Learning Rules for Adaptive Planning

Dimitris Vrakas, Grigorios Tsoumakas, Nick Bassiliades and Ioannis Vlahavas

Dept. Of Informatics,
Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
[dvrakas, greg, nbassili, vlahavas]@csd.auth.gr

Abstract

This paper presents a novel idea, which combines Planning, Machine Learning and Knowledge-Based techniques. It is concerned with the development of an adaptive planning system that can fine-tune its planning parameters based on the values of specific measurable characteristics of the given planning problem. Adaptation is guided by a rule-based system, whose knowledge has been acquired through machine learning techniques. Specifically, the algorithm of classification based on association rules was applied to a large dataset produced by results from experiments on a large number of problems used in the three AIPS Planning competitions. The paper presents experimental results with the adaptive planner, which demonstrate the boost in performance of the planning system.

Keywords: planning and learning, domain-independent classical planning, machine learning, knowledge based systems

Introduction

In domain independent heuristic planning there is a number of systems that their performance varies between best and worse on a number of toy and real-world planning domains. No planner has been proved yet to be the best for all kinds of problems and domains. Similar instability in their efficiency is also noted when different variations of the same planner are tested on the same problems, when one or more parameters of the planner are changed. Although most planners claim that the default values for their options guarantee a stable and averagely good performance, in most cases, fine tuning the parameters by hand improves the performance of the system for the problem in hand.

Few attempts have been made to explain which are the specific dynamics of a planning problem that favor a specific planning system 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.

In this paper, we used a large dataset with results from executions of our planning system, called HAP, on various problems and we employed machine learning techniques to discover knowledge that associates measurable characteris-

tics of the planning problems with specific values for the parameters of the planning system. The knowledge, acquired by the process of machine learning, was embedded in the planner in a form of a rule-based system, which can automatically change its configuration to best suit the current problem.

The resulting planning system was thoroughly tested on a number of new problems and proved to work better than any static configuration of the system. This actually shows that there are implicit dependencies between the characteristics of a problem and specific parameters of planning that worth further investigation.

The rest of the paper is organized as follows: Section 2 briefly sketches the steps of the complete methodology we followed in this work. Section 3 presents the planning system used for the purposes of our research. The next two sections describe the two different analyses we performed on the data from the experiments using statistics and machine learning respectively. Section 6 describes the implementation of the adaptive planner we developed based on the knowledge acquired from the analyses and provides experimental results. Finally, Section 7 discusses related work and Section 8 presents conclusions and poses future research directions.
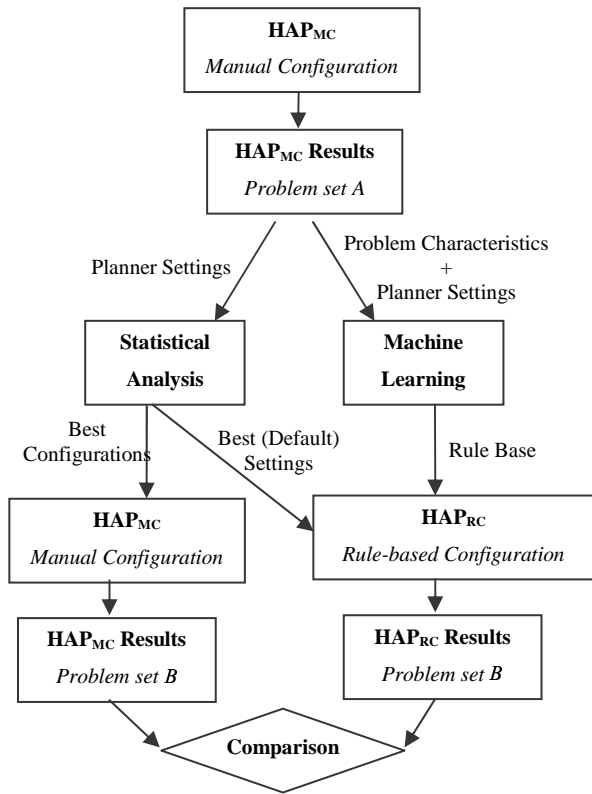
Methodology

The methodology for building and testing our system is shown in Figure 1. Here we will briefly sketch the steps we have followed. Each step is thoroughly presented in the sections to follow.

Initially we have run a large number of planning problems (97) from multiple domains (Problem set A), using many different configurations (432) on our HAP planner. The results obtained from those runs along with the planner settings were statistically analyzed and the settings that achieved the best results (on average) were obtained. These best settings were then used in the planner to run new tests on a different problem set (B) that included new problems from both existing and new domains.

Furthermore, the results obtained on the first problem set (A) along with the planner settings and the problem characteristics were used to discover associations between all

those attributes and the planner's performance. These associations (rules) were used to embed a rule base that automatically decides at run-time which is the best configuration for our planner based merely on the input problem characteristics. Since the rule base may not always have an answer on how to optimally configure the planner's settings, default values are assumed for un-configured parameters. These default values were obtained from the statistical analysis of the results of the first problem set.

The rule-based configurable planner was also tested on the new problem set (B) and the results were finally compared to the ones obtained by the statistically "best" configurations. The comparison showed that the rule-based configuration is (on average) considerably faster, finds plans with shorter length, and is equally stable.

Figure 1. Methodology for building and testing HAP$_{RC}$

The Planner

Our planning system, called HAP, is highly adjustable and can be customized by the user through a number of parameters, which are illustrated in Table 1 along with their value sets. These parameters concern the type of search, the quality of the heuristic and several other features that affect the planning process. The HAP system is based on the BP planning system (Vrakas and Vlahavas, 2001) and uses an extended version of the ACE heuristic (Vrakas and Vlahavas, 2002).

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 *direction* of search is the first adjustable parameter of HAP used in tests, with the following values: a) 0 (Regression or Backward chaining) and b) 1 (Progression or Forward chaining).

Name	Value Set	Best Setting
direction	{0,1}	0
weights (w_1 and w_2)	{0,1,2,3}	1
sof_agenda	{1,100,1000}	100
violation_penalty	{0,10,100}	10
heuristic_order	{1,2,3}	1
equal_estimation	{0,1}	1

Table 1. The value sets for planning parameters

As for the search itself, 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). For the tests with HAP, we 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 ($w_1 =1$, $w_2 =1$).

The size of the planning agenda (denoted as *sof_agenda*) of HAP also affects the search strategy and it can also be set by the user. For example, if we set *sof_agenda* to 1 and w_2 to 0, the search algorithm becomes pure Hill-Climbing, while by setting *sof_agenda* to 1, 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, even if at least one exists, 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. For the tests we used three different settings for the size of the agenda: a) 1, b) 100 and c) 1000.

The *OB* and *OB-R* functions introduced in BP and ACE respectively, are also adopted by HAP in order to search the states for violations of orderings between the facts of either the initial state or the goals, depending on the direction of the search. For each violation contained in a state, the estimated value of this state that is returned by the heuristic function, is increased by violation penalty, which is a constant number supplied by the user. For the experiments of this work we tested the HAP system with three different values of *violation_penalty*: a) 0, b) 10 and c) 100.

The HAP system employs the heuristic function of the ACE planner, plus two variations of it. There are implementations of the heuristic functions for both planning directions. All the heuristic functions are constructed in a pre-planning phase by performing a relaxed search in the

opposite direction of the one used in the search phase. During this relaxed search the heuristic function computes estimations for the distances of all grounded actions of the problem. The initial heuristic function, i.e. the one used in the ACE planning system, is described 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 MPS(S) returns a set of actions, with near minimum accumulated cost, achieving S and its algorithm 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 2. Function MPS(S)

Apart from the initial heuristic function described above, HAP embodies two variations, which in general, are more fine-grained. 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 heuristic function uses the enriched instead of the traditional add-list in the MPS function but only in the second part of the function that updates state S. So the command $Set\ S = S - add(A') \cap S$ is altered to $Set\ S = S - enriched\_add(A') \cap S$.

The second variation of the heuristic function pushes the above ideas one step further. The *enriched_add* list is also used in the first part of function MPS, 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 by configuring the *heuristic_order* parameter. The three acceptable values are: a) 1 for the initial heuristic, b) 2 for the first variation and c) 3 for the second variation.

The last parameter of HAP is *equal_estimation*, which defines the way in which states with the same estimated distances are treated. If *equal_estimation* is set to 0 then between two states with the same value in the heuristic function, 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, which is closer to the starting state.

Statistical Analysis

In order to find out which is the best configuration for our planner we have run a large number of planning problems (97) from 8 domains (*Problem Set A*, see Table 2), using 432 different configurations (Table 1) on our HAP planner. Then we tried to statistically analyze the results (plan length L_{ij} and planning time T_{ij}, for the *i*-th problem and the *j*-th configuration) obtained from those runs in order to find out their potential relationship with the planner settings.

Domain	Source
Blocks-world (3 operators)	Bibliography
Blocks-world (4 operators)	AIPS 98, 2000
Driver	AIPS 2002
Sokoban	New domain
Gripper	AIPS 98
Logistics	AIPS 98, 2000
Miconic-10	AIPS 2000
Zeno	AIPS 2002

Table 2. Domains for Problem Set A

We have performed the following:
- We found the shortest plan and minimum planning time for each problem among the tested planner configurations.

$$L_i^{min} = \min_j(L_{ij}),\ T_i^{min} = \min_j(T_{ij})$$

- We "normalized" the results by dividing the plan length and planning time of each run with the corresponding minimum problem value.

$$L_{ij}^{norm} = \frac{L_{ij}}{L_i^{min}},\ T_{ij}^{norm} = \frac{T_{ij}}{T_i^{min}}$$

- We calculated the average "normalized" length & time for each planner configuration.

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

- We counted for each planner configuration how many times it failed to solve a problem (F_j).

In order to find out which the best configurations are and how good their average performance is, we have selected those planner configurations that their F_j count is less than 5% of the total runs, the L_j^{avg} is less than 1.21 (21% worse than the minimum) and the T_j^{avg} is less than 2.5 (150% worse than the minimum). Those limits were obtained by observing the actual results in order to obtain few (5) best candidate configurations for comparison with the rule-based configuration, which will be presented later in the paper.

Furthermore, we have 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. This was done in order to decide which the best default setting is for each planner parameter, when the rule-based configuration cannot set one or more parameters. The results are shown in the third column of Table 1. We notice here that there was no clear winner value for each parameter since some values had better L_j^{avg} while others had better T_j^{avg}. The final selection of values was based first on the minimum F_j count, then on the smallest L_j^{avg} and finally on the smallest T_j^{avg}.

Machine Learning

The purpose of applying machine learning was to find interesting knowledge that associates the characteristics of a planning problem with the parameters of HAP and leads to good performance. Therefore, a first necessary step that we performed was a theoretical analysis of a planning problem, in order to discover salient characteristics that could potentially influence the choice of parameters of HAP. This resulted in an initial set of 26 measurable characteristics, presented in Table 3.

These attributes can be divided in three categories: The first category (attributes A01-A13) refer to simple and easy-to-sense-their-values characteristics of planning problems. The second category (attributes A14 – A20) consists of more sophisticated characteristics that arise from features of modern planners, such as mutexes, orderings (between goals and initial facts) and *useless* facts. The last category (attributes A21-A26) contains attributes that can be instantiated after the calculation of the heuristic functions. Attributes A23 and A26 are general and can also be used even with different heuristics. The other four attrib-

utes however, are tailored for the exact heuristics used by HAP, since they use the notion of distance of action. However, in the case of different heuristics, they could be easily replaced by other attributes such as average distance of facts.

Name	Explanation
A01	Number of facts in the initial state
A02	Number of dynamic facts in the initial state
A03	Number of static facts in the initial state
A04	Number of Goals
A05	Total number of grounded facts
A06	Total number of dynamic facts
A07	Total number of grounded actions
A08	Average number of facts per predicate
A09	Standard deviation of facts per predicate
A10	Average number of actions per operator
A11	Standard deviation of actions per operator
A12	Forward branching factor of the initial state
A13	Backward branching factor of the goals
A14	Average number of mutual exclusions per fact. A fact f is mutually exclusive with fact q, if no valid state can contain both of them. For example, empty(tank) and full(tank) are mutually exclusive
A15	Standard deviation of mutual exclusions per fact
A16	Ratio between useless and total facts in the initial state. A fact is useless if it can be safely removed without affecting the planning process
A17	Number of orderings among the goals of the problem. An ordering between goals g1 and g2 exists, if goal g1 must be achieved before g2
A18	Ratio between number of goal orderings and total number of goals
A19	Number of orderings among the facts of the initial state. These are similar to the goal orderings but are used by regression planners
A20	Ratio between number of fact orderings in the Initial state and the total number of facts in the initial state
A21	Average distance of all actions for the forward direction
A22	Standard deviation of distances of all actions for the forward direction
A23	Estimated distance between the goals and the initial state moving forward
A24	Average distance of all actions for the backward direction
A25	Standard deviation of distances of all actions for the backward direction
A26	Estimated distance between the initial state and the goals moving backward

Table 3. Problem characteristics

The next step was to study these attributes in order to discover any useful transformations that could lead to attributes carrying more meaningful and general information about planning problems. For example, we decided to transform attribute *A02* (number of dynamic facts in the

initial state) into *A02/A06* (*A06* is the total number of dynamic facts in the problem). In addition, we studied the histograms of the values of both the original and the transformed attributes calculated for the 97 problems of set A, in order to explore their distribution for interesting or trivial patterns. These plots assisted the decision for the final selection of 19 attributes (B01-B19), which are presented in Table 4.

Name	Explanation	Name	Explanation
B01	A08	*B11*	A17-A23
B02	A09	*B12*	A18-A24
B03	A10	*B13*	A23/A26
B04	A11	*B14*	A12/A13
B05	A14	*B15*	A03/A05
B06	A15	*B16*	A02/A06
B07	A18	*B17*	A04/A06
B08	A20	*B18*	A02/A04
B09	A25	*B19*	A07/A06
B10	A18-A20		

Table 4. Selected problem characteristics

The data about the selected attributes for the 97 problems of set A were subsequently joined with the data about the parameters and performance from the runs of HAP with all possible 432 configurations on the same problems. This led to a dataset of 41904 instances with 27 attributes: 19 problem characteristics (B01-B19), 6 parameters of HAP (first column of Table 1) and its performance (number of steps in plan L_{ij} and execution time T_{ij}).

The next step was to select the type of learning task that should be applied to discover a model of the dependencies between problem characteristics, planner parameters and good planning performance. A first requirement was the interpretability of the model, so that the acquired knowledge would be transparent and open to inquiries of a planning expert. Apart from developing an adaptive planner with good performance to any given planning problem, we were also interested in this work to study the resulting model for interesting new knowledge and justifications for its performance.

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. Instead, we decided to learn a rule-based classification model that would discriminate between good and bad performance based on the rest of the attributes. Then we could only select rules that have both problem characteristics and planner settings as antecedents and "*good*" performance as conclusion.

This raised the issue of how to discriminate between "*good*" and "*bad*" performance. 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 time, or create awfully large plans quickly. Since none of these cases are acceptable in real-time situations, we decided to adopt the second scenario.

In order to combine the two metrics we first normalized plan length and planning time according to the transformation presented in the previous section. We then created a combined attribute about plan quality:

$$Q_{ij} = \begin{cases} good, & L_{ij}^{norm} < 1.2, \ T_{ij}^{norm} < 1.3 \\ bad, & otherwise \end{cases}$$

This means that "*a plan is good if it is at the most 20% longer in steps than the minimum plan for the same problem and simultaneously it can be found in at most 30% longer time than the minimum required time to find any plan for the same problem*". Given the above attribute, 34% of the runs had the value *good* and the rest *bad*.

We decided to use the DMII program (Liu et al, 1999) that performs classification based on association rules (Liu et al, 1998) in order to discover useful and interpretable rules from the data. DMII requires discrete data as it is based on association rule mining. For the discretization of the 19 numeric attributes that we had, we studied histograms of the values of the attributes and split their domain into 3 regions (small, medium and large) depending on their distribution. There also exist automatic techniques for discretization, but this could result to incomprehensible rules, hence we performed manual discretization based on statistics and planning expertise.

The final dataset including the transformation of length and time into a single categorical attribute and the discretization of the 19 problem attributes, had the format shown in Table 5. DMII was run on this dataset and produced 249 rules characterizing a plan as good or bad based on the rest of the attributes.

No	Field	Range of values
1	*direction*	{0,1}
2	*weights*	{0,1,2,3}
3	*sof_agenda*	{1,100,1000}
4	*violation_penalty*	{0,10,100}
5	*heuristic_order*	{1,2,3}
6	*equal_estimation*	{0,1}
7	*B01*	{small,medium,large}
8	*B02*	{small,medium,large}
	....	
25	*B19*	{small,medium,large}
26	*quality*	{good,bad}

Table 5. Record format

The Adaptive Planner

This section describes how the results of machine learning (classification rules) have been embedded in HAP as a rule-based system that decides the optimal configuration of planning parameters based on the characteristics of a given problem. In addition, it presents experimental results that illustrate the efficiency of the resulting planning system and its superiority over the manually configurable version.

Embedding knowledge in the Planner

In order to create a rule-based system and embed it in the HAP planning system, certain issues had to be addressed:

i) Should all the rules be included?

The rules that could actually be used for adaptive planning are those that associated, 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 or only planning parameters as their andecedents. This process filtered out 13 rules from the initial set of 249 rules.

Within the remaining 236 rules there were 167 rules modeling "*bad*" performance and 69 modeling "*good*" performance. Although the rules modeling bad performance contain knowledge that could possibly be used by the planer, they were not considered in the current work.

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

The 69 rules that were finally selected mainly tune the *direction* and *weights* parameters, but there were also rules affecting all the rest of the parameters. Table 6 shows the number of rules affecting each planning parameter.

Parameter	Number of rules
direction	59
weights	31
Sof_agenda	2
violation_penalty	2
heuristic_order	15
equal_estimation	6

Table 6. Distribution of rules over parameters

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

Each rule was accompanied by two metrics (confidence and support) used in association rules. The confidence factor indicates 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 support factor indicates how often the pattern in the condition and the conclusion of the rule is met compared to the complete data set. A low support indicates that the rule describes a rare situation.

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 a primary concern. That is why the conflict resolution strategy used in our rule-based system is based on the total ordering of rules according first to the confidence and then on the support factors, both 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. Then, among rules with the same confidence we prefer to first examine a rule with better support, i.e. a rule that describes a more frequent pattern, because this rule is more likely to apply to a random situation, than the rest.

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 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.

iii) 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 2.7 planning parameters, leaving at the same time 3.3 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 used statistics to find the best settings for each independent parameter. These settings are illustrated in Table 1.
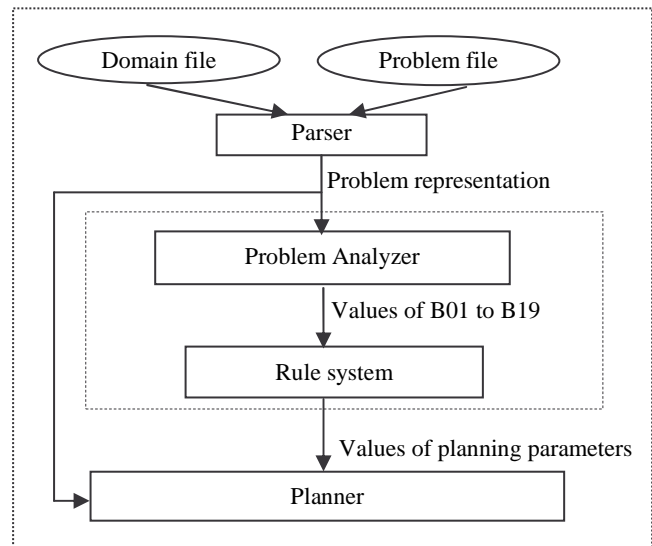

Figure 3. **HAP$_{RC}$ Architecture**

The rule configurable version of HAP, which is outlined in Figure 3 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 19 problem characteristics (B01-B19) used by the rules. These values are then passed in 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 in the planning module, in order to solve the problem.

Experimental Results

In order to test the efficiency of HAP_{RC} and the boost in performance offered by the adaptive way in which the parameters are configured, we decided to run it on two different sets of problems: Problem set A, which was used in the statistical analysis and in the Machine Learning process and Problem set B, which contains 50 new problems; 30 from domains in set A (Blocks, Logistics, MIC-10) and 20 from new domains (puzzle, hanoi, mystery). The experiments with problem set A, aim at verifying the correct implementation of the rule system in HAP_{RC} and testing whether there is actual need for different setups for different problems, while problem set B aims at showing if the learned model can generalize effectively to new problems and domains.

All the runs of HAP_{RC} and HAP_{MC}, 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 unsolvable.

Table 7 presents the average "normalized" length (L_j^{avg}) and planning time (T_j^{avg}) for the best five manual configurations of HAP_{MC} and for the HAP_{RC} system over the problems of set A. It is worth noting here that the five configurations presented in **Table 7** are the best of those being at the same time good (length of plans), fast (planning time) and stable (failures). This means that there were other configurations, not included in this table, which achieved better performance for one factor but performed very bad for the other two. For example, the absolute minimum L_j^{avg} (1.043) was achieved by a configuration that exhibited very bad T_j^{avg} (42.890) and managed to solve less than 65% of the problems. Similarly the fastest configuration had T_j^{avg} =1.530, but exhibited L_j^{avg} =1.223 and did not find a solution for more than 22% of the tested problems.

Planner	L_j^{avg}	T_j^{avg}	Failures
MC1	1,180	2,390	2%
MC2	1,205	2,310	2%
MC3	1,205	2,465	2%
MC4	1,206	2,396	2%
MC5	1,208	2,200	3%
HAP_RC	1,098	2,131	2%

Table 7. Comparative Results for Problem set A

From the configurations being at the same time good in all criteria, HAP_{RC} was the fastest and managed to find much shorter plans. It is clear from these results that there is no such thing as a generally best configuration and the best configuration for a specific problem seems to depend on the problem characteristics traced by HAP_{RC}.

Problem	MC1	MC2	MC3	MC4	MC5	HAP_RC
B1	*10*	*10*	*10*	*10*	*10*	*10*
B2	24	24	24	24	24	*22*
B3	*20*	*20*	*20*	*20*	*20*	*20*
B4	40	40	36	36	40	*28*
B5	42	42	42	42	42	*32*
B6	30	34	30	34	30	*30*
B7	64	64	64	64	64	*44*
B8			62	62		*56*
B9	42	42	*38*	*38*	42	48
B10	106	114		116	*106*	
L1	51	51	49	49	51	*43*
L2	71	71	76	76	71	*63*
L3	84	84	84	84	84	*72*
L4	87	87	76	76	85	*66*
L5	90	90	98	98	90	*76*
L6	86	86	93	93	86	*71*
L7	114	114	104	104	112	*104*
L8	122	122	134	134	*114*	134
L9	107	107	107	107	*99*	107
L10	*112*	*112*	113	110	114	113
S1	35	35	*34*	*34*	35	35
S2	*38*	*38*	40	40	*38*	40
S3	42	42	42	42	42	*41*
S4	*47*	*47*	48	48	*47*	*47*
S5	52	52	52	52	52	*51*
S6	*54*	*54*	*54*	*54*	*54*	*54*
S7	*58*	*58*	*58*	*58*	*58*	59
S8	58	58	59	59	58	*57*
S9	64	64	*61*	*61*	64	64
S10	*68*	*68*	*68*	*68*	*68*	*68*
L_j^{avg}	1.120	1.128	1.082	1.122	1.114	*1.006*

Table 8. Plan length for new problems

The next step was to test whether the knowledge learned by the planning system from problem set A could be used to effectively guide the system for other problems and do-

mains not included in the learning set. For this reason, we tested HAP$_{RC}$ and the best five configurations of HAP$_{MC}$ on the problems of set B and recorded for each run the time needed to solve the problem and the plan length.

Table 8 and **Table 9** present the length of found plans and the planning time needed by the six planners to solve the 30 new problems of the previously used domains. The last rows present the average normalized length and time over the 30 problems. A planner that did not find a plan has a void result. Best results are emphasized.

With respect to plan length we notice that HAP$_{RC}$ is the best planner on average; it is only 0.6% worse than the best configurations for each problem and had the best plan length for 21 out of 30 problems. The second best manual configuration on average MC3 was 8.2% worse than the best configurations and had the best plan for 8 out of the 30 problems. The results for planning time were similar. HAP$_{RC}$ was the best planner being this time 8.4% worse than the best configurations and being the best in 16 out of 30 problems. The second best planner was this time MC1.

Table 10 and Table 11 present the length of found plans and the planning time needed by the six planners to solve the 20 problems of the three new domains. The formatting of the tables is similar to the previous ones.

Problem	MC1	MC2	MC3	MC4	MC5	HAP$_{RC}$
P1	47	47	47	47	47	*43*
P2	47	47	47	47	47	*35*
P3	46	46	46	72	46	90
P4	46	46	36	36	46	*22*
P5	132	132	110	106	122	*110*
P6	194	194	166	180	202	*166*
H1	*7*	*7*	*7*	*7*	*7*	7
H2	*15*	*15*	*15*	18	*15*	15
H3	*47*	*47*	*47*	*47*	*47*	47
H4	86	86	79	82	86	*77*
H5	182	182	190	196	182	*133*
H6	354	354	363	396	354	*275*
M1	*5*	*5*	*5*	*5*	*5*	5
M2	*8*	*8*	10	10	*8*	8
M3	*7*	*7*	*7*	*7*	*7*	9
M4						
M5	*4*	*4*	*4*	*4*	*4*	*4*
M6	7	7	7	7	7	*5*
M7						
M8	8	*6*	7	9	7	*6*
L_j^{avg}	1.250	1.231	1.204	1.281	1.240	*1.125*

Table 10. Plan length for new domains

Problem	MC1	MC2	MC3	MC4	MC5	HAP$_{RC}$
B1	*50*	*50*	60	*50*	60	*50*
B2	70	70	70	70	70	*60*
B3	*90*	*90*	100	*90*	*90*	*90*
B4	*110*	120	120	*110*	120	*110*
B5	190	210	280	260	210	*160*
B6	240	240	240	*230*	240	*230*
B7	540	580	1860	570	590	*360*
B8			48900	62490		*890*
B9	540	550	540	*530*	560	550
B10	87810	118320		*40410*	112640	
L1	280	290	280	*270*	280	290
L2	*310*	*310*	320	*310*	320	320
L3	330	330	320	*300*	330	330
L4	580	600	*570*	590	650	610
L5	*640*	660	1080	930	670	670
L6	*610*	*610*	920	760	660	690
L7	2660	2760	1240	*1200*	2160	1240
L8	1340	1360	2390	2290	*1200*	2400
L9	*1150*	1180	2390	2000	*1150*	2400
L10	1460	1490	1380	*1350*	1650	1380
S1	290	310	290	*280*	300	*280*
S2	*360*	*360*	360	*360*	370	370
S3	450	460	460	440	460	*430*
S4	540	570	600	570	560	*530*
S5	670	690	700	670	690	*660*
S6	790	790	820	800	810	*780*
S7	980	990	980	970	1010	*950*
S8	1150	1140	1160	1140	1160	*1120*
S9	1410	1420	1360	1350	1460	*1310*
S10	*1640*	1660	1650	1660	1690	1650
T_j^{avg}	1.158	1.212	3.109	3.467	1.204	**1.084**

Table 9. Planning time for new problems

Problem	MC1	MC2	MC3	MC4	MC5	HAP$_{RC}$
P1	230	240	260	230	240	*210*
P2	230	240	260	230	230	*210*
P3	*230*	240	*230*	300	240	260
P4	380	410	210	210	390	*200*
P5	32450	33020	61230	*14410*	79060	63620
P6	9690	9980	12630	6660	*5040*	7900
H1	40	40	40	40	40	*30*
H2	*40*	50	60	*40*	50	50
H3	*70*	80	100	*70*	*70*	90
H4	170	200	220	140	190	*130*
H5	600	670	740	*590*	690	900
H6	*4130*	4390	4960	4380	4590	4350
M1	210	210	220	*200*	*200*	210
M2	1540	1570	1980	4120	1540	*950*
M3	420	430	440	430	*390*	*390*
M4						
M5	160	160	160	160	*150*	170
M6	940	950	*920*	1360	1110	1010
M7						
M8	800	790	500	600	1000	*340*
T_j^{avg}	1.390	1.462	1.587	1.387	1.600	*1.365*

Table 11. Planning time for new domains

With respect to plan length we notice that HAP_{RC} is again the best planner on average, being 12.5% worse than the best configurations for each problem and had the best plan length for 16 out of 20 problems. The second best manual configuration on average was again MC3 which was 20.4% worse than the best configurations and had the best plan for 6 out of the 20 problems. The results for planning time were again similar. HAP_{RC} was the best planner being this time 36.5% worse than the best configurations and being the best in 8 out of 20 problems. The second best planner was again MC1, being 39% worse than the best configurations.

It is clear from these results that HAP_{RC} is able to adapt itself and use the configuration that best fits each problem. It was on average faster than all the other configurations in problems from "known" domains and it was also able to create shorter plans. The generality of the learned knowledge was also empirically shown, as HAP_{RC} was on average better in both planning time and plan length even on problems of new domains.

The superiority of HAP_{RC} over the static configurations can be better noticed from the overall results of problem set B, including old and new domains, in Table 12 that support our main argument: "*There might be a specific configuration which bests HAP_{RC} in a problem or in a few problems of the same domain. However, in the large picture, where the planners are tested on many problems from various domains, HAP_{RC} is clearly better than any static configuration*".

Planner	L_j^{avg}	T_j^{avg}	Failures
MC1	1.169	1.245	6%
MC2	1.166	1.305	6%
MC3	1.127	2.550	6%
MC4	1.180	2.702	4%
MC5	1.160	1.350	6%
HAP_{RC}	1.050	1.187	6%

Table 12. Overall results for problem set B

An interesting point that rises from studying the results is that the difference in performance of HAP_{RC} with the manual configurations is greater in plan length than in planning time, especially for the 20 problems of the new domains. This could be associated with the fact that we were more strict with the plan length (< 1.2) than with the planning time (< 1.3) in the definition of the quality attribute. It seems that the definition of the plan quality attribute that combines the two metrics is influencing the learning process and the final rule-base, and thus can be used for biasing the system towards better plan lengths or fastest planning. Our configuration ensures the best of both with a leaning to plan length, which reflects the general truism in planning that one is willing to sacrifice a little more time in order to achieve a much better plan.

Furthermore, we notice that the difference of HAP_{RC} performance in comparison to the best configurations is smaller in the problems of the previously used domains than in the problems of the new domains. This was expected as the number of problems used for learning is in general small and the learned knowledge is biased towards the domains of these problems. Still the rules were able to generalize to the new domains with acceptable performance. Using more problems for the learning process is expected to increase the stability and quality of the rule-base.

Finally we notice that from the manual configurations MC3 was consistently better in plan length and MC1 better in planning time than the other manual configurations. This is an indication that static configurations either favor plan length or planning time, while an adaptive planner like HAP_{RC} can perform best in both given a specific problem.

Related work

Machine Learning has been exploited in the past for Planning, mainly in order to learn control rules. The PRODIGY Architecture (Veloso et al, 1995) was the main representative of this trend. This architecture, supported by various learning modules, focuses on learning the necessary knowledge that guides a planner to decide what action to take next during plan execution.

Machine Learning has also been utilized for automatically extracting rules for plan rewriting (Ambite, Knoblock and Minton, 2000). Plan rewriting rules are used for improving easy-to-generate low quality plans.

Approaches towards exploiting domain and problem characteristics in a pre-planning phase have been presented in the past by Fox and Long (Long and Fox, 1999, Fox et al, 2001). Their research is mainly focused on state analysis and its use by automated planning systems, such as STAN (Long and Fox, 1998) and Hybrid STAN (Fox and Long, 2000).

Hoffman (Hoffman, 2001) discusses the matter of when a specific planner will behave well and when not by performing domain analysis. He created taxonomy of most of the planning domains based on the existence of specific characteristics such as local minima and dead ends in these domains. With this taxonomy he is able to explain the variations in performance of some of the state-of-the-art planning systems.

Probably, the only approach to the direction of adaptive planning done in the past is the work presented in (Howe and Dahlman, 1993, Howe et al, 1999). They have created a system called BUS, which incorporates six state-of-the-art planners (STAN, IPP, SGP, BlackBox, UCPOP and Prodigy) and runs them using a round-robin schema, until one of them finds a solution. BUS is adaptable in the sense of deciding the ordering of the six planners and the duration of the time slices dynamically based on the values of five problem characteristics and some rules extracted from

a statistical analysis on past runs. The system achieved more stable behavior but it was not as fast as one may have expected.

MULTI-TAC (Minton, 1996) is a learning system which uses a library of heuristics and generic algorithms and automatically fine tunes itself in order to synthesize the most appropriate constraint satisfaction program to solve a problem. The methodology we followed in this paper presents some similarities with MULTI-TAC.

Conclusions and Future Work

This paper reported on ongoing research in the field of applying Machine Learning and Rule-based techniques on Planning in order to build an adaptive planning system that can automatically fine-tune its parameters based on the values of measurable characteristics of each problem. The adaptable planner we created was tested on a large number of problems from various domains and the experimental results have proven that there is no static configuration, adopted by statistical methods, of the planner that has such a stably good performance over different problems and domains.

The rule-based configuration approach we have developed produces better results than the best configurations and the best individual settings, because we treat planner parameters as associations of the problem characteristics, whereas the statistical analysis tries to associate planner performance with planner settings, ignoring problem characteristics.

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.

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 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.

Acknowledgments

References

Ambite, J. L., Knoblock, C., and Minton, S., 2000. Learning Plan Rewriting Rules. In Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling Systems.

Fox, M., and Long, D., 2000. Hybrid Stan: Identifying and managing combinatorial sub-problems in planning. In Proceedings of the 19th UK Planning and Scheduling SIG workshop.

Fox, M., Long, D., Bradley, S., and McKinna, J., 2001. Using model checking for pre-planning analysis. In Proceedings of the AAAI Symposium on Model–based Validation of Intelligence.

Hoffman, J., 2001. Local search topology in planning benchmarks: An empirical analysis. In Proceedings of the 17th International Joint Conference on Artificial Intelligence.

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

Howe, A., et al. 1999. Exploiting Competitive Planner Performance. In Proceedings of the 5th European Conference on Planning.

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

Liu, B., Hsu, W., Ma, Y., and Chen, S., 1999. Discovering Interesting Knowledge using DM-II. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Industrial Track).

Long, D., and Fox, M., 1998. Efficient implementation of the plan graph in stan. Journal of Artificial Intelligence Research 10:87-115.

Long, D. and Fox, M., 1999. Automatic synthesis and use of generic types in planning. Technical Report.

Minton, S., 1996. Automatically Configuring Constraint Satisfaction Programs: A Case Study. Constraints 1(1).

Veloso, M., et. al 1995. Integrating planning and learning: The prodigy architecture. Journal of Experimetnal and Theoretical Artificial Intelligence 7(1):81-120.

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

Vrakas, D. and Vlahavas, I. A heuristic for planning based on action evaluation. In Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems and Applications.