Machine Learning for Adaptive Planning

Dimitris Vrakas, Grigorios Tsoumakas, Nick Bassiliades and Ioannis Vlahavas

Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, GREECE

Tel: +30 2310 998231, +30 2310 998418, +30 2310 998145

Fax: +30 2310 998362, +30 2310 998419

E-mail: [dvrakas, greg, nbassili, vlahavas]@csd.auth.gr

Machine Learning for Adaptive Planning

Abstract

This chapter is concerned with the enhancement of planning systems using techniques from Machine Learning in order to automatically configure their planning parameters according to the morphology of the problem in hand. It presents two different adaptive systems that set the planning parameters of a highly adjustable planner based on measurable characteristics of the problem instance. The planners have acquired their knowledge from a large data set produced by results from experiments on many problems from various domains. The first planner is a rule-based system that employs propositional rule learning to induce knowledge that suggests effective configuration of planning parameters based on the problem's characteristics. The second planner employs instance-based learning in order to find problems with similar structure and adopt the planner configuration that has proved in the past to be effective on these problems. The validity of the two adaptive systems is assessed through experimental results that demonstrate the boost in performance in problems of both known and unknown domains. Comparative experimental results for the two planning systems are presented along with a discussion of their advantages and disadvantages.

INTRODUCTION

Domain independent heuristic planning relies on ingenious techniques, such as heuristics and search strategies, to improve the execution speed of planning systems and the quality of their solutions in arbitrary planning problems. However, no single technique has yet proved to be the best for all kinds of problems. Many modern planning systems incorporate more than one such optimizing techniques in order to capture the peculiarities of a wider range of problems.

However, to achieve the optimum performance these planners require manual fine-tuning of their run-time parameters.

Few attempts have been made to explain which are the specific dynamics of a planning problem that favor a specific planning technique and even more, which is the best setup for a planning system given the characteristics of the planning problem. This kind of knowledge would clearly assist the planning community in producing flexible systems that could automatically adapt themselves to each problem, achieving best performance.

This chapter focuses on the enhancement of Planning Systems with Machine Learning techniques in the direction of developing Adaptive Planning Systems that can configure their planning parameters automatically in order to effectively solve each different Planning problem. More specifically, it presents two different Machine Learning approaches for Adaptive Planning: a) Rule learning and b) Instance-based learning. Both approaches are described in detail and their performance is assessed through several experimental results that exhibit different aspects of the learning process. In addition, the chapter provides an extended overview of past approaches on combining Machine Learning and Automated Planning, two of the most important areas of Artificial Intelligence.

The rest of the chapter is organized as follows: The next section reviews related work on combining learning and planning and discusses the adopted learning techniques. Then the problem of the automatic configuration of planning systems is analyzed. The following two sections present the two learning approaches that have been used for the adaptive systems and present experimental results that compare them and show the gain in the performance over the initial planner. Finally, the last section discusses several issues concerning the two learning approaches, concludes the chapter and poses future research directions.

MACHINE LEARNING FOR AUTOMATED PLANNING

Machine Learning is the area of Artificial Intelligence concerned with the design of computer programs that improve at a category of tasks with experience. It is a very broad field with many learning methodologies and numerous algorithms, which have been extensively exploited in the past to support Planning systems in many ways. Since it is a usual case for seemingly different planning problems to present similarities in their structure, it is reasonable enough to believe that planning strategies that have been successfully applied to some problems in the past will be also effective for similar problems in the future. Learning can assist planning systems in three ways: a) to learn domain knowledge, b) to learn control knowledge and c) to learn optimization knowledge.

Domain knowledge is utilized by planners in pre-processing phases in order to either modify the description of the problem in a way that it will make it easier for solving or make the appropriate adjustments to the planner to best attack the problem. Control knowledge can be utilized during search in order to either solve the problem faster or produce better plans. For example, the knowledge extracted from past examples can be used to refine the heuristic functions or create a guide for pruning non-promising branches. Most work on combining machine learning and planning in the past has focused on learning control knowledge since it is crucial for planners to have an informative guide during search. Finally, optimization knowledge is utilized after the generation of an initial plan, in order to transform it in a new one that optimizes certain criteria, e.g. number of steps or usage of resources.

Learning Domain Knowledge

OBSERVER (Wang, 1996) is a learning module built on top of the PRODIGY system that uses the hints and past knowledge of experts in order to extract and refine the full description of the

operators for a new domain. The description of the operators include negative, positive and conditional preconditions and effects. OBSERVER uses a multistrategy learning technique that combines learning by observing and refining through practice (learning by doing). Knoblock (1990) presented another learning module for PRODIGY, called ALPINE, that learns abstraction hierarchies and thus reduces the required search. ALPINE classifies the literals of the given planning problem, abstracts them and performs an analysis on the domain to aid ordering and combination of the abstractions.

MULTI-TAC (Minton, 1996) is a learning system that automatically fine tunes itself in order to synthesize the most appropriate constraint satisfaction program to solve a problem utilizing a library of heuristics and generic algorithms. The methodology we followed in this chapter for one of the adaptive systems (HAP_{RC}) presents some similarities with MULTI-TAC, since both approaches learn models that associate problem characteristics with the most appropriate setups for their solvers. The learned model of MULTI-TAC is a number of rules that are extracted using two complementary methods. The first one is analytic and employs meta-level theories in order to reason about the constraints, while the second one, which is based on the generate–and–test schema, extracts all possible rules and uses test problems in order to decide about their quality.

One of the few past approaches towards the direction of adaptive planning is the BUS system (Howe & Dahlman, 1993; Howe et al, 1999). BUS runs six state-of-the-art planners, namely STAN, IPP, SGP, BlackBox, UCPOP and PRODIGY, using a round-robin schema, until one of them finds a solution. BUS is adaptive in the sense of dynamically deciding the ordering of the six planners and the duration of the time slices based on the values of five problem characteristics and some rules extracted from the statistical analysis of past runs. The system

achieved more stable behaviour than all the individual planners but it was not as fast as one may have expected.

The authors have worked during the past few years in exploiting Machine Learning techniques for Adaptive Planning and have developed two systems that are described in detail later in this chapter. The first system, called HAP_{RC} (Vrakas et al, 2003a ; 2003b), is capable of automatically fine-tuning its planning parameters based on the morphology of the problem in hand. The tuning of HAP_{RC} is performed by a rule system, the knowledge of which has been induced through the application of a classification algorithm over a large dataset containing performance data of past executions of HAP (Highly Adjustable Planner). The second system, called HAP_{NN} (Tsoumakas et al, 2003), adopts a variation of the *k* Nearest Neighbour machine learning algorithm that enables the incremental enrichment of its knowledge and allows users to specify their level of importance on the criteria of plan quality and planning speed.

Learning Control Knowledge

The history of learning control knowledge for guiding planning systems, sometimes called *speedup learning*, dates back to the early 70's. The STRIPS planning system was soon enhanced with the MACROPS learning method (Fikes et. al, 1972) that analyzed past experience from solved problems in order to infer successful combinations of action sequences (macro-operators) and general conditions for their application. MACROPS was in fact the seed for a whole new learning methodology, called *Explanation-Based Learning* (EBL).

EBL belongs to the family of analytical learning methods that use prior knowledge and deductive reasoning to enhance the information provided by training examples. Although EBL encompasses a wide variety of methods, the main underlying principle is the same: The use of prior knowledge to analyze, or explain each training example in order to infer which example

features and constraints are relevant and which irrelevant to the learning task under consideration. This background knowledge must be correct and sufficient for EBL to generalize accurately. Planning problems offer such a correct and complete domain theory that can be readily used as prior knowledge in EBL systems. This apparently explains the very strong relationship of EBL and planning, as the largest scale attempts to apply EBL have addressed the problem of learning to control search. An overview of EBL computer programs and perspectives can be found in (Ellman, 1989).

The PRODIGY architecture (Carbonell et al, 1991; Veloso et al, 1995) was the main representative of control-knowledge learning systems. This architecture, supported by various learning modules, focuses on learning the necessary knowledge (rules) that guides a planner to decide what action to take next during plan execution. The system mainly uses EBL to explain fails and successes and generalize the knowledge in control rules that can be utilized in the future in order to select, reject or prefer choices. Since the overhead of testing the applicability of rules was quite large (utility problem) the system also adopted a mixed criterion of usability and cost for each rule in order to discard some of them and refine the rest. The integration of EBL into PRODIGY is detailed in (Minton, 1988).

Borrajo and Veloso (1996) developed HAMLET, another system combining planning and learning that was built on top of PRODIGY. HAMLET combines EBL and inductive learning in order to incrementally learn through experience. The main aspects responsible for the efficiency of the system were: the lazy explanation of successes, the incremental refinement of acquired knowledge and the lazy learning to override only the default behavior of the planner.

Another learning approach that has been applied on top of PRODIGY, is the STATIC algorithm (Etzioni, 1993), which used *Partial Evaluation* to automatically extract search-control knowledge from training examples. Partial Evaluation, a kind of program optimization method

used for PROLOG programs, bares strong resemblance to EBL. A discussion of their relationship is provided in (van Harmelen & Bundy, 1988).

DYNA-Q (Sutton, 1990) followed a *Reinforcement Learning* approach (Sutton & Barto, 1998). Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. DYNA-Q employed the Q-learning method, in order to accompany each pair of state-action with a reward (Q-value). The rewards maintained by DYNA-Q are incrementally updated as new problems are faced and are utilized during search as a means of heuristic function. The main problems faced by this approach were the very large memory requirements and the amount of experience needed for solving non-trivial problems.

A more recent approach of learning control knowledge for domain independent planning was presented by Martin and Geffner (2000). They focus on learning *generalized policies* that serve as heuristic functions, mapping states and goals into actions. In order to represent their policies they adopt a concept language, which allows the inference of more accurate models using less training examples. The learning approach followed in this project was a variation of Rivest's Decision Lists (1987), which is actually a generalization of other concept representation techniques, such as decision trees.

Eureka (Jones & Langley, 1995) adopts a flexible means-ends analysis for planning and is equiped with a learning module that performs *Analogical Reasoning* over stored solutions. The learning approach of Analogical Reasoning is based on the assumption that if two situations are known to be similar in some respects, it is likely that they will be similar in others. The standard computational model of reasoning by analogy defines the source of an analogy to be a problem solution, example, or theory that is relatively well understood. The target is not completely

understood. Analogy constructs a mapping between corresponding elements of the target and source. Analogical inferences extend this mapping to new elements of the target domain.

Eureka, actually maintains a long-term semantic network which stores representations of past situations along with the operators that led to them. The semantic network is constantly modified by either adding new experiences or updating the strength of the existing knowledge. Daedalus (Langley & Allen, 1993) is a similar system that uses a hierarchy of probabilistic concepts in order to summarize its knowledge. The learning module of Daedalus is quite complex and in a sense it unifies a large number of learning techniques including Decision Tree Construction, Rule Induction and EBL.

Another example of utilizing learning techniques for inferring control knowledge for automated planning systems is the family of planners that employ *Case-based Reasoning* (Kolodner, 1993). Case-based Eeasoning (CBR) is an instance-based learning method that deals with instances that are usually described by rich relational representations. Such instances are often called cases. In contrast to instance-based methods that perform a statistical computation of a distance metric based on numerical values, CBR systems must compute a complex similarity measure. Another distinctive feature of CBR is that the output for a new case might involve the combination of the output of several retrieved cases that match the description of the new case. The combination of past outputs might involve the employment of knowledge-based reasoning due to the rich representation of cases.

CBR is actually very related to analogical reasoning. Analogical reasoning provides the mechanism for mapping the output of an old case to an output for a new case. Cased-based reasoning was based on analogical reasoning but also provided a complete framework for dealing with issues like the representation of cases, strategies for organizing a memory of prior cases, retrieval of prior cases and the use of prior cases for dealing with new cases.

Two known case-based planning systems are CHEF (Hammond, 1989) and PRIAR (Kambhampati & Hendler, 1992). CHEF is one of the earliest case-based planners and used the Szechwan cooking as the application domain. CHEF used memory structures and indexes in order to store successful plans, failed plans and repairs among with general conditions allowing it to reuse past experience. PRIAR is a more general case-based system for plan modification and reuse that uses hierarchical non-linear planning, allowing abstraction and least-commitment.

Learning Optimization Knowledge

Ambite, Knoblock and Minton (2000) have presented an approach for learning Plan Rewriting Rules that can be utilized along with local search, in order to improve easy-to-generate low quality plans. In order to learn the rules, they obtain an optimal and a non-optimal solution for each problem in a training set, transform the solutions into graphs, and then extract and generalize the differences between each pair of graphs (optimal and non-optimal) and form rules in a manner similar to EBL.

IMPROVE (1998), deals with the improvement of large probabilistic plans in order to increase their probability of being successfully carried out by the executor. IMPROVE uses a simulator in order to obtain traces of the execution of large plans and then feeds these traces to a sequential discovery data mining algorithm in order to extract patterns that are common in failures but not in successes. Qualitative reasoning (Kuipers, 1994) is then applied in order to improve the plans.

Summary and Further Reading

Table 1 summarizes the 18 approaches that were presented in this Section. It shows the name of each system, the type of knowledge that was acquired, the way this knowledge was utilized and the learning techniques that were used for inducing it. Further information on the topic of

Machine Learning for Automated Planning can be found in the extended survey of Zimmerman and Kambhampati (2003) and also in (Gopal, 2000).

System	Knowledge	Utilization	Learning Techniques
OBSERVER	Domain	Refine problem definition	Learning by Observing, Refining via Practice
MULTI-TAC	Domain	Configure System	Meta-Level Theories, Generate and Test
ALPINE	Domain	Abstract the problem	Domain Analysis, Abstraction
BUS	Domain	Configure System	Statistical Analysis
HAP_{RC}	Domain	Configure System	Classification Rules
HAP_{NN}	Domain	Configure System	*k*NN
PRODIGY	Control	Search guide	EBL
HAMLET	Control	Search guide	EBL, Rule Learning
STATIC	Control	Search guide	Partial Evaluation
STRIPS	Control	Macro-operators	EBL
Generalized Policies	Control	Search guide	Decision Lists
DYNA-Q	Control	Heuristic	Reinforcement Learning
CHEF	Control	Canned plans	CBR
PRIAR	Control	Canned plans	CBR
EUREKA	Control	Search guide	Analogical Reasoning
DAEDALUS	Control	Search guide	Analogical Reasoning, Conceptual Clustering
Plan Rewriting	Optimization	Reduce plan size	EBL
IMPROVE	Optimization	Improve plan applicability	Sequential Patterns

Table 1. System name, type of knowledge, utilization and learning techniques

THE PLANNING PROBLEM

The rest of the chapter addresses learning domain knowledge for the automatic configuration of planning systems. The aim of this approach is to build an adaptive planning system that can automatically fine-tune its parameters based on the morphology of the problem in hand. This is a very important feature for planning systems, since it combines the efficiency of customized solutions with the generality of domain independent problem solving.

There are two main issues for investigation: a) what sort of customization should be performed on a domain-independent planner and b) how can the morphology of a planning problem be captured and quantified. These are addressed in the remaining of this section.

The Planning System

The planning system used as a test bed for our research is HAP (Highly Adjustable Planner), a domain-independent, state-space heuristic planning system, which can be customized through a

number of parameters. HAP is a general planning platform which integrates the search modules of the BP planner (Vrakas & Vlahavas, 2001), the heuristics of AcE (Vrakas & Vlahavas, 2002) and several techniques for speeding up the planning process. Apart from the selection of the planning direction, which is the most important feature of HAP, the user can also set the values of 6 other parameters that mainly affect the search strategy and the heuristic function. The seven parameters along with their value sets are outlined in Table 2.

Name	**Value Set**
Direction	{0,1}
Heuristic	{1,2,3}
Weights (w_1 and w_2)	{0,1,2,3}
Penalty	{10,100,500}
Agenda	{10,100,1000}
Equal_estimation	{0,1}
Remove	{0,1}

Table 2 The value sets for planning parameters

HAP is capable of planning in both directions (progression and regression). The system is quite symmetric and for each critical part of the planner, e.g. calculation of mutexes, discovery of goal orderings, computation of the heuristic, search strategies etc., there are implementations for both directions. The search *Direction* is the first adjustable parameter of HAP with the following values: a) 0 (Regression or Backward chaining) and b) 1 (Progression or Forward chaining). The planning direction is a very important factor for the efficiency of a planning system, since the best direction strongly depends on the morphology of the problem in hand and there is no easy answer which direction should be preferred.

The HAP system employs the heuristic function of the AcE planner, as well as two variations. Heuristic functions are implemented for both planning directions during the pre-planning phase by performing a relaxed search in the direction opposite to the one used in the search phase. The heuristic function computes estimations for the distances of all grounded

actions of the problem. The original heuristic function of the AcE planning system, is defined by the following formula:

$$dist(A) = \begin{cases} 1, & if\ \ prec(A) \subseteq I \\ 1 + \sum_{X \in MPS(prec(A))} dist(X), & if\ \ prec(A) \not\subset I \end{cases}$$

where *A* is the action under evaluation, *I* is the initial state of the problem and *MPS(S)* is a function returning a set of actions, with near minimum accumulated cost, achieving state *S*. The algorithm of *MPS* is outlined in Figure 1.

```
Function MPS(S)
Input: a set of facts S
Output: a set of actions achieving S with near minimum accumulated dist
═══════════════════════════════════════════════════════════════════
Set G = ∅
S = S – S ∩ I
Repeat
       f is the first fact in S
       Let act(f) be the set of actions achieving f
       for each action A in act(f) do
              val(A) = dist(A) / |add(A) ∩ S|

       Let A' be an action in act(f) that minimizes val
       Set G = G ∪ A'
       Set S = S – add(A') ∩ S
Until S = ∅
Return G
```

Figure 1. Function MPS(S)

Apart from the original AcE heuristic function described above, HAP embodies two more fined-grained variations. The general idea behind these variations lies in the fact that when we select a set of actions in order to achieve the preconditions of an action *A*, we also achieve several other facts (denoted as *implied(A)*), which are not mutually exclusive with the preconditions of *A*. Supposing that this set of actions was chosen in the plan before *A*, then after the application of *A*, the facts in *implied(A)* would exist in the new state, along with the ones in the add-list of *A*.

Taking all these into account, we produce a new list of facts for each action (named *enriched_add*) which is the union of the add-list and the implied list of this action.

The first variation of the AcE heuristic function uses the enriched instead of the traditional add-list in the MPS function in the second part of the function that updates state S. So the command $Set\ S = S - add(A') \cap S$ becomes $Set\ S = S - enriched\_add(A') \cap S$.

The second variation pushes the above ideas one step further. The *enriched_add* list is also used in the first part of the MPS function, which ranks the candidate actions. So, it additionally alters the command *val(A)=dist(A)/*$|add(A) \cap S|$ to *val(A)=dist(A)/* $|enriched\_add(A) \cap S|$.

The user may select the heuristic function to be used by the planner by configuring the *Heuristic* parameter. The acceptable values are three: a) 1 for the AcE heuristic, b) 2 for the first variation and c) 3 for the second variation.

Concerning search, HAP adopts a weighted A* strategy with two independent weights: w_1 for the estimated cost for reaching the final state and w_2 for the accumulated cost of reaching the current state from the starting state (initial or goals depending on the selected direction). In this work we have used four different assignments for the variable *weights* which correspond to different assignments for w_1 and w_2: a) 0 (w_1 =1, w_2 =0), b) 1 (w_1 =3, w_2 =1), c) 2 (w_1 =2, w_2 =1) and d) 3 (w_1 =1, w_2 =1). By selecting different value sets for the weights one can emulate a large number of search strategies such as *Best-First-Search* (w_1 =1, w_2 =0) or *Breadth-First-Search* (w_1 =0, w_2 =1). It is known that although certain search strategies perform better in general, the ideal treatment is to select the strategy which bests suits the morphology of the problem in hand.

The HAP system embodies two fact-ordering techniques (one for the initial state *I* and another one for the goals *G*), which try to find strong orderings in which the facts (of either *I* or

G) should be achieved. In order to find these orderings, the techniques make extensive use of mutual exclusions between facts, performing a limited search. These orderings are utilized during normal search phase, in order to identify possible violations. For each violation contained in a state, the estimated heuristic value of this state is increased by *Penalty*, a constant number supplied by the user. In this work we have tested the HAP system with three different values for *Penalty*: a) 10, b) 100 and c) 500. The reason for not being very strict with states containing violations of orderings, is the fact that sometimes the only path to the solution is through these states.

The HAP system allows the user to set an upper limit in the number of states in the planning agenda. This enables the planner to handle very large problems, since the memory requirements will not grow exponentially with the size of the problem. However, in order to keep a constant number of states in the agenda, the algorithm prunes branches, which are less likely to lead to a solution, and thus the algorithm cannot guarantee completeness. Therefore, it is obvious that the size of the planning agenda significantly affects the search strategy. For example, if we set Agenda to 1 and w_2 to 0, the search algorithm becomes pure Hill-Climbing, while by setting Agenda to larger values, w_1 to 1 and w_2 to 1 the search algorithm becomes A*. Generally, by increasing the size of the agenda we reduce the risk of not finding a solution, while by reducing the size of the agenda the search algorithm becomes faster and we ensure that the planner will not run out of memory. In this work we have used three different settings for the size of the agenda: a) 10, b) 100 and c) 1000.

Another parameter of HAP is *Equal_estimation* that defines the way in which states with the same estimated distances are treated. If *Equal_estimation* is set to 0 then when two states with the same value in the heuristic function exist, the one with the largest distance from the starting

state (number of actions applied so far) is preferred. If *Equal_estimation* is set to 1, then the search strategy will prefer the state that is closer to the starting state.

HAP also embodies a technique for simplifying the definition of the current sub-problem (current state and goals) during the search phase. This technique eliminates from the definition of the sub-problem all the goals that: a) have already been achieved in the current state and b) do not interfere with the achievement of the remaining goals. In order to do this, the technique performs a dependency analysis on the goals of the problem off-line, before the search process. Although the technique is very useful in general, the dependency analysis is not complete. In other words, there are cases where an already achieved sub-goal should be temporarily destroyed in order to continue with the achievement of the rest of the goals. Therefore, by removing this fact from the current state the algorithm may risk completeness. The parameter *Remove* can be used to turn on (value 1) or off (value 0) this feature of the planning system.

The parameters presented above are specific to the HAP system. However, the methodology presented in this chapter is general enough and can be applied to other systems as well. Most of the modern planning systems support or can be modified to support all or some of the parameterized aspects presented in this section. For example, there are systems such as the progression planner HSP (Bonet et. al, 1997) that were accompanied by versions working in the opposite directions; HSP-R (Bonet & Geffner, 1999) is a regression planner based on HSP.

Moreover, most of the planning systems presented during the last years can be customized through their own set of parameters. For example, the GRT planning system (Refanidis & Vlahavas, 2001) allows the user to customize the search strategy (Best-first or Hill-climbing) and to select how the goals of the problem are enriched (this affects the heuristic function). LPG (Gerevini et al, 2003) can be customized through a large number of planning parameters and could also be augmented using the proposed methodology. The user may select options such as

the heuristic function (there are two available), the search strategy, the number of restarts, the depth of the search, the way mutexes are calculated and others. The MIPS system (Edelkamp & Helmert, 2001) also allows some customization, since it uses a weighted A* search strategy, the weights of which can be set by the user, in a manner similar to HAP. Furthermore, the user can also set the optimization level.

Quantifying the structure of planning problems

Selecting a set of numerical attributes that represent the dynamics of problems and domains is probably the most important task in the process of building an adaptive planning system. These attributes should be able to group problems with similar structure and discriminate uneven ones. Moreover, these attributes should clearly influence specific choices for the values of the available planning parameters. Therefore, their selection strongly depends on the underlying planning system.

The result of a theoretical analysis on a) the morphology of problems, b) the way this is expressed through the PDDL language and c) the technology of the HAP planning system, was a set of 35 measurable characteristics that are presented in Table 3. In Table 3, *h(I)* refers to the number of steps needed to reach *I* (initial state) by regressing the goals, as estimated by the backward heuristic function. Similarly, *h(G)* refers to the number of steps needed to reach the goals by progressing the initial state, estimated by the forward heuristic function.

Our main concern was to select simple attributes that their values are easily calculated and not complex attributes that would cause a large overhead in the total planning time. Therefore, most of the attributes come directly from the PDDL input files and their values can be calculated during the standard parsing process. We have also included a small number of attributes closely related to specific features of the HAP planning system, such as the heuristics or the fact-ordering

techniques. In order to calculate the values of these attributes, the system must perform a limited search. However, the overhead is negligible compared to the total planning time.

Name	Description
A1	Percentage of dynamic facts in Initial state over total dynamic facts
A2	Percentage of static facts
A3	Percentage of goal facts over total dynamic facts
A4	Ratio between dynamic facts in Initial state and goal facts
A5	Average number of actions per dynamic fact
A6	Average number of facts per predicate
A7	Standard deviation of the number of facts per predicate
A8	Average number of actions per operator
A9	Standard deviation of the number of actions per operator
A10	Average number of mutexes per fact
A11	Standard deviation of the number of mutexes per fact
A12	Average number of actions requiring a fact
A13	Standard deviation of the number of actions requiring a fact
A14	Average number of actions adding a fact
A15	Standard deviation of the number of actions adding a fact
A16	Average number of actions deleting a fact
A17	Standard deviation of the number of actions deleting a fact
A18	Average ratio between the number of actions adding a fact and those deleting it
A19	Average number of facts per object
A20	Average number of actions per object
A21	Average number of objects per object class
A22	Standard deviation of the number of objects per object class
A23	Ratio between the actions requiring an initial fact and those adding a goal (Relaxed branching factors)
A24	Ratio between the branching factors for the two directions
A25	h(I)/h(G) [1st heuristic] - h(I)/h(G) [2nd heuristic]
A26	h(I)/h(G) [1st heuristic] - h(I)/h(G) [3rd heuristic]
A27	h(I)/h(G) [2nd heuristic] - h(I)/h(G) [3rd heuristic]
A28	Average number of goal orderings per goal
A29	Average number of initial orderings per initial fact
A30	Average distance of actions / h(G) [forward direction]
A31	Average distance of actions / h(I) [backward direction]
A32	a30/a31
A33	Percentage of standard deviation of the distance of actions over the average distance of actions [Forward direction]
A34	Percentage of standard deviation of the distance of actions over the average distance of actions [Backward direction]
A35	Heuristics deviation [a33/a34]

Table 3. Problem characteristics

A second concern was the fact that the attributes should be general enough to be applied to all domains. Furthermore, their values should not largely depend on the size of the problem, otherwise the knowledge learned from easy problems can not be efficiently applied to difficult

ones. For example, instead of using the number of mutexes (mutual exclusions between facts) in the problem, which is an attribute that strongly depends on the size of the problem (larger problems tend to have more mutexes), we divide it by the total number of dynamic facts (attribute A10) and this attribute (mutex density) identifies the complexity of the problem without taking into account whether it is a large problem or not. This is a general solution followed in all situations where a problem attribute depends nearly linearly on the size of the problem.

The attributes can be classified in three categories: The first category (attributes A01-A9, A12-A24) refer to simple and easily measured characteristics of planning problems that source directly from the input files (PDDL). The second category (attributes A10, A11, A28, A29) consists of more sophisticated features of modern planners, such as mutexes or orderings (between goals and initial facts). The last category (attributes A25-A27, A30-A35) contains attributes that can be instantiated only after the calculation of the heuristic functions and refer to them.

The attributes presented above aim at capturing the morphology of problems expressed in a quantifiable way. The most interesting aspects of planning problems according to this attribute set are: a) the size of the problem, which mainly refers to the dimensions of the search space, b) the complexity of the problem, c) the directionality of the problem that indicates the most appropriate search direction, and d) the heuristic that best suits the problem.

The first two categories, namely the size and the complexity, are general aspects of planning problems. The directionality is also a general aspect of planning problems that is additionally, of great importance to HAP, due to its bi-directional capabilities. The last category depends strongly on the HAP planning system, concerning the suitability of the heuristic functions for the problem in hand. Although the four aspects that the selection of attributes was based on are not enough to completely represent any given planning problem, they form a non trivial set that one can base

the setup of the planning parameters of HAP. Table 4 sketches the relation between the four problem aspects described above and the 35 problem attributes adopted by this work.

Attribute	Size	Complexity	Directionality	Heuristics
A1	•			
A2	•	•		
A3	•			
A4	•		•	
A5	•	•		
A6	•			
A7	•	•		
A8	•			
A9	•	•		
A10		•		•
A11		•		•
A12		•		
A13		•		
A14		•		
A15		•		
A16		•		
A17		•		
A18		•	•	
A19	•	•		
A20	•	•		
A21		•		
A22		•		
A23		•	•	
A24			•	
A25			•	•
A26			•	•
A27			•	•
A28		•	•	
A29		•	•	
A30				•
A31				•
A32			•	•
A33		•	•	•
A34		•	•	•
A35		•	•	•

Table 4. Relation between problem aspects and attributes

LEARNING APPROACHES

The aim of the application of learning techniques in planning is to find the hidden dependencies among the problem characteristics and the planning parameters. More specifically, we are interested in finding those combinations of problem attributes and planning parameters that guarantee good performance of the system. One way to do this is by experimenting with all

possible combinations of the values of 35 problem attributes and the 7 planning parameters and then process the collected data in order to learn from it. However, this is not tractable since most of the problem attributes have continuous value ranges and even by discretizing them it would require a tremendous number of value-combinations. Moreover, it would not be possible to find or create enough planning problems to cover all the cases (value combinations of attributes).

One solution to the problem presented above is to select a relatively large number of problems, uniformly distributed in a significant number of domains covering as many aspects of planning as possible. Then experiment with these problems, called training set, and all the possible setups of the planning system (864 in the case of HAP), record all the data (problem attributes, planner configuration and the results in terms of planning time and plan length) and try to learn from that. It is obvious that the selection of problems for the training set is the second crucial part of the whole process. In order to avoid the over fitting and the disorientation of the learned model the training set must be significantly large and uniformly distributed over a large and representative set of different domains.

After the collection of the data there are two important stages in the process of building the adaptive system: a) selecting and implementing an appropriate learning technique in order to extract the model and b) embedding the model in an integrated system that will automatically adapt to the problem in hand. Note however, that these steps cannot be viewed as separate tasks in all learning approaches.

The rest of the section addresses these issues and presents details concerning the development of two adaptive systems, namely HAP_{RC} and HAP_{NN}.

Data Preparation

A necessary initial step in most data mining applications is data preparation. In our case, the data were collected from the execution of HAP using all 864 parameter configurations on 30 problems from each of the 15 planning domains of Table 5. The process of collecting the data is sketched in Figure 2. The recorded data for each run contained the 35 problem attributes presented in Section 0, the 7 planner parameters presented in Section 0, the number of steps in the resulting plan and the required time for building it.

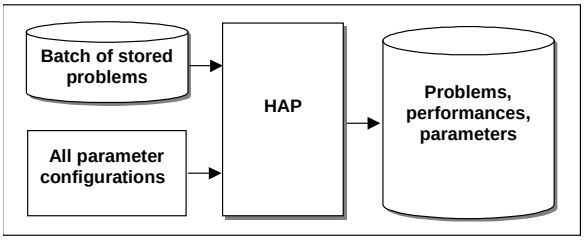

Figure 2. Preparing the training data

In the case where the planner did not manage to find a solution within the upper time limit of 60 seconds, a special value (999999) was recorded for both steps and time. This led to a dataset of 388.800 (450 problems * 864 configurations) records with 44 fields, the format of which is presented in Figure 3.

Domain	**Source**
Assembly	New domain
Blocks-world (3 operators)	Bibliography
Blocks-world (4 operators)	AIPS 98, 2000
Driver	AIPS 2002
Ferry	FF collection
Freecell	AIPS 2000, 2002
Gripper	AIPS 98
Hanoi	Bibliography
Sokoban	New domain
Logistics	AIPS 98, 2000
Miconic-10	AIPS 2000
Mystery	AIPS 98
Tsp	FF collection
Windows	New domain
Zeno	AIPS 2002

Table 5. Domains used for the creation of the learning data

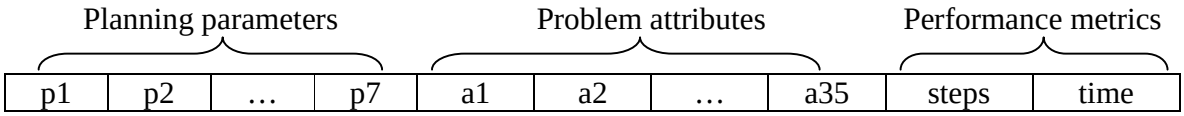

Figure 3. The format of the records

This dataset did not explicitly provide information on the quality of each run. Therefore, a data pre-processing stage was necessary that would decide about the performance of each configuration of HAP (for a given problem) based on the two performance metrics (number of plan steps and the required time for finding it). However, it is known within the planning community, that giving a solution quickly and finding a short plan are contradicting directives for a planning system. There were two choices in dealing with this problem: a) create two different models, one for fast planning and one for short plans, and then let the user decide which one to use or b) find a way to combine these two metrics and produce a single model which uses a trade-off between planning time and length of plans. We tested both scenarios and noticed that in the first one the outcome was a planner that would either create short plans after too long a time, or create awfully large plans quickly. Since none of these cases are acceptable in real-world situations, we decided to adopt the second scenario.

In order to combine the two metrics we first normalized the plan steps and planning time according to the following transformation:

- Let S_{ij} be the number of plan steps and T_{ij} be the required time to build it for problem i (i=1..450) and planner configuration j (j=1..864).
- We first found the shortest plan and minimum planning time for each problem among the tested planner configurations.

$$S_i^{\min} = \min_j(S_{ij}),\ T_i^{\min} = \min_j(T_{ij})$$

- We then normalized the results by dividing the minimum plan length and minimum planning time of each run with the corresponding problem value. For the cases where the planner could not find a solution within the time limits, the normalized values of steps and time were set to zero.
- $$S_{ij}^{norm} = \begin{cases} \frac{S_i^{\min}}{S_{ij}}, & S_{ij} \neq 999999 \\ 0, & otherwise \end{cases}, \quad T_{ij}^{norm} = \begin{cases} \frac{T_i^{\min}}{T_{ij}}, & T_{ij} \neq 999999 \\ 0, & otherwise \end{cases}$$
- We finally created a combined metric about plan attribute M_{ij}, which uses a weighted sum of the two normalized criteria:

  $$M_{ij} = w_s * S_{ij}^{norm} + w_t * T_{ij}^{norm}$$

Classification Rules

Learning sets of if-then rules is an appealing learning method, due to the easily understandable representation of rules by humans. There are various approaches to rule learning, including transforming decision trees to rules and using genetic algorithms to encode each rule set. We will here briefly describe another approach that is based on the idea of *Sequential Covering* that it has been exploited by a number of planning systems.

Sequential covering is a family of algorithms for learning rule sets based on the strategy of learning one rule, removing the data it covers, then iterating this process (Mitchell, 1997). The first rule will be learned based on all the available training examples. We then remove any positive examples covered by this rule and then invoke it again to learn a second rule based on the remaining training examples. It is called a sequential covering algorithm because it sequentially learns a set of rules that together cover the full set of positive examples. The final set of rules can then be sorted so that more accurate rules will be considered first when a new instance must be classified.

Learning a rule usually involves performing a heuristic search in the space of potential attribute-value pairs to be added to the current rule. Depending on the strategy of this search and the performance measure used for guiding the heuristic search several variations of sequential covering have been developed.

The CN2 program (Clark & Niblett, 1989) employs a general to specific beam search through the space of possible rules in search of a rule with high accuracy, though perhaps incomplete coverage of the data. Beam search is a greedy non-backtracking search strategy in which the algorithm maintains a list of the k best candidates at each step, rather than a single best candidate. On each search step, specializations are generated for each of these k best candidates, and the resulting set is again reduced to the k most promising members. A measure of entropy is the heuristic guiding the search.

AQ (Michalski et al, 1986) also conducts a general-to-specific beam-search for each rule, but uses a single positive example to focus this search. In particular, it considers only those attributes satisfied by the positive example as it searches for progressively more specific hypotheses. Each time it learns a new rule it selects a new positive example from those that are not yet covered, to act as a seed to guide the search for this new disjunct.

IREP (Furnkranz & Widmer, 1994), RIPPER (Cohen, 1995) and SLIPPER (Cohen & Singer, 1999) are three rule learning systems that are based on the same framework but use reduced error pruning to prune the antecedents of each discovered rule. IREP was a first algorithm that employed reduced-error pruning. RIPPER is an enhanced version of the IREP approach dealing with several limitations of IREP and producing rules of higher accuracy. SLIPPER extends RIPPER by using confidence-rated boosting and manages to achieve even better accuracy.

Classifying executions

In order to learn classification rules from the dataset, a necessary step was to decide for the two classes (good run or bad run) based on the value of the combined quality metric M_{ij}. Therefore, we split the records of the training data into two categories: a) the class of good runs consisting of the records for which M_{ij} was larger than a threshold and b) the class of bad runs consisting of the remaining records. In order to create these two sets of records, we calculated the value Q_{ij} for each run, which is given by the following formula:

$$Q_{ij} = \begin{cases} good, & M_{ij} > c \\ bad, & M_{ij} \leq c \end{cases}$$

where *c*, is the threshold constant controlling the quality of the good runs. For the M_{ij} metric, we used the value of 1 for both w_s and w_t in order to keep the balance between the two quality criteria.

For example, for *c* equal to 1.6 the above equation means that "*a plan is good if its combined steps and time are at most 40% worse (bigger) than the combined minimum plan steps and time for the same problem*". Since normalized steps and time are combined with a 1:1 ratio, the above 40% limit could also be interpreted as an average of 20% for each steps and time. This is a flexible definition that would allow a plan to be characterized as good even if its steps are for example 25% worse than the minimum steps as long as its time is at most 15% worse than the minimum time, provided that their combination is at most 40% worse than the combined minimum steps and time. In the general case the combined steps and time must be at most (2-*c*)*100% worse than the combined minimum steps and time. After experimenting with various values for *c* we ended up that 1.6 was the best value to be adopted for the experiments.

Modeling

The next step was to apply a suitable machine learning algorithm in order to discover a model of the dependencies between problem characteristics, planner parameters and good planning performance. A first requirement was the interpretability of the resulting model, so that the acquired knowledge would be transparent and open to the inquiries of a planning expert. Apart from developing an adaptive planner with good performance to any given planning problem, we were also interested in studying the resulting model for interesting new knowledge and justifications for its performance. Therefore, symbolic learning approaches were at the top of our list.

Mining association rules from the resulting dataset was a first idea, which however was turned down due to the fact that it would produce too many rules making it extremely difficult to produce all the relevant ones. In our previous work (Vrakas et al, 2003a), we have used the approach of classification based on association rules (Liu, Hsu & Ma, 1998), which induces association rules that only have a specific target attribute on the right hand side. However, such an approach was proved inappropriate for our current much more extended dataset.

We therefore turned towards classification rule learning approaches, and specifically decided to use the SLIPPER rule learning system (Cohen & Singer, 1999) which is fast, robust, easy to use, and its hypotheses are compact and easy to understand. SLIPPER generates rule sets by repeatedly boosting a simple, greedy rule learner. This learner splits the training data, grows a single rule using one subset of the data and then prunes the rule using the other subset. The metrics that guide the growing and pruning of rules is based on the formal analysis of boosting algorithms. The implementation of SLIPPER that we used handles only two-class classification problems. This suited fine our two-class problem of "good" and "bad" performance. The output

of SLIPPER is a set of rules predicting one of the classes and a default rule predicting the other one, which is engaged when no other rule satisfies the example to be classified. We run SLIPPER so that the rule set predicts the class of "good" performance.

The Rule-Based Planner Tuner

The next step was to embed the learned rules in HAP as a rule-based system that decides the optimal configuration of planning parameters based on the characteristics of a given problem. In order to perform this task certain issues had to be addressed:

a. Should all the rules be included?

The rules that could actually be used for adaptive planning are those that associate, at the same time, problem characteristics, planning parameters and the quality field. So, the first step was to filter out the rules that included only problem characteristics as their antecedents. This process filtered out 21 rules from the initial set of 79 rules. We notice here that there were no rules including only planning parameters. If such rules existed, then this would mean that certain parameter values are good regardless of the problem and that the corresponding parameters should be fixed in the planner.

The remaining 58 rules that model good performance, were subsequently transformed so that only the attributes concerning problem characteristics remained as antecedents and the planning parameters were moved to the right-hand side of the rule as conclusions, replacing the rule quality attribute. In this way, a rule decides one or more planning parameters based on one or more problem characteristics.

What conflict resolution strategy should be adopted for firing the rules?

Each rule was accompanied by a confidence metric, indicating how valid a rule is, i.e. what percentage of the relevant data in the condition confirms the conclusion-action of the rule. A 100% confidence indicates that it is absolutely certain that when the condition is met, then the action should be taken.

The performance of the rule-based system is one concern, but it occupies only a tiny fragment of the planning procedure, therefore it is not of primary concern. That is why the conflict resolution strategy used in our rule-based system is based on the total ordering of rules according to the confidence factor, in descending order. This decision was based on our primary concern to use the most certain (confident) rules for configuring the planner, because these rules will most likely lead to a better planning performance.

Rules are appropriately encoded so that when a rule fires and sets one or more parameters, then all the other rules that might also set one (or more) of these parameters to a different setting are "disabled". In this way, each parameter is set by the most confident rule (examined first), while the rest of the rules that might affect this parameter are skipped.

What should we do with parameters not affected by the rule system?

The experiments with the system showed that on average the rule based system would affect approximately 4 planning parameters, leaving at the same time 3 parameters unset. According to the knowledge model, if a parameter is left unset, its value should not affect the performance of the planning system. However, since the model is not complete, this behavior could also be interpreted as an inability of the learning process to extract a rule for the specific case. In order to deal with this problem we performed a statistical analysis in order to find the best default settings for each independent parameter.

For dealing with situations where the rule-based systems leaves all parameters unset we calculated the average normalized steps and time for each planner configuration:

$$S_j^{avg} = \frac{\sum_i S_{ij}^{norm}}{\sum_i 1},\ T_j^{avg} = \frac{\sum_i T_{ii}^{norm}}{\sum_i 1}$$

and recorded the configuration with the best sum of the above metrics, which can be seen in Table 6.

For dealing with situations where the rule system could only set part of the parameters, but not all of them, we repeated the above calculations for each planner parameter individually, in order to find out if there is a relationship between individual settings and planner performance. Again for each parameter we recorded the value with the best sum of the average normalized steps and time. These settings are illustrated in Table 6.

Name	**Best Configuration**	**Best Individual Value**
Direction	0	0
Heuristic	1	1
Weights (w_1 and w_2)	2	2
Penalty	10	100
Agenda	100	10
Equal_estimation	1	1
Remove	0	1

Table 6: Best combined and individual values of parameters

In the future we will explore the possibility to utilize learned rules that predict bad performance as integrity constraints that guide the selection of the unset planner parameters in order to avoid inappropriate configurations.

The rule configurable version of HAP, which is outlined in Figure 4 contains two additional modules, compared to the manually configurable version of the system, that are run in a pre-planning phase. The first module, noted as *Problem Analyzer*, uses the problem's representation, constructed by the *Parser*, to calculate the values of the 35 problem characteristics used by the

rules. These values are then passed to the *Rule System* module, which tunes the planning parameters based on the embedded rule base and the default values for unset parameters. The values of the planning parameters along with the problem's representation are then passed to the planning module, in order to solve the problem.

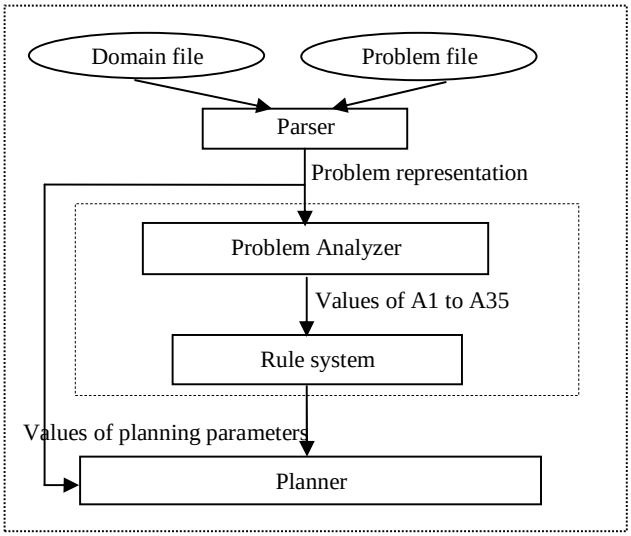

Figure 4. HAP_{RC} Architecture

k Nearest Neighbors

Apart from the rule-based approaches, we also experimented with other learning methodologies, mainly in order to overcome several limitations of the former. A very interesting learning approach, which could be easily adapted to our problem, is the k Nearest Neighbors (kNN) algorithm. According to this approach, when the planner is faced with a new problem, it identifies the k nearest instances from the set of training problems, aggregates the performance results for the different planner configurations and selects the one with the best average performance.

This is the most basic instance-based learning method for numerical examples. The nearest neighbors of an instance are defined in terms of some distance measure for the vectors of values of the examples. Considering the following instance x, that is described by the attributes:

$$\langle \alpha_1(x), \alpha_1(x), ..., \alpha_n(x) \rangle$$

where $\alpha_r(x)$ denotes the value of the instance for the *r*th attribute. Then the distance d of two instances x1, x2 can be measured using any suitable L norm:

$$d(x_i, x_j) = \sqrt[L]{\sum_{r=1}^{n} \left| a_r(x_i) - a_r(x_j) \right|^L}$$

For L=1 we get the Manhattan distance, while for L=2 we get the Euclidean distance.

When a new instance requires classification, the k nearest neighbor approach first retrieves the k nearest instances to this one. Then it selects the classification that most of these instances propose.

Preparing the Training Data

According to the methodology previously described, the system needs to store two kinds of information: a) the values for the 35 attributes for each one of the 450 problems in the training set in order to identify the k closest problems to a new one and b) the performance (steps and time) of each one of the 864 planner configurations for each problem in order to aggregate the performance of the k problems and then find the best configuration.

The required data were initially in the flat file produced by the preparation process described in a previous section. However, they were later organized as a multi-relational data set, consisting of 2 primary tables, *problems* (450 rows) and *parameters* (864 rows), and a relation table *performances* (450*864 rows), in order to save storage space and enhance the search for the *k* nearest neighbors and the retrieval of the corresponding performances. The tables were

implemented as binary files, with the *performances* table being sorted on both the problem id and the parameter id.

Online Planning Mode

Given a new planning problem, HAP_{NN} first calculates the values of the problem characteristics. Then the *k*NN algorithm is engaged in order to retrieve the *ids* of the *k* nearest problems from the *problems* file. The number of neighbors, *k*, is a user-defined parameter of the planner. In the implementation of *k*NN we use the Euclidean distance measure with the normalized values of the problem attributes to calculate the nearest problem.

Using the retrieved *ids* and taking advantage of the sorted binary file, HAP_{NN} promptly retrieves the performances for all possible configurations in a *k**864 two-dimensional matrix. The next step is to combine these performances in order to suggest a single parameter configuration with the optimal performance, based on past experience of the *k* nearest problems. The optimal performance for each problem is calculated using the M_{ij} criterion, where the two weights w_s and w_t are set by the user.

We can consider the final *k**864 2-dimensional matrix as a classifier combination problem, consisting of *k* classifiers and 864 classes. We can combine the decisions of the *k* classifiers, using the average Bayes rule, which essentially comes down to averaging the planner scores across the *k* nearest problems and selecting the decision with the largest average. Thus, the parameter configuration *j* (*j*=1..864) with the largest *C* is the one that is proposed and used.

$$C_j = \frac{1}{k}\sum_{i=1}^{k} M_{ij}$$

The whole process for the online planning mode of HAP_{NN} is depicted in Figure 5. It is worth noting that HAP_{NN} actually outputs an ordering of all parameter configurations and not just

one parameter configuration. This can be exploited for example in order to output the top 10 configurations and let the user decide amongst them. Another useful aspect of the ordering, is that when the first parameter configuration fails to solve the problem within certain time, then the second best could be tried. Another interesting alternative in such a case is the change of the weight setting so that time has a bigger weight. The effect of the weights in the resulting performance is empirically explored in the experimental results section that follows.

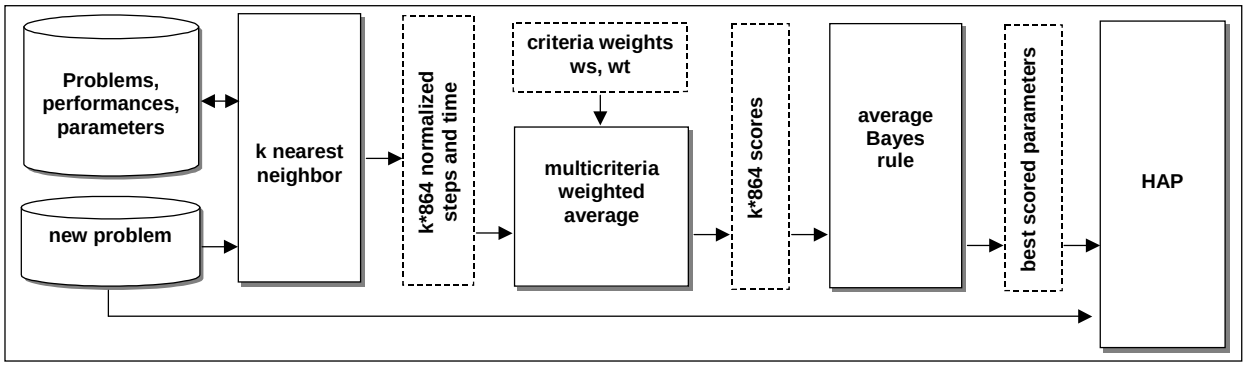

Figure 5. Online planning mode

Offline Incremental Training Mode

HAP_{NN} can be trained incrementally with each new planning problem that arises. Specifically, the planner stores each new examined planning problem, so that it can later train from it offline. As in the training data preparation phase, training consists of running the HAP planner on the batch of newly stored problems using all 864 value combinations of the 7 parameters. For each run, the features of the problem, the performance of the planner (steps of the resulting plan and required planning time) and the configuration of parameters are recorded.

The incremental training capability is an important feature of HAP_{NN}, stemming from the use of the *k*NN algorithm. As the generalization of the algorithm is postponed for the online phase, learning actually consists of just storing past experience. This is an incremental process

that makes it possible to constantly enhance the performance of the adaptive planner with the advent of new problems.

EXPERIMENTAL RESULTS

We have conducted four sets of comprehensive experiments in order to evaluate the potential gain in performance offered by the adaptive way in which the planner parameters are configured and to compare the two different approaches (rule-based and *k*NN). For the experiments presented below we used HAP_{NN} with the value of *k* set to 7.

All the runs of the planning systems (static and adaptive), including those used in the statistical analysis and the machine learning process, were performed on a SUN Enterprise Server 450 with 4 ULTRA-2 processors at 400 MHz and 2 GB of shared memory. The Operating system of the computer was SUN Solaris 8. For all experiments we counted CPU clocks and we had an upper limit of 60 sec, beyond which the planner would stop and report that the problem is not solved.

Adapting to problems of known domains

This experiment aimed at evaluating the generalization of the adaptive planners' knowledge to new problems from domains that have already been used for learning. Examining this learning problem from the viewpoint of a machine learner we notice that it is quite a hard problem. Its multi-relational nature (problem characteristics and planner parameters) resulted in a large dataset, but the number of available problems (450) was small, especially compared to the number of problem attributes (35). This gives rise to two problems with respect to the evaluation of the planners: a) Since the training data is limited (450 problems), a proper strategy must be followed for evaluating the planners' performance, b) evaluating on already seen examples must definitely be avoided, because it will lead to rather optimistic results due to overfitting.

For the above reasons we decided to perform 10-fold cross-validation. We have split the original data into 10 cross-validation sets, each one containing 45 problems (3 from each of the 15 domains). Then we repeated the following experiment 10 times: In each run, one of the cross-validation sets was withheld for testing and the 9 rest were merged into a training set. The training set was used for learning the models of HAP_{RC} and HAP_{NN} and the test set for measuring their performance. Specifically, we calculated the sum of the average normalized steps and time. In addition we calculated the same metric for the best static configuration based on statistical analysis of the training data (HAP_{MC}), in order to calculate the gain in performance. Finally, we calculated the same metric for the best configuration for any given problem (HAP_{ORACLE}) in order to compare with the maximum performance that the planners could achieve if it had an oracle predicting the best configuration. The results of each run were averaged and thus a proper estimation was obtained, which is presented in Table 7.

Fold	**HAP_{MC}**	**HAP_{ORACLE}**	**HAP_{RC}**	**HAP_{NN}**
1	1,45	1,92	1,60	1,74
2	1,63	1,94	1,70	1,73
3	1,52	1,94	1,60	1,70
4	1,60	1,94	1,70	1,75
5	1,62	1,92	1,67	1,73
6	1,66	1,92	1,67	1,76
7	1,48	1,91	1,69	1,72
8	1,47	1,91	1,57	1,74
9	1,33	1,91	1,47	1,59
10	1,43	1,92	1,65	1,73
Average	**1,52**	**1,92**	**1,63**	**1,72**

Table 7. Comparative results for adapting to problems of known domains

Studying the results of Table 7 we notice that both adaptive versions of HAP significantly outperformed HAP_{MC}. The difference in the performance between HAP_{RC} and HAP_{MC} was 0.11 on average, which can be translated as a 7% average gain combining both steps and time. HAP_{NN} performed even better, scoring on average 0.2 more (13% gain) than the static version. Moreover,

the auto-configurable versions outperformed the static one in all folds, exhibiting a consistently good performance. This shows that the learning methodologies we followed were fruitful and resulted in models that successfully adapt HAP to unknown problems of known domains.

Adapting to problems of unknown domains

The second experiment aimed at evaluating the generalization of the adaptive planners' knowledge to problems of new domains that have not been used for learning before. In a sense this would give an estimation for the behavior of the planner when confronted with a previously unknown problem of a new domain.

This is an even harder learning problem considering the fact that there are very few domains that have been used for learning (15), especially compared again to the 35 problem attributes. To evaluate the performances of HAP_{RC} and HAP_{NN} we used leave-one-(domain)-out cross-validation. We split the original data into 15 cross-validation sets, each one containing the problems of a different domain. Then we repeated the following experiment 15 times: In each run, one of the cross-validation sets was withheld for testing and the 14 rest were merged into a training set. As in the previous experiment, the training set was used for learning the models and the test set for measuring its performance.

The results show that all the planners performed worse than the previous experiment. Still HAP_{RC} and HAP_{NN} managed to increase the performance over HAP_{MC}, as it can be seen in Table 8.

We notice a 3% average gain of HAP_{RC} and 2% average gain of HAP_{NN} over the static version in the combined metric. This is a small increase in performance, but it is still a success considering that there were only 15 domains available for training. The enrichment of data from

more domains will definitely increase the accuracy of the models, resulting in a corresponding increase in the performance of the adaptive systems.

Test Domain	HAP_{MC}	HAP_{ORACLE}	HAP_{RC}	HAP_{NN}
Assembly	1,31	1,89	1,46	1,08
Blocks	1,13	1,98	1,10	1,77
Blocks_3op	1,69	1,99	1,52	1,81
Driver	1,52	1,92	1,49	1,45
Ferry	1,03	2,00	1,66	1,41
Freecell	1,43	1,96	1,39	1,70
Gripper	1,75	1,99	1,62	1,61
Hanoi	1,08	1,87	1,03	1,10
Logistics	1,66	1,91	1,69	1,75
Miconic	1,79	1,96	1,71	1,07
Mystery	1,21	1,97	1,11	0,88
Sokoban	1,20	1,96	1,57	1,45
Tsp	1,56	1,74	1,56	1,29
Windows	1,30	1,78	1,26	1,55
Zeno	1,26	1,93	1,34	1,35
Average	**1,39**	**1,92**	**1,43**	**1,42**

Table 8. Comparative results for adapting to problems of unknown domains

Scalability of the methodology

The third experiment aimed at showing the ability of the adaptive systems to learn from easy problems (problems that require little time to be solved) and to use the acquired knowledge as a guide for difficult problems. It is obvious that such a behavior would be very useful, since according to the methodology, each problem in the training set must be attacked with every possible combination of the planner's parameters and for hard problems this process may take enormous amounts of time.

In order to test the scalability of the methodology, we have split the initial data set into two sets: a) the training set containing the data for the 20 easiest problems from each domain and b) the test set containing the 10 hardest problems from each domain. The metric used for the discrimination between hard and easy problems was the average time needed by the 864 different

planner setups to solve the problem. We then used the training set in order to learn the models and statistically find the best static configuration of HAP and tested the two adaptive planners and HAP_{MC} on the problems of the test set. For each problem we have also calculated the performance of HAP_{ORACLE} in order to show the maximum performance that could have been achieved by the planner.

The results of the experiments, which are presented in Table 9, are quite impressive. The rule based version managed to outperform the best static version in 11 out of the 15 domains and its performance was approximately 40% better on average. Similarly HAP_{NN} was better in 11 domains too and the average gain was approximately 33%. There are some very interesting conclusions that can be drawn from the results:

- With the exception of a small number of domains, the static configurations which are effective for easy problems do not perform well for the harder instances of the same domains.
- There are some domains (e.g. Hanoi) where there must be great differences between the morphology of easy and hard problems and therefore neither the statistical nor the learning analyses can effectively scale up.
- It is clear that some domains present particularities in their structure and it is quite difficult to tackle them without any specific knowledge. For example, in *Freecell* all the planners and specifically HAP_{RC} and HAP_{MC} that were trained from the rest of the domains only, did not perform very well (see Table 8), while the inclusion of Freecell's problems in their training set, gave them a boost (see Table 9).
- There are domains where there is a clear trade-off between short plans and little planning time. For example, the low performance of HAP_{ORACLE} in the Tsp domain shows that the

configurations that result in short plans require a lot of planning time and the ones that solve the problems quickly produce bad plans.

- The proposed learning paradigms can scale up very well and the main reason for this is the general nature of the selected problem attributes.

Test Domain	**HAP_{MC}**	**HAP_{ORACLE}**	**HAP_{RC}**	**HAP_{NN}**
Assembly	0,91	1,86	1,64	1,80
Blocks	0,91	1,86	1,64	1,72
Blocks_3op	1,86	1,98	1,72	1,86
Driver	1,22	1,92	1,72	1,51
Ferry	0,31	2,00	1,89	1,85
Freecell	1,86	1,96	1,87	1,84
Gripper	1,68	1,99	1,76	1,96
Hanoi	0,45	1,80	1,19	0,50
Logistics	1,68	1,87	1,80	1,81
Miconic	1,93	1,96	1,93	1,93
Mystery	0,67	1,94	1,73	1,52
Sokoban	0,79	1,92	1,66	1,47
Tsp	1,35	1,54	1,32	1,46
Windows	1,52	1,65	1,49	1,42
Zeno	0,89	1,91	1,77	1,29
Average	**1,20**	**1,88**	**1,68**	**1,60**

Table 9. Scalability of the methodology

Ability to learn a specific domain

The fourth experiment aimed at comparing general models, which have been learned from a variety of domains versus specific models that have been learned from problems of a specific domain. The reason for such an experiment is to have a clear answer to the question whether the planning system could be adapted to a target domain by using problems of solely this domain. The rationale behind this is that a general-purpose domain independent planner can be used without having to hand code it in order to suit the specific domain. Furthermore, the experiment can also show how disorienting can the knowledge from other domains be.

In order to carry out this experiment, we created 15 train sets, each one containing the 20 easiest problems of a specific domain and 15 test sets with the 10 hardest instances. The next step was to learn specific models for each domain, and test them on the hardest problems of the same domain. For each domain we compared the performance of the specialized models versus the performance of general models, which have been trained from the 20 easier problems from all 15 domains (see previous subsection). The results from the experiment are presented in Table 10, where:

- HAP_{MC} corresponds to the manually configured version according to the statistical analysis on the 20 easy problems of each domain,
- specific HAP_{RC} and HAP_{NN} correspond to the adaptive versions trained only from the 20 easier problems of each domain,
- general HAP_{RC} and HAP_{NN} correspond to the adaptive versions trained from the 300 problems (20 easier problems from each one of the 15 domains) and
- HAP_{Oracle} corresponds to the ideal configuration.

According to the results presented in Table 10, HAP_{RC} outperforms the best static one in 13 out of the 15 domains and on average it is approximately 7% better. This shows that we can also induce efficient models that perform well in difficult problems of a given domain when solely trained on easy problems of this domain. However, this is not the case for HAP_{NN}, whose not very good performance indicates that the methodology requires more training data, especially because there is a large number of attributes.

Comparing the specialized models of HAP_{RC} with the general ones, we see that it is on average 4% better. This shows that in order to adapt to a single domain, it is better to train the planner exclusively from problems of that domain, although such an approach would

compromise the generality of the adaptive planner. The results also indicate that on average there is no actual difference between the performance of the general and the specific versions of HAP_{NN}. To some extend this behavior is reasonable and can be justified by the fact that most of the nearest neighbors of each problem belong to the same domain and no matter how many redundant problems are included in the training set, the algorithm will select the same problems in order to learn the model.

			HAP_{RC}		**HAP_{NN}**	
Test Domain	**HAP_{MC}**	**HAP_{ORACLE}**	specific	general	specific	general
Assembly	1,68	1,86	1,72	1,64	1,84	1,80
Blocks	1,68	1,86	1,74	1,64	1,64	1,72
Blocks_3op	1,85	1,98	1,88	1,72	1,89	1,86
Driver	1,68	1,92	1,78	1,72	1,22	1,51
Ferry	1,83	2,00	1,85	1,89	1,85	1,85
Freecell	1,88	1,96	1,85	1,87	1,84	1,84
Gripper	1,66	1,99	1,78	1,76	1,96	1,96
Hanoi	1,00	1,80	1,38	1,19	0,50	0,50
Logistics	1,80	1,87	1,81	1,80	1,81	1,81
Miconic	1,93	1,97	1,93	1,93	1,93	1,93
Mystery	1,65	1,94	1,83	1,73	1,52	1,52
Sokoban	1,61	1,92	1,88	1,66	1,57	1,47
Tsp	1,36	1,54	1,38	1,32	1,46	1,46
Windows	1,35	1,65	1,48	1,49	1,46	1,42
Zeno	1,43	1,91	1,80	1,78	1,44	1,29
Average	**1,63**	**1,88**	**1,74**	**1,68**	**1,60**	**1,60**

Table 10. General vs. specialized models

DISCUSSION AND CONCLUSION

This chapter presented our research work in the area of using Machine Learning techniques in order to infer and utilize domain knowledge in Automated Planning. The work consisted of two different approaches: The first one utilizes classification rules learning and a rule-based system and the second one uses a variation of the k-Nearest Neighbors learning paradigm.

In the first approach the learned knowledge consists of rules that associate specific values or value ranges of measurable problem attributes with the best values for one or more planning

parameters, such as the direction of search or the heuristic function. The knowledge is learned offline and it is embedded in a rule system, which is utilized by the planner in a pre-processing phase in order to decide for the best setup of the planner according to the values of the given problem attributes.

The second approach is also concerned with the automatic configuration of planning systems in a pre-processing phase, but the learning is performed on-line. More specifically, when the system is confronted with a new problem, it identifies the *k* nearest instances from a database of solved problems and aggregates the planner setups that resulted in the best solutions according to the criteria imposed by the user.

The model of the first approach is very compact and it consists of a relatively small number (less than 100) of rules that can be easily implemented in the adaptive system. Since the size of the model is small it can be easily consulted for every new problem and the overhead imposed to the total planning time is negligible. However, the inference of the model is a complicated task that involves many subtasks and requires a significant amount of processing time. Therefore, the model cannot be easily updated with new problems. Furthermore, if the user wishes to change the way the solutions are evaluated (e.g. emphasizing more on plan size) this would require rebuilding the whole model.

On the other hand, the model of the k Nearest Problems approach is inferred on-line every time the system is faced with a new problem. The data that are stored in the database of the system are in raw format and this allows incremental expansion and easy update. Furthermore, each run is evaluated on-line and the weights of the performance criteria (e.g. planning time or plan size) can be set by the user. However, since the system maintains raw data for all the past runs, it requires a significant amount of disk size which increases as new problems are added in

the database. Moreover, the overhead imposed by the processing of data may be significant, especially for large numbers of training problems.

Therefore, the decision on which method to follow strongly depends on the application domain. For example, if the planner is used as a consulting software for creating large plans, e.g. for logistics companies, then neither the size requirements or the few seconds overhead of the k Nearest Problems would be a problem. On the other hand, if the planner must be implemented as a guiding system on a robot with limited memory then the rule based model would be more appropriate.

According to the experimental results, both systems have exhibited promising performance that is on average better than the performance of any statistically found static configuration. The speedup improves significantly when the system is tested on unseen problems of known domains, even when the knowledge was induced by far easier problems than the tested ones. Such a behavior can prove very useful in customizing domain independent planners for specific domains using only a small number of easy-to-solve problems for training, when it cannot be afforded to reprogram the planning system.

The speedup of our approach compared to the statistically found best configuration can be attributed to the fact that it treats planner parameters as associations of the problem characteristics, whereas the statistical analysis tries to associate planner performance with planner settings, ignoring the problem morphology.

In the future, we plan to expand the application of Machine Learning to include more measurable problem characteristics in order to come up with vectors of values that represent the problems in a unique way and manage to capture all the hidden dynamics. We also plan to add more configurable parameters of planning, such as parameters for time and resource handling and enrich the HAP system with other heuristics from state-of-the-art planning systems. Moreover, it

is in our direct plans to apply learning techniques to other planning systems, in order to test the generality of the proposed methodology.

In addition, we will explore the applicability of different rule-learning algorithms, such as decision-tree learning that could potentially provide knowledge of better quality. We will also investigate the use of alternative automatic feature selection techniques that could prune the vector of input attributes thus giving the learning algorithm the ability to achieve better results. The interpretability of the resulting model and its analysis by planning experts will also be a point of greater focus in the future.

REFERENCES

Ambite, J. L., Knoblock, C., & Minton, S. (2000). Learning Plan Rewriting Rules. Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling, 3-12.

Bonet, B., and Geffner, H. (1999). Planning as Heuristic Search: New Results, Proceedings of the 5th European Conference on Planning, 360-372.

Bonet, B., Loerincs, G., and Geffner, H. (1997). A robust and fast action selection mechanism for planning. Proceedings of the 14th International Conference of AAAI, 714-719.

Borrajo, D., & Veloso, M. (1996). Lazy Incremental Learning of Control Knowledge for Efficiently Obtaining Quality Plans. Artificial Intelligence Review. 10, 1-34.

Carbonell, J. G. (1983). Learning by Analogy: Formulating and generalizing plans from past experience. Machine Learning: An Artificial Intelligence Approach. Tioga Press, 137-162.

Carbonell, J., Knoblock, C. & Minton, S. (1991). PRODIGY: An integrated architecture for planning and learning, Architectures for Intelligence. Lawrence Erlbaum Associates, 241-278.

Cardie, C. (1994). Using decision trees to improve case-based learning. Proceedings of the 10th International Conference on Machine Learning, 28-36.

Clark, P. & Niblett, R. (1989). The CN2 induction algorithm. Machine Learning. 3(4), 261-284.

Cohen, W. & Singer Y. (1999). A Simple, Fast, and Effective Rule Learner, Proceedings of the 16th Conference of AAAI, 335-342.

Cohen, W. (1995). Fast Effective Rule Induction, Proceedings of the 12th International Conference on Machine Learning, 115-123.

Ellman, T. (1989). Explanation-based learning: A survey of programs and perspectives. Computing Surveys. 21(2), 163-221.

Edelkamp, S., & Helmert, M. (2001). The Model Checking Integrated Planning System. AI-Magazine. Fall, 67-71.

Etzioni, O. (1993). Acquiring Search-Control Knowledge via Static Analysis. Artificial Intelligence. 62 (2). 265-301.

Fikes, R., Hart, P., & Nilsson, N. (1972). Learning and Executing Generalized Robot Plans. Artificial Intelligence. 3, 251-288.

Furnkranz J. & Widmer G. (1994). Incremental reduced error pruning. Proceedings of the 11th International Conference on Machine Learning, 70-77.

Gerevini, A., Saetti, A. & Serina, I. (2003). Planning through Stochastic Local Search and Temporal Action Graphs. Journal of Artificial Intelligence Research. 20, 239-290.

Gopal, K. (2000). An Adaptive Planner based on Learning of Planning Performance. Master Thesis, Office of Graduate Studies, Texas A&M University.

Hammond, K. (1989). Case-Based Planning: Viewing Planning as a Memory Task. Academic Press.

van Harmelen, F. & Bundy, A. (1988). Explanation-based generalization = partial evaluation. Artificial Intelligence. 3(4), 251-288.

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

Howe, A., & Dahlman, E. (1993). A critical assessment of Benchmark comparison in Planning. Journal of Artificial Intelligence Research. 1, 1-15.

Howe, A., Dahlman, E., Hansen, C., vonMayrhauser, A., & Scheetz, M. (1999). Exploiting Competitive Planner Performance. Proceedings of the 5th European Conference on Planning, 62-72.

Jones, R. & Langley, P. (1995). Retrieval and Learning in Analogical Problem Solving. Proceedings of the 7th Conference of the Cognitive Science Society, 466-471.

Kambhampati, S., & Hendler, H. (1992). A Validation-Structure-Based Theory of Plan Modification and Reuse. Artificial Intelligence. 55, 193-258.

Knoblock, C. (1990). Learning Abstraction Hierarchies for Problem Solving. Proceedings of the 8th National Conference on Artificial Intelligence, 923-928.

Kolodner, J. L. (1993). Case-based Reasoning. Morgan Kaufmann.

Kuipers, B. (1994). Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge. MIT Press.

Langley, P., & Allen, J. A. (1993). A Unified Framework for Planning and Learning. Machine Learning Methods for Planning, S. Minton ed. Morgan Kaufman, 317-350.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating Classification and Association Rule Mining. Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (Plenary Presentation).

Martin, M., & Geffner, H. (2000). Learning Generalized Policies in Planning Using Concept Languages. Proceedings of the 7th International Conference on Knowledge Representation and Reasoning, 667-677.

Michalski, R. S., Mozetic, I., Hong, J. & Lavrac, H. (1986). The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. Proceedings of the 5th National Conference on Artificial Intelligence, 1041-1045.

Minton, S., (1996). Automatically Configuring Constraint Satisfaction Programs: A Case Study. Constraints. 1(1/2), 7-43.

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

Mitchell, T. (1977). Machine Learning. McGraw-Hill.

Refanidis, I., and Vlahavas, I. (2001). The GRT Planner: Backward Heuristic Construction in Forward State-Space Planning. Journal of Artificial Intelligence Research. 15, 115-161.

Rivest, R. (1987). Learning Decision Lists. Machine Learning. 2(3), 229-246.

Sutton, R. (1990). Integrated Architectures for learning, planning and reacting based on approximating dynamic programming. Proceedings of the 7th International Conference on Machine Learning, 216-224.

Sutton, R. S. & Barto A.G. (1998). Reinforcement Learning: An Introduction. MIT Press.

Tsoumakas, G., Vrakas, D., Bassiliades, N., & Vlahavas, I. (2004). Using the k nearest problems for adaptive multicriteria planning. Proceedings of the 3d Hellenic Conference on Artificial Intelligence, 132-141.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. Journal of Experimental and Theoretical Artificial Intelligence. 7(1), 81-120.

Vrakas, D., & Vlahavas, I., (2001). Combining progression and regression in state-space heuristic planning. Proceedings of the 6th European Conference on Planning, 1-12.

Vrakas, D. & Vlahavas, I. (2002). A heuristic for planning based on action evaluation. Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems and Applications, 61-70.

Vrakas, D., Tsoumakas, G., Bassiliades, N., & Vlahavas, I., (2003a). Learning rules for Adaptive Planning. Proceedings of the 13th International Conference on Automated Planning and Scheduling, 82-91.

Vrakas, D., Tsoumakas, G., Bassiliades, N., & Vlahavas, I., (2003b). Rule Induction for Automatic Configuration of Planning Systems. Technical Report TR-LPIS-142-03 , LPIS Group, Dept. of Informatics, Aristotle University of Thessaloniki, Greece.

Wang, X., (1996). A Multistrategy Learning System for Planning Operator Acquisition. Proceedings of the 3rd International Workshop on Multistrategy Learning, 23-25.

Zimmerman, T., & Kambhampati, S., (2003). Learning-Assisted Automated Planning: Looking Back, Taking Stock, Going Forward. AI Magazine. 24(2), 73-96.