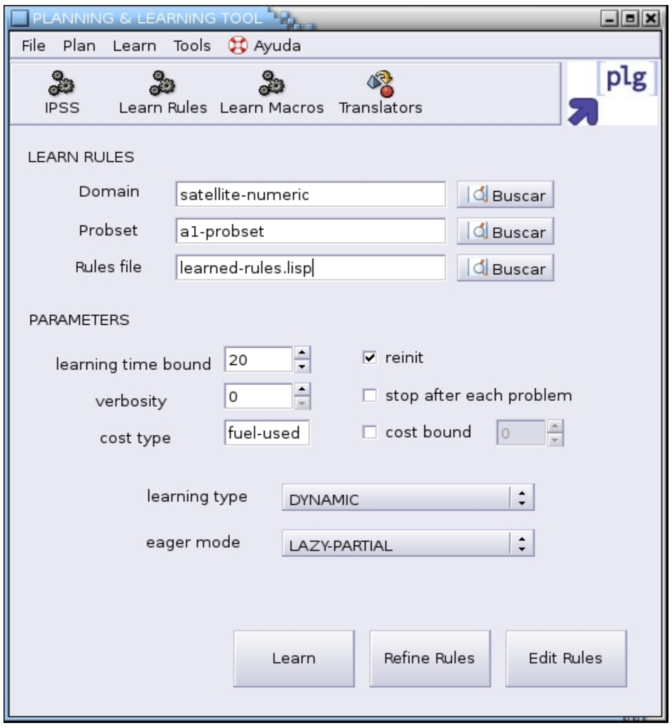
The following example illustrates how the user could interact with the PLTOOL to learn control rules in the numeric version of the satellite domain. In the satellite domain, satellites need to take images in different directions, in certain modes, using appropriate instruments. Each state of the domain is defined by some satellites, some directions, some instruments and a number of modes. It provides the following operators:

- `turn-to`: turns the satellite from one direction to another. The operator can be applied between any pair of directions.
- `switch-on`: switches on an instrument which is on board a satellite. This operator can only be applied if the satellite has power available. When an instrument is switched on, it is no longer calibrated, and the satellite has no more power available.

- `switch-off`: switches off an instrument on board a satellite, when the instrument is powered on. As a result, the satellite has power available, but the instrument is powered off.
- `calibrate`: calibrates an instrument on board a satellite in one direction. The satellite will be pointing to the direction, the direction must be a calibration target for the instrument and the instrument must have power. As an effect, the instrument is calibrated.
- `take-image`: takes an image with an instrument on board a satellite in one direction and a mode. The satellite must be pointing to the direction, and the instrument must be powered on, be calibrated and support the mode. Once the operator is applied, the image is taken in the specified direction and mode

Usually, the goals are to have images in a number of directions in a number of modes and to have some satellites pointing to specified directions. The initial states are such that each satellite has power available and no instrument is either powered on or calibrated. The numeric version of the domain introduces data capacities into the satellites (the operator `take-image` can be only applied when the satellite has enough data capacity) and fuel used in slewing among targets. Different instantiations of the operator `turn-to` consume different quantities of fuel that depend on the slew time from the initial direction to the final direction. The plans are expected to minimize fuel used when obtaining the goals.

The example uses the numeric version of the satellite domain. Thus, the smaller the quality metric (fuel-used), the better the plan. Therefore, the aim of the interaction between the user and the tool is to obtain a set of control rules to help the planner in generating "good" plans. Suppose that the user initially does not know how to generate control rules, which is quite common, given that defining control knowledge (heuristics) is usually harder than specifying domain knowledge. The user decides to see first what HAMLET learns from solving problems, so s/he invokes HAMLET and specifies the domain (`satellite-numeric`), the set of training problems (`a1-probset`), and the file to save the control rules (`learned-rules.lisp`). There are other parameters that can also be specified as Figure 7 shows.



LEARN RULES

Domain: satellite-numeric

Probset: a1-probset

Rules file: learned-rules.lisp

PARAMETERS

learning time bound: 20

verbosity: 0

cost type: fuel-used

[✓] reinit

[ ] stop after each problem

[ ] cost bound: 0

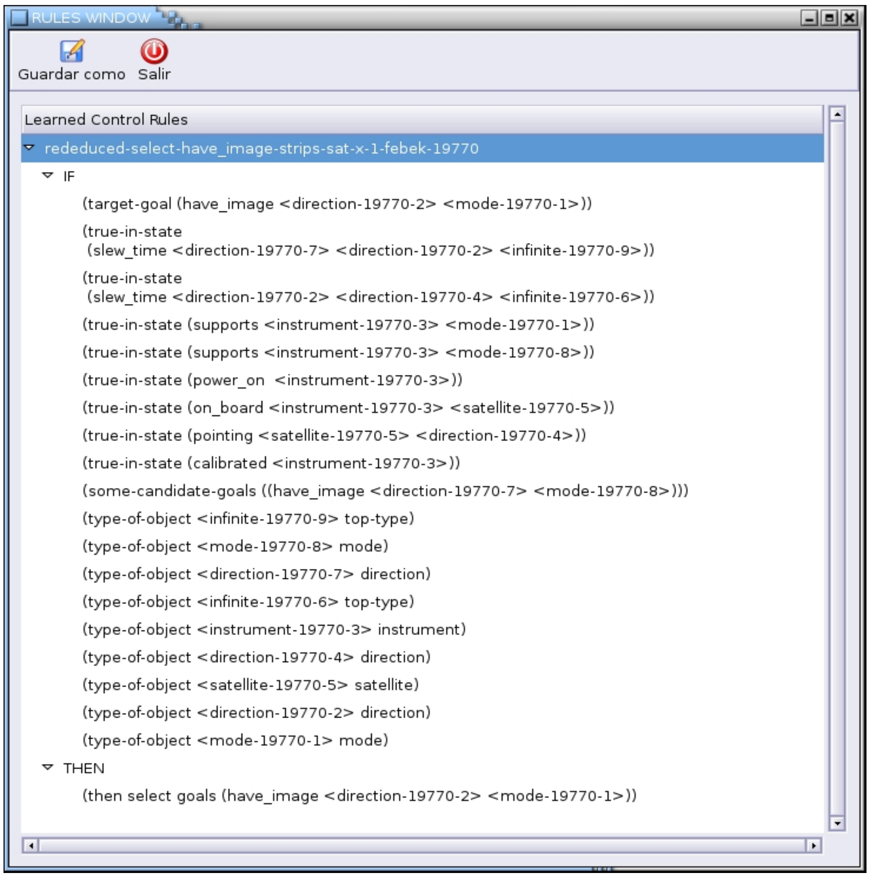
learning type: DYNAMIC

eager mode: LAZY-PARTIAL

Figure 7 Learning window in PLTOOL.

Given that the goal is to reduce the fuel used, the user must set up the `cost-type` parameter (quality metric) to the `fuel-used` value, so that IPSS will search for better solutions in terms of fuel used. In this example, the user has only changed the values of the `learning-time-bound` parameter to 20 seconds and the `eager-mode` parameter to `lazy-partial` because the user intuition (or previous experience) is that IPSS is not going to build the complete search tree for the problems in `a1-probset`. The user leaves the default values for the rest of parameters.

When the learning process finishes, the user can edit the learned rules and modify them if necessary. For example, one of the generated rules after the learning episode in Figure 7 is shown in Figure 8. This rule says that if given a state (specified by the `true-in-state` meta-predicates) in which one satellite is pointing to direction <`direction-19770-4`>, the planner has one goal to have an image at direction <`direction-19770-2`>, and there is another goal to have an image at direction <`direction-19770-7`> with a different mode, and two modes are supported by a calibrated on-board instrument in the satellite, then IPSS should first achieve the goal to have an image at <`direction-19770-2`>. The user can modify the rules, and in the future we plan to add a syntax-driven editor to help the user on the valid definition of control rules.



```
Learned Control Rules
▿ rededuced-select-have_image-strips-sat-x-1-febek-19770
  ▿ IF
      (target-goal (have_image <direction-19770-2> <mode-19770-1>))
      (true-in-state
       (slew_time <direction-19770-7> <direction-19770-2> <infinite-19770-9>))
      (true-in-state
       (slew_time <direction-19770-2> <direction-19770-4> <infinite-19770-6>))
      (true-in-state (supports <instrument-19770-3> <mode-19770-1>))
      (true-in-state (supports <instrument-19770-3> <mode-19770-8>))
      (true-in-state (power_on  <instrument-19770-3>))
      (true-in-state (on_board <instrument-19770-3> <satellite-19770-5>))
      (true-in-state (pointing <satellite-19770-5> <direction-19770-4>))
      (true-in-state (calibrated <instrument-19770-3>))
      (some-candidate-goals ((have_image <direction-19770-7> <mode-19770-8>)))
      (type-of-object <infinite-19770-9> top-type)
      (type-of-object <mode-19770-8> mode)
      (type-of-object <direction-19770-7> direction)
      (type-of-object <infinite-19770-6> top-type)
      (type-of-object <instrument-19770-3> instrument)
      (type-of-object <direction-19770-4> direction)
      (type-of-object <satellite-19770-5> satellite)
      (type-of-object <direction-19770-2> direction)
      (type-of-object <mode-19770-1> mode)
  ▿ THEN
      (then select goals (have_image <direction-19770-2> <mode-19770-1>))
```

Figure 8 Automatically learned control rules for the satellite domain.

Figure 9 shows a schematic view of a problem where the previous rule could be triggered in some search node. Initially there are three directions, gs-2 (a groundstation), phe-6 (a phenomenon) and star-5 (a star) and one satellite pointing to gs-2 (indicated by the dotted arrow). In addition, in this initial state of the problem, an instrument is on board the satellite, and the calibration target for the instrument is gs-2. The goals are to have an image of phe-6 and two images of star-5 with different modes (supported by the instrument).

When matching the control rule against the meta-state of the search, the matcher would generate the bindings: (`<direction-19770-4>` . gs-2), (`<direction-19770-2>` . phe-6), (`<direction-19770-7>` . star-5), (`<satellite-19770-5>` . satellite). Thus, this rule says that the planner should work first on the goal of having the phe-6 image.

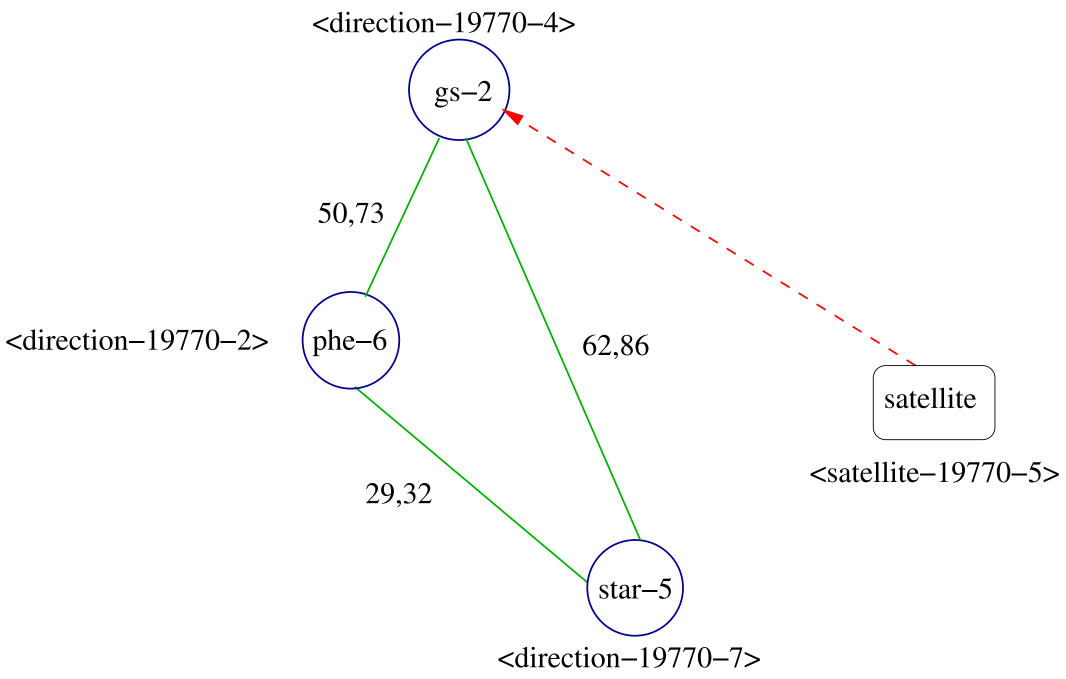
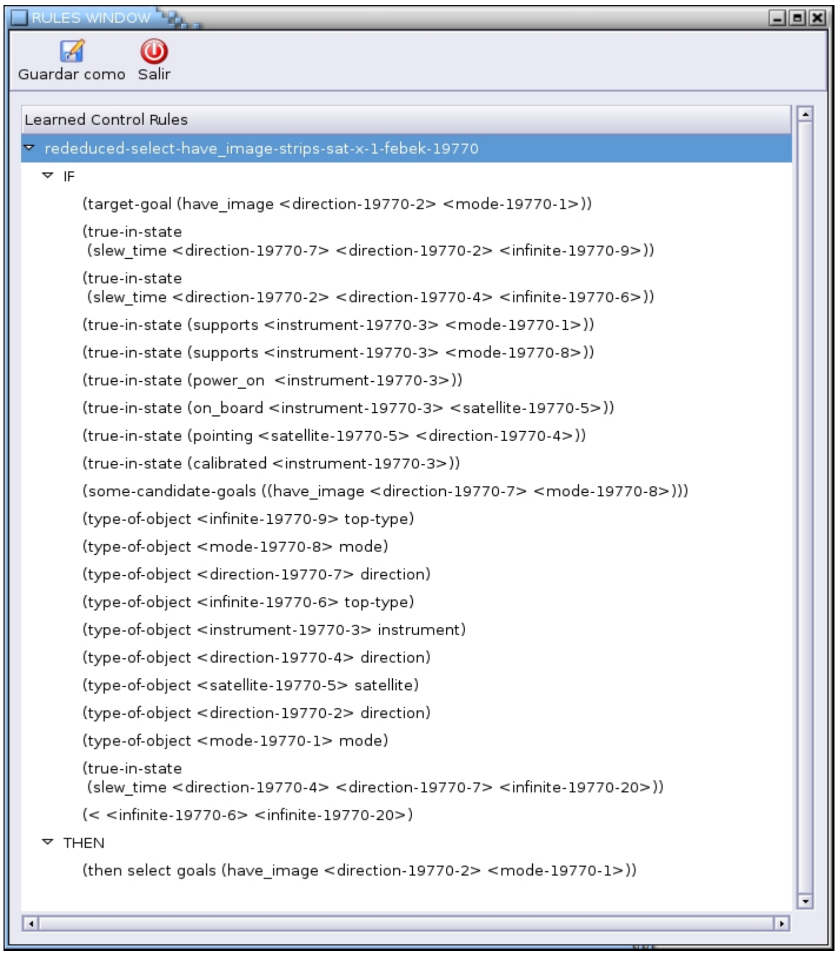


Figure 9 Slew times example in the satellite domain.

This rule is not correct because it does not express the real reason for choosing the goal of having the image at phe-6 first (in fact, this is a better choice because in this way the total fuel used is less): suppose that the numbers in Figure 9 represent the slew times the satellite needs to turn from an initial direction to a final direction. If we first turn the satellite from gs-2 to star-5 and then from star-5 to phe-6, the total fuel used is $62.86 + 29.32 = 92.18$. But, if we first turn the satellite from gs-2 to phe-6 and then from phe-6 to start-5, the fuel the satellite would spend is $50.73 + 29.32 = 80.05$. So it is better to turn to phe-6 first instead of to star-5 because the total cost in terms of fuel is less. This is the most important missed condition in the control rule. The learning engine of HAMLET does not yet have a mechanism to learn numerical conditions. In fact, there is not any ML technique in planning with the ability to learn numerical conditions. Nonetheless, the user can manually refine the rule including numerical conditions. In this case, if we translate the previous reasoning to variables representing directions, the user should add a condition expressing that the slew time to turn the satellite from `<direction-19770-4>` (gs-2) to `<direction-19770-2>` (phe-6) is smaller than the slew time to turn the satellite from `<direction-19770-4>` (gs-2) to `<direction-19770-7>` (star-5).

Figure 10 shows the previous control rule modified by the user using the GUI. It has only added the last two meta-predicates in the antecedent of the rule. In this rule, `<infinite-19770-6>` is a variable representing the slew time from `<direction-19770-4>` to `<direction-19770-2>` and `<infinite-19770-20>` is a variable representing the slew time from `<direction-19770-4>` to `<direction-19770-7>`. In the next section we show several experiments using this refined rule.

The new refined rule set can be used as input for a new learning process where HAMLET could generalize or specialize them according to the training set. Moreover, through PLTOOL the user can graphically explore the search tree expanded by the planner with and without the rules (see Figure 11). Then, the user can follow the planner's decisions to see if the generated



```
Learned Control Rules
rededuced-select-have_image-strips-sat-x-1-febek-19770
  IF
    (target-goal (have_image <direction-19770-2> <mode-19770-1>))
    (true-in-state
     (slew_time <direction-19770-7> <direction-19770-2> <infinite-19770-9>))
    (true-in-state
     (slew_time <direction-19770-2> <direction-19770-4> <infinite-19770-6>))
    (true-in-state (supports <instrument-19770-3> <mode-19770-1>))
    (true-in-state (supports <instrument-19770-3> <mode-19770-8>))
    (true-in-state (power_on  <instrument-19770-3>))
    (true-in-state (on_board <instrument-19770-3> <satellite-19770-5>))
    (true-in-state (pointing <satellite-19770-5> <direction-19770-4>))
    (true-in-state (calibrated <instrument-19770-3>))
    (some-candidate-goals ((have_image <direction-19770-7> <mode-19770-8>)))
    (type-of-object <infinite-19770-9> top-type)
    (type-of-object <mode-19770-8> mode)
    (type-of-object <direction-19770-7> direction)
    (type-of-object <infinite-19770-6> top-type)
    (type-of-object <instrument-19770-3> instrument)
    (type-of-object <direction-19770-4> direction)
    (type-of-object <satellite-19770-5> satellite)
    (type-of-object <direction-19770-2> direction)
    (type-of-object <mode-19770-1> mode)
    (true-in-state
     (slew_time <direction-19770-4> <direction-19770-7> <infinite-19770-20>))
    (< <infinite-19770-6> <infinite-19770-20>)
  THEN
    (then select goals (have_image <direction-19770-2> <mode-19770-1>))
```

Figure 10 An example of user refined rule for the satellite domain.

rules are correct (i.e. they have been applied correctly at the appropriate moment in the tree). Furthermore, the search tree view helps to detect from which decisions it would be appropriate to generate new rules.

In the future, we plan to endow the search tree view with the ability of automatically creating new rules from a user-chosen decision point.

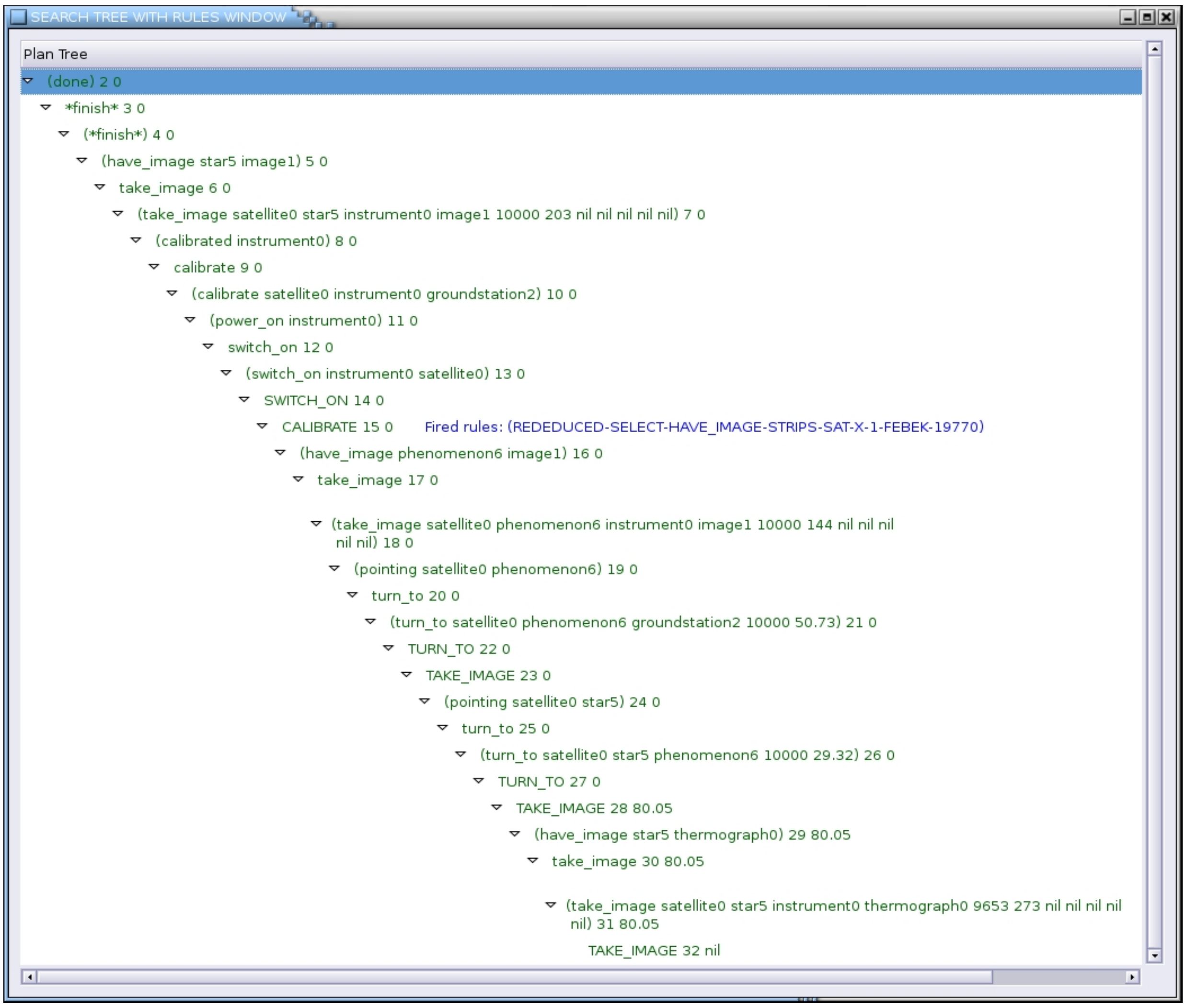
6 Some experimental results

In this section experiments are carried out to show that PLTOOL is able to generate control knowledge that improves future unseen planning problems and help users define useful control knowledge. In these experiments, we have used some benchmark domains from the common repository used in previous planning competitions.[6]

We have performed the following experiments which are described in the next subsections:

- Planning capabilities of the base planner
- Stand-alone machine learning
- Using FF to help HAMLET
- Mixed-initiative acquisition of control knowledge
- Using mixed-initiative learning for improving quality
- Combining macro-operators and mixed-initiative learning

[6]http://www.icaps-conference.org



SEARCH TREE WITH RULES WINDOW

Plan Tree

```
(done) 2 0
  *finish* 3 0
    (*finish*) 4 0
      (have_image star5 image1) 5 0
        take_image 6 0
          (take_image satellite0 star5 instrument0 image1 10000 203 nil nil nil nil) 7 0
            (calibrated instrument0) 8 0
              calibrate 9 0
                (calibrate satellite0 instrument0 groundstation2) 10 0
                  (power_on instrument0) 11 0
                    switch_on 12 0
                      (switch_on instrument0 satellite0) 13 0
                        SWITCH_ON 14 0
                          CALIBRATE 15 0     Fired rules: (REDEDUCED-SELECT-HAVE_IMAGE-STRIPS-SAT-X-1-FEBEK-19770)
                            (have_image phenomenon6 image1) 16 0
                              take_image 17 0
                                (take_image satellite0 phenomenon6 instrument0 image1 10000 144 nil nil nil
                                 nil nil) 18 0
                                  (pointing satellite0 phenomenon6) 19 0
                                    turn_to 20 0
                                      (turn_to satellite0 phenomenon6 groundstation2 10000 50.73) 21 0
                                        TURN_TO 22 0
                                          TAKE_IMAGE 23 0
                                            (pointing satellite0 star5) 24 0
                                              turn_to 25 0
                                                (turn_to satellite0 star5 phenomenon6 10000 29.32) 26 0
                                                  TURN_TO 27 0
                                                    TAKE_IMAGE 28 80.05
                                                      (have_image star5 thermograph0) 29 80.05
                                                        take_image 30 80.05
                                                          (take_image satellite0 star5 instrument0 thermograph0 9653 273 nil nil nil nil
                                                           nil) 31 80.05
                                                            TAKE_IMAGE 32 nil
```

Figure 11 Search tree view.

6.1 Planning capabilities of the base planner

As a starting point, we have evaluated the base planner of PLTOOL, IPSS, with some of the repository domains and problems. Table 1 shows the number of solved problems (*Solved*) and the average of solved problems (*Average*) with respect to all problems (*Total*) for each domain without using control knowledge. The time limit used was 15 seconds.[7]

As expected, given that the current planner we are using within the architecture is not a heuristic planner, the search becomes intractable for most problems. An interesting result is that there is a big difference in behaviour between the Zenotravel domain and its Zenotime version. The Zenotravel domain describes fuel consumption using predicates, so the planner generates a lot more instantiations, because the planner needs to instantiate the fuel variable of the operators with all possible existing levels of fuel. However, when using the Zenotime version of the domain, fuel is handled as a numeric variable. Given that IPSS handles this type of variables, IPSS can only instantiate the fuel level to its current value. Therefore, IPSS does not branch so much, and

[7]We have verified experimentally that increasing this time does not contribute to any significant improvement. For problems of this size if a solution is not reached within 15 seconds, the probability of finding a solution with a time limit one order of magnitude larger is small.

Table 1 Solved problems for each domain in the common repository.

Domain	Total	Solved	Average
Depots	22	0	0%
Driverlog	20	0	0%
Logistics	28	0	0%
Zenotravel	20	2	10%
Zenotime	20	9	45%
Miconic	150	15	10%
Rover	20	5	25%
Blocks	35	10	28%
Satellite	20	20	100%

solves more problems. Something similar happens in the Satellite domain. We believe this is a good quality of IPSS given that most real world domains require this type of reasoning.

6.2 Stand-alone machine learning

As a second set of experiments, we have evaluated PLTOOL fully-automated learning capability comparing HAMLET against pure EBL with different utility thresholds. In order to learn from different kinds of domains, we have chosen one domain where IPSS is able to solve only a few problems (10%), Miconic, and another domain where IPSS is able to solve almost half of the problems (45%), Zenotime. These domains are the most interesting domains from this perspective. Learning in “hard” domains is difficult, given that HAMLET requires IPSS to solve problems. We will show in the next subsections some approaches for doing so. Learning in “easy” domains is usually not required. We have used the problems in the repository (simplest problems for training).[8] Then, we execute the PLTOOL learning modules that generate control-rule sets from the learning problems varying the *learning-type* HAMLET parameter, described in section 4.1, and the utility thresholds. Table 2 shows percentages of solved problems (*Solved*) when IPSS used each rule set, and the number of used rules (*Rules*). IPSS columns show the results when IPSS did not use any control knowledge. HAMLET columns show when the planner used the rules learned using HAMLET (*dynamic* value of the *learning-type* parameter). And EBL columns show the results when using the rules learned by EBL with different utility thresholds for pruning rules; *NT* no pruning, *T0* rules with an utility value less than 0 are pruned, *T5* rules with an utility value less than 5 are pruned. HAMLET rules allow IPSS to solve 100% of the problems in the Miconic domain, while IPSS base only solves 10% and EBL rules 68%.

Table 2 A comparison of HAMLET and EBL in two domains.

Domain	IPSS	HAMLET		EBL					
				NT		T0		T5	
	Solved	Solved	Rules	Solved	Rules	Solved	Rules	Solved	Rules
Zenotime	45%	55%	3	40%	17	40%	5	40%	4
Miconic	10%	100%	5	7%	43	43%	10	68%	10

[8]The training problems must be of a small size to guarantee that the planner finds all possible solutions, so that the search provides the positive and negative decision points made necessary for learning. The training set for the Miconic domain was formed by problems *mixed-f2-p1-r0, mixed-f2-p1-r1, mixed-f2-p1-r2, mixed-f2-p1-r3, mixed-f2-p1-r4, mixed-f4-p2-r0, mixed-f4-p2-r1, mixed-f4-p2-r2, mixed-f4-p2-r3, mixed-f4-p2-r4*. And, in the Zenotime domain by problems *ztrav01-1-2, ztrav02-1-3, ztrav03-2-4*.

The training set in the Zenotime was too small (only 3 problems) for testing HAMLET induction capability. Thus, we generated another training set with 100 random problems of 1 and 3 goals. Then, we tested against 100 randomly generated test problems ranging from 6 to 10 goals. Table 3 shows percentages of solved problems when it used: no control knowledge (*No learning*); the 3 rules learned from repository problems (*Repository*); and the 32 rules learned from random problems (*Random*). The table also shows the gain percentage in time to solve and quality of solutions (in the number of operators in the solution) of two rules sets compared against the base level IPSS. In order to compute the gain in time to solve and quality, we only used the problems solved by all configurations. As shown, learning from only three problems can significantly increase both the number of solved problems, but also the time spent on solving them, and the quality of the solutions. But, if we provide more training problems, the induction component of HAMLET allows us to improve even more the results with respect to the three criteria.

Table 3 Comparison of IPSS base with HAMLET trained with 3 problems and with 100 random problems in the Zenotime domain.

Configuration	Number of rules	Solved	Time Gain	Quality Gain
No learning	0	67%	-	-
Repository	3	73%	8%	8%
Random	32	88%	53%	21%

Figure 12 shows the accumulated time IPSS spent in solving all problems with and without rules. We only report on problems solved by all configurations. Figure 13 shows the quality of solutions of problems solved by all configurations.

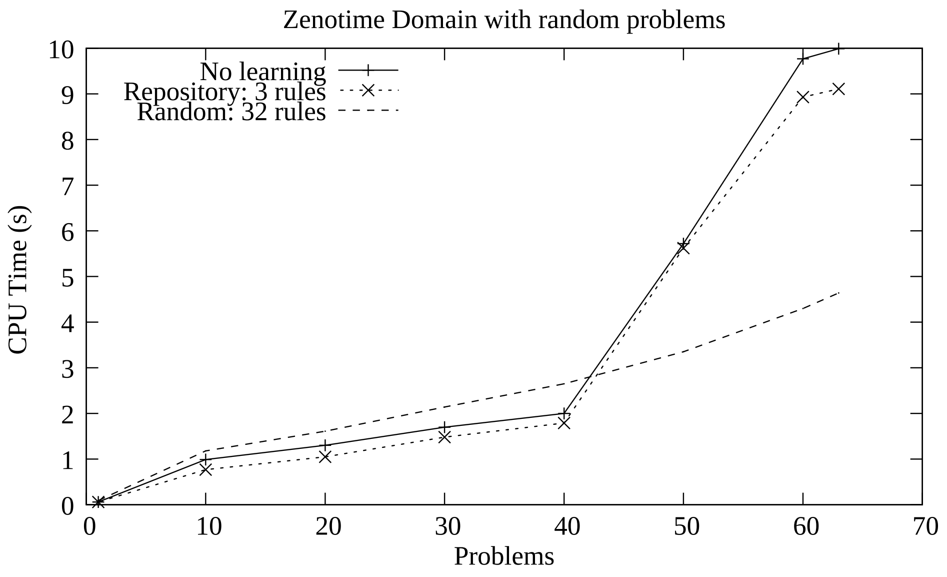


Figure 12 Accumulated time of IPSS base with HAMLET trained with 3 problems and with 100 random problems in the Zenotime domain.

6.3 *Using* FF *to help* HAMLET

Since PLTOOL incorporates FF, we experimented on using FF for bootstrapping the learning process (solving training problems that IPSS could not solve, so that HAMLET could then learn) as was explained in Section 4.1. Table 4 shows the obtained results. We used two domains where IPSS could not solve any competition problem, Logistics and Depots. In both domains,

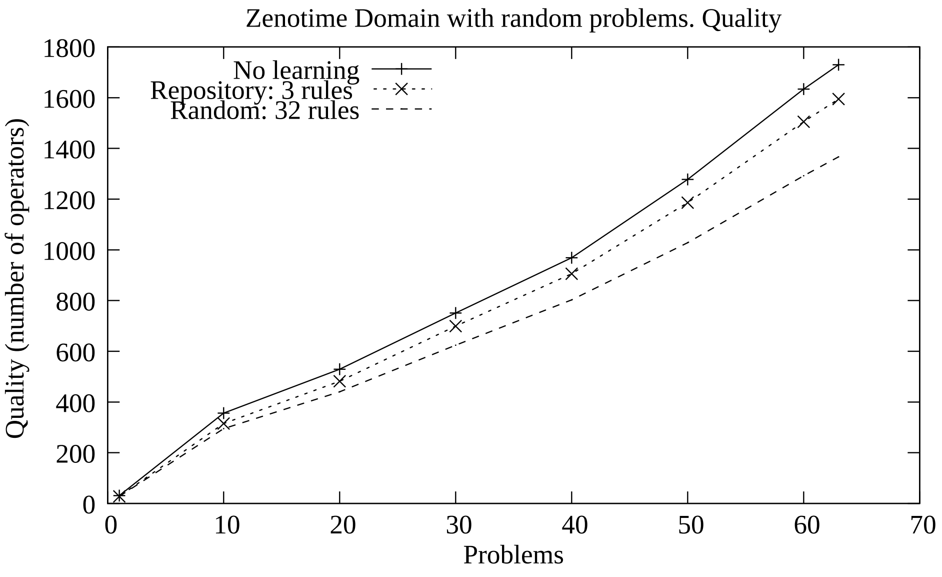


Figure 13 Accumulated quality of solutions of IPSS base with HAMLET trained with 3 problems and with 100 random problems in the Zenotime domain.

we separately trained HAMLET with 400 randomly generated training problems with 1 and 2 goals in the Logistics domain, and with 200 randomly generated training problems also with 1 and 2 goals in the Depots domain. In each case, the learning process generated the respective set of control rules. The same training problems were used to train HAMLET again, but calling FF when IPSS could not solve a problem, generating another set of rules. Then, we tested against randomly generated test problems. In the Logistics domain, we used 100 test problems ranging from 1 to 5 goals. In the Depots domain, we used 100 test problems also ranging from 1 to 5 goals. The time limit used in both domains was 15 seconds.

Table 4 Using FF planner for helping HAMLET. Results are percentage of solved problems.

Domain	IPSS	IPSS HAMLET Rules		IPSS+FF HAMLET Rules	
	Solved	Solved	Rules	Solved	Rules
Logistics	12%	23%	32	33%	34
Depots	8%	13%	5	26%	12

As shown, FF Rules increase the percentage of problems solved by more than 10% with respect to HAMLET rules and by about 20% with respect to the IPSS base planner in both domains. We are currently not using the complete power of this interaction, given that it is difficult to completely reconstruct the search tree that IPSS would follow from an FF solution. Therefore, in the future we expect to improve these results further by either learning control rules in FF and then using them in IPSS (we are doing something similar now with GRAPHPLAN) or by finding better ways of reconstructing the search tree.

6.4 Mixed initiative acquisition of control knowledge

The use of learning techniques used in previous experiments has two main drawbacks:

- when IPSS cannot solve the training problems, the learning tools cannot learn anything. Therefore, the learning tools cannot improve the IPSS base behaviour; and

- even in the case of domains where IPSS can solve problems, HAMLET and EBL by themselves are not able to obtain the optimal set of control rules (those that would allow IPSS to solve all problems, without backtracking and with the best quality).

In order to solve these problems, one could devise several strategies, such as developing better learning tools (we had good results in the past with genetic programming approaches (Aler et al. 2002), though from a KE perspective the good results obtained from the generated rules were hard to understand). Another alternative would be using better planners, which we are now working on. Yet another alternative would involve the user in this task. We believe that this is a valid approach from the perspective of building real systems, because the user then incorporates his/her experience, and can potentially refine the rules correctly.

Thus, in this subsection, we show the results of using a mixed-initiative approach to acquire control knowledge. We have chosen two domains where IPSS could not solve any repository problem, the Depots and the Driverlog domains. The time spent on generating the control knowledge (combining manual and machine-learning definition) was about 8 hours in each domain. The time limit used in both domains was 35 seconds. In the Depots domain, a human [9] defined the rules manually using 100 randomly generated problems of 1 and 2 goals, and then let HAMLET learn from the same problems. The human observes the search tree generating during problem solving. Then, s/he identifies the nodes where the planner chooses the wrong branch and defines a rule for choosing the correct option at each point. In the Driverlog domain, the human first trained HAMLET, which generated a set of control rules. Then, the user refined the heuristics using PLTOOL. The detailed results are shown in the appendix A: Table 8 for the Depots domain and Table 9 for the Driverlog domain. The *Hand-made rules* columns represent time to solve (*T* columns) and quality as the number of operators in the solution (*Q* columns) when the planner used the initial rules generated manually by the human. HAMLET *rules* show the results of the learned rules starting from the hand-made rules. We tested them using the problems in the repository. Table 5 summarizes those results. It shows the number of tested problems (*Total*) and the number of solved problems (*Solved*) by IPSS using the HAMLET rules and without rules. As shown, the mixed-initiative approach solved 17 (in the Depots) and 10 (in the Driverlog) of the problems while IPSS base could not solve any of them. Hand-made rules solved 16 problems and 2 more problems were solved after HAMLET learning. However, quality and time to solve are a bit better with hand-made rules.

Table 5 Number of solved problems of the mixed-initiative approach for acquiring heuristics, using the repository problems.

Domain	Total	No rules	HAMLET rules	
		Solved	Solved	Rules
Depots	22	0	17	17
Driverlog	20	0	10	30

6.5 *Using mixed-initiative learning for improving quality*

In the experiments of the previous section we showed how we can improve the behaviour of the planner by using a mixed-initiative approach in hard domains. However, we used the standard quality metric: the number of operators in the solution. Given that IPSS can handle different cost metrics, we did more experiments whose objective was to show the capabilities of PLTOOL for performing mixed-initiative acquisition of control knowledge to improve the quality of solutions using other cost metrics. In this case, we used two domains that have versions in which the goal is

[9]A PhD student with one year of experience on planning and control knowledge definition.

to improve the quality of solutions in terms of time or fuel: Zenotime (time version) and Satellite (numeric version).

Table 10 in appendix A shows the results in the Zenotime domain when we asked IPSS to find only the first solution. We used the time to execute the solution as the quality metric. We tested the base level planner IPSS, and two different rule sets generated using the mixed-initiative approach. The first set of rules, *Rules1* was generated by an interaction between HAMLET and the human without including numerical conditions in the control rules. The second set of rules *Rules2* was generated from the previous set of rules adding three rules manually that focused on the quality of the solutions by including numerical conditions. The T columns represent CPU-time in seconds to find the solution, the Q columns represent the quality of the solutions (in terms of the time to execute the solution), and the O columns represent the number of operators in the solution. As shown, both rule sets not only improved quality of the solutions but also allowed the planner to solve more problems. *Rules1* went from 9, IPSS base, up to 11 and *Rules2* allowed us to solve 16 of the problems. The CPU-time to solve the problems is nearly the same in all the cases.

We also tested how IPSS behaved when trying to find multiple solutions in a given time limit, 20s. This can potentially lead to find increasingly better-quality solutions (due to the branch-and-bound search). Table 11 in appendix A shows all the results in detail. Figure 14 shows the accumulated quality of the problems solved by all configurations when finding multiple solutions within the 20s time limit. Quality is measured as execution time of the solutions. As shown, the first rules improve the quality by 43% and the second rules by 50%.

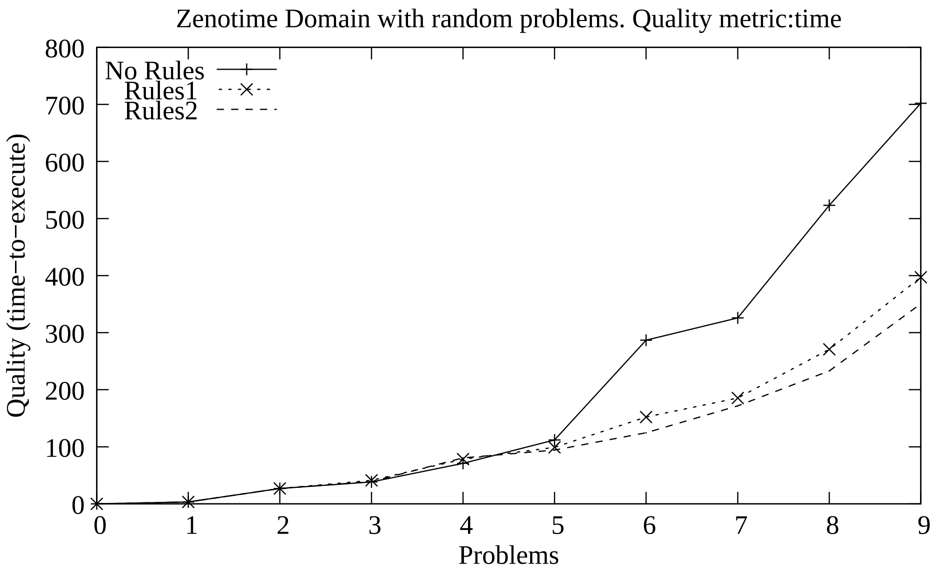


Figure 14 Accumulated quality (execution time) when using a mixed-initiative approach to generate control rules in the Zenotime domain with multiple solutions in 20s.

Table 12 in appendix A shows the results in the Satellite domain when asking IPSS to find only the first solution. The time limit was 60s. The quality metric was total fuel used to execute the solution. We used the rule generated using the mixed-initiative approach explained in Section 5 (see Figure 10). As shown, using only one rule improved the quality by about 19%, but there is a time penalty on finding the solutions, about 13s altogether. Then, we let IPSS find multiple solutions for 60s. Table 13 in appendix A shows all the results in detail. Figure 15 shows the accumulated quality when finding multiple solutions within the 60s time limit. As shown, the rule improves the quality by about 20%.

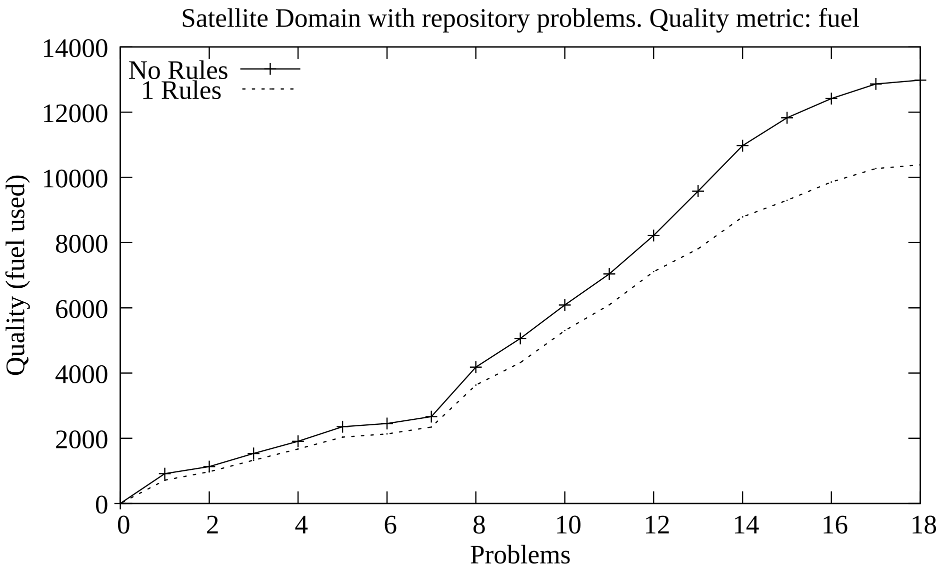


Figure 15 Results using control rules generated by the mixed-initiative approach in the Satellite domain. The quality metric was fuel used and multiple solutions were searched for 20s.

6.6 Combining macro-operators and mixed-initiative learning

As a final experiment, we wanted to explore the generation and use of macro-operators with a mixed-initiative approach of control-rules acquisition. A similar approach was introduced in (McCluskey 1987) where the control rules were learned automatically. We could have started by learning a set of macro-operators from some training problems. However, from a practical perspective, we found it better to select a macro-operator that has been reported to work well in the SATELLITE domain under MACRO-FF: `switch-on-turn-to-calibrate-turn-to-take-image`. It is composed of five base operators (`switch-on, turn-to, calibrate, turn-to,` and `take-image`). An interesting issue is that this macro-operator was learned and used in a forward-chaining planner, while we report on its use on a backward-chaining planner. This transfer of knowledge from one planner to another is easier in the case of macro-operators, while it is not so easy in the case of control rules.

We added this macro-operator at the end of the domain file, as described previously in Section 4.3. Then, we let HAMLET learn control rules that would tell the planner when to use it. HAMLET learned only one rule (shown in Figure 16) to select the macro-operator, but we saw that the rule was not good enough. The rule forces the selection of the macro-operator whenever the planner has a goal of `have_image` and there is a satellite without a calibrated instrument. Thus, this rule fires in almost all problems with many goals with poor results. This is so because the last operator is `take-image`. And this is the only domain operator that achieves the goal `have_image`. Given that almost all goals of all problems are to `have_image`, the macro-operator is relevant for trying to achieve all goals of all problems.

Thus, we generated a macro-operator again without the last operator: `switch-on-turn-to-calibrate-turn-to`. Then, we tested it in the `satellite-numeric` domain without considering quality (only the number of operators in the solution). We did not consider quality, because macro-operators focus more on efficiency (more solved problems in less time) than on quality. In this case, HAMLET learns three control rules to select the macro-operator and two control rules to select its bindings. HAMLET was only trained with the problem in Section 5.

After trying to solve several problems, we observed that the learned control rules for selecting bindings do not work well because they lack one condition. This condition is needed to correctly

```
(control-rule SELECT-SWITCH-ON-TURN-TO-CALIBRATE-TURN-TO-TAKE-IMAGE
  (if (and (current-goal (have_image <direction-15036-1> <mode-15036-2>))
           (true-in-state (supports <instrument-15036-5> <mode-15036-2>))
           (true-in-state (calibration_target <instrument-15036-5> <direction-15036-6>))
           (true-in-state (on_board <instrument-15036-5> <satellite-15036-3>))
           (true-in-state (pointing <satellite-15036-3> <direction-15036-7>))
           (true-in-state (power_avail <satellite-15036-3>))
           (some-candidate-goals ((have_image <direction-15036-8> <mode-15036-9>)))
           (type-of-object <infinite-15036-18> top-type)
           (type-of-object <infinite-15036-14> top-type)
           (type-of-object <direction-15036-7> direction)
           (type-of-object <satellite-15036-3> satellite)
           (type-of-object <direction-15036-6> direction)
           (type-of-object <direction-15036-8> direction)
           (type-of-object <mode-15036-9> mode)
           (type-of-object <instrument-15036-5> instrument)
           (type-of-object <mode-15036-2> mode)
           (type-of-object <direction-15036-1> direction)))
(then select operators switch-on-turn-to-calibrate-turn-to-take-image))
```

Figure 16 Control rule learned by HAMLET for selecting macro-operator `switch-on-turn-to-calibrate-turn-to-take-image` in satellite domain.

instantiate the final direction of the last `turn_to` operator in the macro-operator. Since taking an image is the only reason to switch on and calibrate an instrument, the correct instantiation is the direction of a `have_image` goal in which the planner was working. Therefore the user adds the condition `(prior-goal (have_image <direction-4043-3> <mode>))` which tests whether this goal is the top-level goal of the goal the planner is currently working on. Figure 17 shows an example of a refined control rule for selecting the bindings.

Also, s/he adds a preference control rule to start solving the goal `have_image` instead of the goal to have the satellite pointing to a given direction. In the satellite domain it is more important to focus first on how to take images, and then on where the satellite points to. As shown in Table 6, by combining the macro-operator with automatically learned control rules, we improve the behaviour over IPSS in terms of the number of operators in the solutions, O. However, it is worse in terms of time to solve, T, given that the bindings control rules are incorrect. But, if we let the user interact and modify those control rules, we obtain much better behaviour (30% better quality over IPSS base and a 10% improvement in time to solve). The time spent on generating the control knowledge (combining manual and machine-learning definition) was about 3 hours.

In the next experiment we tested the quality of solutions when using the macro-operator (the same macro-operator as before). We added the macro-operator at the beginning (*Macro first*) and at the end (*Macro last*) of the domain file. We also used the same learned control rules by HAMLET and the rules modified by the mixed-initiative approach. By adding it at the beginning, we force IPSS to always use it as the first option for achieving one of its effects. Given that it is composed of four base operators, solutions require more fuel than really needed (more operators than really needed are used). So, even if the time to solve is better than not using it, the quality is worse. Even if the control rules do not incorporate numeric information, needed for improving the quality of solutions using them, they tell us when to use the macro-operator. Results show an improvement in quality over using the macro-operator only. They do not improve the time to solve because of the missing condition of bindings control rules previously mentioned. Using the mixed-initiative approach, IPSS time to solve is worse (probably due to control rules matching, given that it decreases the number of searched nodes), but quality of solutions improves by more than 50%.

```
(control-rule SELECT-BIND-SWITCH-ON-TURN-TO-CALIBRATE-TURN-TO
  (if (and (current-operator switch-on-turn-to-calibrate-turn-to)
           (current-goal (power_on <instrument-4037-2>))
           (prior-goal (have_image  <direction-4043-3> <mode>))
           (true-in-state (on_board <instrument-4037-2> <satellite-4037-1>))
           (true-in-state (power_avail <satellite-4037-1>))
           (true-in-state (pointing <satellite-4037-1> <direction-4037-5>))
           (true-in-state (calibration_target <instrument-4037-2> <direction-4037-4>))
           (some-candidate-goals ((calibrated <instrument-4037-2>)
                                  (pointing <satellite-4037-1> <direction-4037-3>)))
           (type-of-object <satellite-4037-1> satellite)
           (type-of-object <instrument-4037-2> instrument)
           (type-of-object <direction-4037-3> direction)
           (type-of-object <direction-4037-4> direction)
           (type-of-object <direction-4037-5> direction)
           (type-of-object <infinite-4037-6> top-type)
           (type-of-object <infinite-4037-18> top-type)
           (type-of-object <infinite-4037-15> top-type)))
  (then select bindings
        ((<s> . <satellite-4037-1>) (<i> . <instrument-4037-2>)
         (<d_new> . <direction-4037-3>) (<d_prev> . <direction-4037-4>)
         (<d_prevprev> . <direction-4037-5>) (<f> . <infinite-4037-6>)
         (<sta> . <infinite-4037-18>) (<stb> . <infinite-4037-15>))))
```

Figure 17 Example of refined control rule for selecting the bindings of the macro-operator `switch-on turn-to calibrate turn-to` in the satellite domain.

Table 6 The time to solve and the number of operators in the solution on using a macro and rules generated by the mixed-initiative approach in the Satellite domain.

Problem	Without macros		Macro+HAMLET rules		Macro+Mixed-initiative rules	
	T	O	T	O	T	O
STRIPS-SAT-X-1	0.17	147	0.49	123	0.53	119
STRIPS-SAT-X-2	0.08	107	0.27	91	3.79	83
STRIPS-SAT-X-3	0.11	123	0.30	95	0.23	71
STRIPS-SAT-X-4	0.05	83	0.11	67	0.20	63
STRIPS-SAT-X-5	0.08	91	0.09	83	0.11	71
STRIPS-SAT-X-6	0.05	103	0.12	83	0.14	75
STRIPS-SAT-X-7	0.02	59	0.10	51	0.04	47
STRIPS-SAT-X-9	0.02	71	0.06	51	0.06	43
STRIPS-SAT-X-11	0.76	227	3.13	131	7.81	111
STRIPS-SAT-X-12	1.06	239	26.80	199	2.99	175
STRIPS-SAT-X-15	1.41	335	2.65	211	3.30	179
STRIPS-SAT-X-17	0.86	223	1.23	171	1.40	163
STRIPS-SAT-X-18	0.18	143	0.78	127	0.50	147
STRIPS-SAT-X-19	19.02	147	0.45	127	0.41	119
STRIPS-SAT-X-20	0.01	39	0.03	31	0.09	27
TOTAL	23.88	2137	36.61	1641	21.60	1493

7 Conclusions and future work

One of the reasons for the lack of real applications of AI planning technology, is the gap between current planning techniques and tools needed to build, execute and monitor planning applications (McCluskey et al. 2003a). During the ICAPS conference in 2005, the first international competition on using KE tools for planning was held. One of the competition goals was to push the field of planning and scheduling towards building KE tools. KE for AI Planning has been defined as *the process that deals with the acquisition, validation and maintenance of planning*

Table 7 The time to solve, the number of nodes and the quality of solutions on using a macro combined with control rules in the SATELLITE domain.

Problem	Macro first			Macro last + HAMLET rules			Macro last + Mixed-initiative rules		
	T	N	Q	T	N	Q	T	N	Q
STRIPS-SAT-X-1	0.20	119	2380.65	0.58	123	1045.16	0.55	119	792.01
STRIPS-SAT-X-2	0.30	91	798.41	0.40	91	235.42	0.44	83	217.90
STRIPS-SAT-X-3	0.24	164	1978.26	0.35	95	562.17	0.20	71	358.18
STRIPS-SAT-X-4	0.11	67	704.46	0.18	67	428.17	0.14	63	360.35
STRIPS-SAT-X-5	0.17	83	696.84	0.12	83	466.22	0.13	71	621.86
STRIPS-SAT-X-6	0.12	75	218.98	0.19	83	477.34	0.12	75	558.05
STRIPS-SAT-X-7	0.07	51	503.14	0.09	51	168.44	0.04	47	109.80
STRIPS-SAT-X-9	0.09	47	421.45	0.07	51	338.85	0.06	43	209.73
STRIPS-SAT-X-10	6.62	279	3443.93	4.12	275	1669.99	5.27	267	1719.90
STRIPS-SAT-X-12	1.85	215	2702.66	3.14	199	1215.12	2.92	175	1009.29
STRIPS-SAT-X-14	1.34	215	2361.96	3.51	239	1108.86	2.69	227	1106.47
STRIPS-SAT-X-18	0.65	123	1693.69	27.91	127	663.41	0.44	147	559.14
STRIPS-SAT-X-19	0.59	137	1413.39	0.53	127	558.90	0.43	119	448.69
STRIPS-SAT-X-20	0.03	35	461.08	0.08	31	197.38	0.03	27	184.28
TOTAL	12.38	1701	19778.90	41.27	1642	9135.43	13.46	1534	8255.65

domain models, and the selection and optimization of appropriate planning machinery to work on them. This paper presents a new KE tool, PLTOOL, which integrates several planners together with several machine learning techniques. The main aim is to help in the KE stage related with the acquisition of knowledge oriented to optimizing the behaviour of the planner.

PLTOOL currently incorporates two planners, IPSS and FF. It also incorporates three different learning modules that can automatically acquire control knowledge or new operators (macro-operators). We believe that users can greatly benefit from using this type of tools when building new planning applications, given that they remove part of the KE burden. More specifically, in the case of PLTOOL, the focus is on acquiring domain-dependent heuristics that can guide planners towards good quality solutions. Acquiring good heuristics could require: combining the knowledge learned by means of several machine learning techniques (for instance, as shown in Section 6.6, macro-operators and control rules); using several planners in the learning process (such as the combination of IPSS and FF explained in Section 4); or allowing the user to revise and modify the automatically learned heuristics. Therefore, we provide an integration of different learning techniques and planners in the same tool, together with a friendly way of human interaction.

Section 6 shows a wide range of experiments that can be carried out with PLTOOL. We have performed these experiments using domains that range from "easy" to "hard" and using different quality criteria. The results show that interaction between the user and a KE tool for planning and learning can be useful for improving not only the efficiency of the planner (the number of solved problems or time to solve), but also the quality of solutions. In most cases, humans have used limited time to generate or modify control knowledge with good results.

In the future, we plan to enhance PLTOOL by adding new planners, so that different planning perspectives can be integrated into the same tool. Once we have several planning modules, we can learn domain-dependent heuristics separately from each planner and create a repository of heuristic knowledge. This knowledge can be re-used by other planners. The user can then extract this heuristic knowledge, from different paradigms, and build better planning applications.

Acknowledgements

This work has been partially supported by the Spanish MCyT project TIC2002-04146-C05-05, MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project UC3M-INF-05-016.

References

R. Aler & D. Borrajo (2002). 'On Control Knowledge Acquisition by Exploiting Human-Computer Interaction'. In M. Ghallab, J. Hertzberg, & P. Traverso (eds.), *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pp. 112–120, Toulouse (France). AAAI Press.

R. Aler, et al. (2000). ‘Knowledge Representation Issues in Control Knowledge Learning’. In P. Langley (ed.), *Proceedings of the Seventeenth International Conference on Machine Learning, ICML’00*, pp. 1–8, Stanford, CA (USA).

R. Aler, et al. (2002). ‘Using Genetic Programming to Learn and Improve Control Knowledge’. *Artificial Intelligence* **141**(1-2):29–56.

J. L. Ambite, et al. (2000). ‘Learning Plan Rewriting Rules’. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pp. 14–17, Breckenbridge, CO (USA).

J. D. Arias, et al. (2005). ‘Using ontologies for planning tourist visits’. In *Working notes of the ICAPS’05 Workshop on Role of Ontologies in Planning and Scheduling*, pp. 52–59, Monterey, CA (EEUU). AAAI, AAAI Press.

F. Bacchus & F. Kabanza (2000). ‘Using Temporal Logics to Express Search Control Knowledge for Planning’. *Artificial Intelligence* **116**:123–191.

C. Bäckström (1992). *Computational Complexity of Reasoning about Plans.* Ph.D. thesis, Linkoping University, Linkoping, Sweden.

A. L. Blum & M. L. Furst (1995). ‘Fast Planning Through Planning Graph Analysis’. In C. S. Mellish (ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, vol. 2, pp. 1636–1642, Montréal, Canada. Morgan Kaufmann.

B. Bonet & H. Geffner (2001). ‘Planning as Heuristic Search’. *Artificial Intelligence* **129**(1-2):5–33.

D. Borrajo, et al. (1999). ‘Multistrategy Relational Learning of Heuristics for Problem Solving’. In M. Bramer, A. Macintosh, & F. Coenen (eds.), *Research and Development in Intelligent Systems XVI. Proceedings of Expert Systems 99, The 19th SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, BCS Conference Series, pp. 57–71, Cambridge, England. Springer-Verlag.

D. Borrajo, et al. (2001). ‘Quality-based Learning for Planning’. In *Working notes of the IJCAI’01 Workshop on Planning with Resources*, pp. 9–17, Seattle, WA (USA). IJCAI Press.

D. Borrajo & M. Veloso (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. Also in the book ”Lazy Learning”, David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.

A. Botea, et al. (2005). ‘Learning Partial-Order Macros from Solutions’. In *Proceedings of ICAPS’05*, Monterrey (USA).

T. Bylander (1994). ‘The computational complexity of propositional STRIPS planning’. *Artificial Intelligence* **69**(1-2):165–204.

J. G. Carbonell, et al. (1992). ‘PRODIGY4.0: The Manual and Tutorial’. Tech. Rep. CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University.

L. Castillo, et al. (2001). ‘Mixing expressiveness and efficiency in a manufacturing planner’. *Journal of Experimental and Theoretical Artificial Intelligence* **13**:141–162.

A. Cesta, et al. (2002). ‘A Constrained-Based Method for Project Scheduling with Time Windows’. *Journal of Heuristics* **8**:109–136.

G. Cortellesa & A. Cesta (2006). ‘Feature Evaluation in Mixed-Initiative Systems: An Experimental Approach’. In D. B. Derek Long, Steve Smith & L. McCluskey (eds.), *Proceedings of ICAPS’06*, Ambleside (UK). AAAI Press.

K. Currie & A. Tate (1991). ‘O-Plan: the Open Planning Architecture’. *Artificial Intelligence* **52**(1):49–86.

T. A. Estlin & R. J. Mooney (1997). ‘Learning to Improve both Efficiency and Quality of Planning’. In M. Pollack (ed.), *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1227–1232. Morgan Kaufmann.

O. Etzioni & S. Minton (1992). ‘Why EBL produces overly-specific knowledge: A critique of the Prodigy approaches’. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 137–143, Aberdeen, Scotland. Morgan Kaufmann.

S. Fernández, et al. (2004). ‘Using Previous Experience for Learning Planning Control Knowledge’. In V. Barr & Z. Markov (eds.), *Proceedings of the Seventeen International Florida Artificial Intelligence (FLAIRS04)*, pp. 713–718, Miami Beach, FL (USA). AAAI Press.

S. Fernández, et al. (2005). ‘Machine Learning in Hybrid Hierarchical and Partial-Order Planners for Manufacturing Domains’. *Applied Artificial Intelligence* **19**(8):783–809.

R. E. Fikes, et al. (1972). ‘Learning and Executing Generalized Robot Plans’. *Artificial Intelligence* **3**:251–288.

M. Ghallab, et al. (2004). *Automated Task Planning. Theory & Practice.* Morgan Kaufmann.

J. Gratch & G. DeJong (1992). ‘COMPOSER: A probabilistic solution to the utility problem in speed-up learning’. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 235–240.

J. Hoffmann & B. Nebel (2001). ‘The FF Planning System: Fast Plan Generation Through Heuristic Search’. *Journal of Artificial Intelligence Research* **14**:253–302.

Y.-C. Huang, et al. (2000). 'Learning Declarative Control Rules for Constraint-Based Planning'. In P. Langley (ed.), *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, Stanford, CA (USA).

R. L. Joseph (1989). 'Graphical Knowledge Acquisition'. In *Proceedings of the* 4th *Knowledge Acquisition For Knowledge-Based Systems Workshop*, Banff, Canada.

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

S. Kambhampati (2000). 'Planning Graph as a (Dynamic) CSP: Exploiting EBL, DDB and other CSP Search Techniques in Graphplan'. *Journal of Artificial Intelligence Research* **12**:1–34.

R. Khardon (1999). 'Learning Action Strategies for Planning Domains'. *Artificial Intelligence* **113**(1-2):125–148.

C. A. Knoblock, et al. (1991). 'Integrating Abstraction and Explanation Based Learning in PRODIGY'. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 541–546.

R. E. Korf (1985). 'Macro-operators: A Weak Method for Learning'. *Artificial Intelligence* **26**:35–77.

L. McCluskey, et al. (2003a). 'Knowledge Engineering for Planning ROADMAP'.

T. L. McCluskey (1987). 'Combining Weak Learning Heuristics in General Problem Solvers'. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI'87).* Morgan Kaufman.

T. L. McCluskey (1989). 'Explanation-based and Similarity-based Heuristic Acquisition in a General Planner'. In *Proceedings of the 4th European Working Session on Learning.* Pitman.

T. L. McCluskey, et al. (2003b). 'GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment'. In *Proceedings of ICAPS'03*, Trento (Italia). AAAI Press.

T. L. McCluskey & J. M. Porteous (1997). 'Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency'. *Artificial Intelligence* **95**(1):1–65.

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

S. Minton, et al. (1989). 'PRODIGY 2.0: The Manual and Tutorial'. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University.

T. M. Mitchell, et al. (1986). 'Explanation-Based Generalization: A Unifying View'. *Machine Learning* **1**:47–80.

K. Myers & D. Wilkins (1997). 'The Act-Editor User's Guide: A Manual for Version 2.2'. Tech. rep., SRI.

D. Nau, et al. (2003). 'SHOP2: An HTN Planning System'. *Journal of Artificial Intelligence Research* **20**:379–404.

Y. Qu & S. Kambhampati (1995). 'Learning Search Control Rules for Plan-space Planners: Factors affecting the Performance'. Tech. rep., Arizona State University.

M. D. Rodríguez-Moreno, et al. (2004a). 'An AI Planning-based Tool for Scheduling Satellite Nominal Operations'. *AI Magazine* **25**(4):9–27.

M. D. Rodríguez-Moreno, et al. (2004b). 'IPSS: A Problem Solver that Integrates P&S'. In *Third Italian Workshop on Planning and Scheduling.*

M. D. Rodríguez-Moreno, et al. (2004c). 'IPSS: A Hybrid Reasoner for Planning and Scheduling'. In R. L. de Mántaras & L. Saitta (eds.), *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, pp. 1065–1066, Valencia (Spain). IOS Press.

M. A. Upal & R. Elio (2000). 'Learning search control rules versus rewrite rules to improve plan quality'. In *Proceedings of the Thirteenth Canadian Conference on Artificial Intelligence*, pp. 240–253, New York. Springer-Verlag.

M. Veloso (1994). *Planning and Learning by Analogical Reasoning.* Springer Verlag.

M. Veloso, et al. (1995). 'Integrating Planning and Learning: The PRODIGY Architecture'. *Journal of Experimental and Theoretical AI* **7**:81–120.

X. Wang (1994). 'Learning Planning Operators by Observation and Practice'. In *Proceedings of the Second International Conference on AI Planning Systems, AIPS-94*, pp. 335–340, Chicago, IL. AAAI Press, CA.

Q. Yang, et al. (2005). 'Learning Action Models from Plan Examples with Incomplete Knowledge'. In *Proceedings of ICAPS'05*, Monterrey (USA).

T. Zimmerman & S. Kambhampati (2003). 'Learning-Assisted Automated Planning: Looking Back, Taking Stock, Going Forward'. *AI Magazine* **24**(2):73–96.

Appendix A Detailed tables of results

The following tables show the detailed results of experiments described in previous sections.

Table 8 Quality solutions and time to solve on a mixed-initiative approach for acquiring heuristics, using the repository problems in the Depots domain. The quality metric is the number of operators in the solution.

Problem	No rules		15 Hand-made rules		17 HAMLET rules	
	Q	T	Q	T	Q	T
DEPOTPROB1817-P22	-	-	127	6.95	132	7.77
DEPOTPROB8715-P21	-	-	37	0.67	37	0.79
DEPOTPROB7615-P20	-	-	-	-	121	2.79
DEPOTPROB6178-P19	-	-	45	0.54	45	0.69
DEPOTPROB1916-P18	-	-	82	1.10	83	1.24
DEPOTPROB6587-P17	-	-	35	0.20	39	0.37
DEPOTPROB4398-P16	-	-	30	0.16	30	0.17
DEPOTPROB4534-P15	-	-	-	-	-	-
DEPOTPROB7654-P14	-	-	31	0.38	31	0.34
DEPOTPROB5646-P13	-	-	25	0.16	29	0.18
DEPOTPROB9876-P12	-	-	70	1.77	68	0.93
DEPOTPROB8765-P11	-	-	-	-	103	1.24
DEPOTPROB7654-P10	-	-	27	0.20	35	0.25
DEPOTPROB5451-P9	-	-	-	-	-	-
DEPOTPROB4321-P8	-	-	35	0.23	52	0.39
DEPOTPROB1234-P7	-	-	24	0.07	32	0.13
DEPOTPROB5656-P6	-	-	-	-	-	-
DEPOTPROB1212-P5	-	-	-	-	-	-
DEPOTPROB6512-P4	-	-	33	0.22	48	0.42
DEPOTPROB1935-P3	-	-	31	0.17	-	-
DEPOTPROB7512-P2	-	-	17	0.04	20	0.10
DEPOTPROB1818-P1	-	-	11	0.02	11	0.07

Table 9 Quality solutions and time to solve of the mixed-initiative approach for acquiring heuristics, using the repository problems of the Driverlog domain. The quality metric is the number of operators in the solution.

Problem	No rules		30 HAMLET rules	
	Q	T	Q	T
DLOG-8-6-25-1	-	-	-	-
DLOG-5-5-20-2	-	-	-	-
DLOG-5-5-10-3	-	-	-	-
DLOG-3-3-6-4	-	-	-	-
DLOG-2-3-6-5	-	-	-	-
DLOG-2-3-6-6	-	-	31	0.36
DLOG-3-3-7-7	-	-	40	0.52
DLOG-3-3-5-8	-	-	21	0.24
DLOG-3-2-4-9	-	-	41	0.29
DLOG-5-5-15-10	-	-	-	-
DLOG-4-4-8-11	-	-	-	-
DLOG-2-3-6-12	-	-	-	-
DLOG-2-3-6-13	-	-	25	0.36
DLOG-2-3-6-14	-	-	68	1.04
DLOG-3-3-6-15	-	-	26	0.37
DLOG-3-2-5-16	-	-	23	0.17
DLOG-2-2-4-17	-	-	28	0.15
DLOG-2-2-3-18	-	-	-	-
DLOG-2-2-2-19	-	-	16	6.04
DLOG-5-5-25-20	-	-	-	-

Table 10 Quality solutions and time to solve in using a mixed-initiative approach to generate control rules in the Zenotime domain. We used time-to-execute as the quality metric. IPSS stopped after finding the first solution.

Problem	No rules			13 Rules1			16 Rules2		
	T	Q	O	T	Q	O	T	Q	O
ZTRAV01-1-2	0.01	3.42	1	0.01	3.42	1	0.01	3.42	1
ZTRAV02-1-3	0.01	23.43	6	0.01	23.43	6	0.02	23.43	6
ZTRAV03-2-4	0.02	13.56	7	0.02	18.43	8	0.04	14.13	14
ZTRAV04-2-5	0.02	32.92	11	0.02	40.94	11	0.07	45.06	12
ZTRAV05-2-4	0.03	41.04	19	0.03	24.74	14	0.04	13.85	16
ZTRAV06-2-5	4.28	174.74	36	0.06	55.27	16	0.05	30.47	22
ZTRAV07-2-6	0.05	38.94	17	0.04	37.29	17	0.04	50.54	17
ZTRAV08-3-6	-	-	-	0.05	53.29	16	0.11	74.37	36
ZTRAV09-3-7	-	-	-	0.09	101.12	29	0.15	74.23	34
ZTRAV10-3-8	0.35	197.38	62	0.11	88.48	34	0.18	63.80	46
ZTRAV11-3-7	-	-	-	-	-	-	0.36	48.65	30
ZTRAV12-3-8	-	-	-	-	-	-	0.14	93.79	40
ZTRAV13-3-10	0.15	179.02	54	0.13	128.64	39	0.22	117.50	43
ZTRAV14-5-10	-	-	-	-	-	-	0.34	96.58	53
ZTRAV15-5-15	-	-	-	-	-	-	0.48	456.04	59
ZTRAV16-5-15	-	-	-	-	-	-	-	-	-
ZTRAV17-5-20	-	-	-	-	-	-	1.46	122.89	108
ZTRAV18-5-20	-	-	-	-	-	-	-	-	-
ZTRAV19-5-25	-	-	-	-	-	-	-	-	-
ZTRAV20-5-25	-	-	-	-	-	-	-	-	-

Table 11 Quality solutions and time to solve in using a mixed-initiative approach to generate control rules in the Zenotime domain. We used time-to-execute as the quality metric. IPSS generated multiple solutions within the time limit of 20 s.

Problem	No rules			13 Rules1			16 Rules2		
	T	Q	O	T	Q	O	T	Q	O
ZTRAV01-1-2	0.02	3.42	1	0.01	3.42	1	0.01	3.42	1
ZTRAV02-1-3	20.63	23.43	6	0.40	23.43	6	0.33	23.43	6
ZTRAV03-2-4	20.01	11.55	7	20.00	14.30	10	20.92	12.30	12
ZTRAV04-2-5	20.00	32.65	12	20.00	37.18	11	20.00	41.30	12
ZTRAV05-2-4	20.00	41.04	19	20.00	20.62	14	20.00	13.85	16
ZTRAV06-2-5	20.00	174.74	36	20.11	53.19	19	20.46	30.47	22
ZTRAV07-2-6	20.00	38.94	17	20.00	33.42	19	20.00	46.95	17
ZTRAV08-3-6	-	-	-	20.00	51.92	16	20.00	74.13	36
ZTRAV09-3-7	-	-	-	20.00	100.15	30	20.00	70.83	34
ZTRAV10-3-8	20.00	197.38	62	20.00	85.42	34	20.00	61.34	46
ZTRAV11-3-7	-	-	-	-	-	-	20.00	48.65	30
ZTRAV12-3-8	-	-	-	-	-	-	20.00	86.39	38
ZTRAV13-3-10	20.00	179.02	54	20.00	126.14	40	20.00	117.50	43
ZTRAV14-5-10	-	-	-	-	-	-	20.00	94.99	53
ZTRAV15-5-15	-	-	-	-	-	-	20.00	456.04	59
ZTRAV16-5-15	-	-	-	-	-	-	-	-	-
ZTRAV17-5-20	-	-	-	-	-	-	20.00	122.89	108
ZTRAV18-5-20	-	-	-	-	-	-	-	-	-
ZTRAV19-5-25	-	-	-	-	-	-	-	-	-
ZTRAV20-5-25	-	-	-	-	-	-	-	-	-

Table 12 Quality solutions and time to solve in using rules generated by the mixed-initiative approach in the Satellite domain. The quality metric is consumed fuel.

Problem	No rules			1 Rule		
	T	Q	O	T	Q	O
STRIPS-SAT-X-1	0.15	914.46	36	0.38	711.60	34
STRIPS-SAT-X-2	0.08	217.90	26	0.22	325.59	26
STRIPS-SAT-X-3	0.07	400.15	30	0.18	352.84	29
STRIPS-SAT-X-4	0.04	372.25	20	0.09	345.83	20
STRIPS-SAT-X-5	0.04	447.67	22	0.10	394.13	20
STRIPS-SAT-X-7	0.02	127.17	14	0.04	127.17	14
STRIPS-SAT-X-9	0.03	209.73	17	0.06	209.73	17
STRIPS-SAT-X-10	1.00	1522.19	99	3.73	1318.28	82
STRIPS-SAT-X-11	0.27	891.06	56	0.84	687.31	47
STRIPS-SAT-X-12	0.43	1029.43	59	2.58	982.23	58
STRIPS-SAT-X-13	0.41	953.68	57	2.08	790.61	53
STRIPS-SAT-X-14	0.52	1175.98	71	1.90	1017.36	62
STRIPS-SAT-X-15	0.52	1361.11	83	1.28	699.81	53
STRIPS-SAT-X-16	0.71	1395.10	74	2.39	978.76	68
STRIPS-SAT-X-17	0.30	857.55	55	1.17	510.55	47
STRIPS-SAT-X-18	0.15	587.72	35	0.44	573.27	33
STRIPS-SAT-X-19	0.11	448.69	36	0.41	411.19	36
STRIPS-SAT-X-20	0.02	184.28	9	0.02	184.28	9
TOTAL	4.87	13096.12	799	17.91	10620.54	708

Table 13 Quality solutions and time to solve in using rules generated by the mixed-initiative approach in the Satellite domain when finding multiple solutions. The quality metric is consumed fuel.

PROBLEM	No rules			1 Rules		
	T	Q	O	T	Q	O
STRIPS-SAT-X-1	60	913.90	37	60.03	711.60	34
STRIPS-SAT-X-2	60.00	217.90	26	60.03	260.15	27
STRIPS-SAT-X-3	60.00	400.15	30	60.00	352.84	29
STRIPS-SAT-X-4	62.25	372.25	20	60.00	345.83	20
STRIPS-SAT-X-5	60.00	447.67	22	60.00	362.79	22
STRIPS-SAT-X-7	60.00	98.56	16	60.00	98.56	16
STRIPS-SAT-X-9	60.00	209.73	17	61.41	209.73	17
STRIPS-SAT-X-10	60.00	1522.19	99	60.00	1290.91	86
STRIPS-SAT-X-11	60.00	875.75	57	60.00	687.31	47
STRIPS-SAT-X-12	60.00	1029.43	59	60.00	982.23	58
STRIPS-SAT-X-13	60.00	953.68	57	60.00	790.61	53
STRIPS-SAT-X-14	60.00	1175.98	71	60.00	1017.36	62
STRIPS-SAT-X-15	60.00	1361.11	83	60.00	699.81	53
STRIPS-SAT-X-16	60.00	1394.74	75	60.00	978.76	68
STRIPS-SAT-X-17	60.00	857.55	55	60.00	510.55	47
STRIPS-SAT-X-18	60.00	587.72	35	60.00	559.08	34
STRIPS-SAT-X-19	61.33	448.69	36	61.67	411.19	36
STRIPS-SAT-X-20	60.00	117.49	10	60.00	117.49	10
TOTAL		12984.49	805		10386.80	719