Lazy Adaptive Multicriteria Planning

Grigorios Tsoumakas and **Dimitris Vrakas** and **Nick Bassiliades** and **Ioannis Vlahavas**[1]

Abstract. This paper describes a learning system for the automatic configuration of domain independent planning systems, based on measurable features of planning problems. The purpose of the Lazy Adaptive Multicriteria Planning (LAMP) system is to configure a planner in an optimal way, concerning two quality metrics (i.e. execution speed and plan quality), for a given problem according to user-specified preferences. The training data are produced by running the planner under consideration on a set of problems using all possible parameter configurations and recording the planning time and the plan length. When a new problem arises, LAMP extracts the values for a number of domain-expert specified problem features and uses them to identify the k nearest problems solved in the past. The system then performs a multicriteria combination of the performances of the retrieved problems according to user-specified weights that specify the relative importance of the quality metrics and selects the configuration with the best score. Experimental results show that LAMP improves the performance of the default configuration of two already well-performing planning systems in a variety of planning problems.

1 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 paper presents a learning system for dealing with the aforementioned issue. The Lazy Adaptive Multicriteria Planning (LAMP) system automatically configures the parameters (such as search direction and agenda size) of a planner based on measurable characteristics (such as number of actions per operator and mutual exclusions between facts) of planning problems. Learning data are produced by running the planner under consideration off-line on several planning problems using all combinations of values for its parameters. When LAMP is faced with a new problem, it retrieves the recorded performance (execution time and plan length) for all parameter configurations of the k nearest problems and performs a multicriteria combination with user-specified weights. The configuration with the best combined score is then used for running the planner with the new problem.

The utility of LAMP was evaluated using two state-of-the-art domain independent planning systems. The results showed that LAMP manages to increase their performance in a variety of planning problems. In addition, the use of different weights for planning speed and plan length had the expected effect of biasing the planning systems towards optimizing the corresponding performance criterion.

To further improve the results we also experimented with a feature weighting method to deal with the potential noisy, interacting and irrelevant features. Specifically, we adapted the RReliefF algorithm [7] for attribute estimation in regression to the requirements of our application domain.

The rest of the paper is organized as follows. Section 2 presents related work on automatic configuration of problem-solving systems. Section 3 describes the features that were extracted from planning problems. The next section describes the training data collection process and the operation of the LAMP system, along with important advantages and disadvantages. Section 5 presents our adaptation of the RReliefF algorithm for feature weighting. Section 6 describes the experimental setup and Section 7 the results and their discussion. The last section concludes this work and points areas for improvements.

2 RELATED WORK

The idea of employing Machine Learning for the automatic configuration of problem-solving systems in order to solve specific problems effectively is not new. It has inspired researchers to develop self-tuned AI systems that deal with problems in various areas, such as Constraint Satisfaction, Planning and Machine Learning.

MULTI-TAC [6] is an Analytic Learning system for Constraint-Satisfaction Problems (CSP). It automatically fine-tunes itself in order to synthesize the most appropriate constraint satisfaction program to solve a problem, using a library of heuristics and generic algorithms.

Fink [3] has presented a methodology for selecting among different search strategies and setting a time-bound for running the search. The application domain that motivated him, was the PRODIGY planner [2] and the 3 different search strategies it employed. Fink's methodology relies mainly on statistical analysis of past performance data. It extends this methodology with domain knowledge by considering just the problem size and using regression for predicting the time-bound for the selected search strategy.

Two adaptive planners that were build with learning from performance data of HAP (Highly Adjustable Planner) [10] are HAP$_{RC}$

[1] Dept. of Informatics, Aristotle University of Thessaloniki, 54124 Greece, email:{greg,dvrakas,nbassili,vlahavas}@csd.auth.gr

[11] and HAP$_{NN}$ [8]. Both are capable of automatically fine-tuning their planning parameters based on features 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 rule learning over a dataset containing performance data of past executions of HAP. The tuning of HAP$_{NN}$ is performed through instance-based learning 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.

The proposed system, LAMP (Lazy Adaptive Multicriteria Planning), uses a generalization of the learning methodology of HAP$_{NN}$ that can be utilized by any modern parameterized domain-independent planning system. In addition, it incorporates a feature weighting approach.

Various approaches of automatic system tuning have recently appeared in the domain of Machine Learning itself. Learning is employed in order to either select the best configuration of a learning algorithm, or the most suitable one from a pool of learning algorithms, based on measurable features of the learning problem at hand. Such approaches are commonly referred to as Meta-Learning [9].

3 FEATURE EXTRACTION

An important decision in the design of any successful learning system is the selection of appropriate experience (training data). For LAMP this comes down to the selection of suitable problem features that correlate with the performance of the different planner configurations. Therefore, a first necessary step that we performed was a theoretical analysis of planning problems, in order to discover salient features that could influence the choice of planning parameters.

Our main concern was to select attributes that their values are easily calculated rather than complex attributes that would cause a large overhead in the total planning time. Therefore, most of the attributes come directly from the PDDL (Planning Domain Definition Language) files, which are the default input to planning systems, and their values can be promptly calculated.

A second concern which influenced the selection of attributes was the fact that they should be general enough to be applied to all domains and their values should not depend so much on the size of the problem. Otherwise the knowledge learned from easy problems would not be applied effectively to difficult ones. For example, instead of using the number of mutexes (mutual exclusions between facts) in the problem as 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 and this attribute (mutex density) identifies the complexity of the problem without taking into account the problem size. This is a general solution followed in all situations where a problem attribute depends nearly linearly on the size of the problem.

Taking the above into consideration we resulted in a set of 26 numerical characteristics, which can be divided in two categories: The first category refers to simple and easily measured characteristics of planning problems, e.g. number of actions per operator, that source directly from the input files. The second category consists of more sophisticated characteristics that arise from features of modern planners, such as mutexes or orderings (between goals and initial facts). These characteristics are useful even for the automatic configuration of planners that do not exploit them, since they capture interesting aspects of the problems' morphology. Moreover, there exist very efficient implementations of techniques for sensing their values and the overhead imposed by them is quite low.

4 THE LAMP SYSTEM

LAMP uses a lazy learning approach. Therefore, the training of the system simply requires the collection of the training data. This process is discussed in the following subsection. The second subsection describes the methodology for selecting the optimal configuration according to user preferences using the k Nearest Neighbors (kNN) algorithm. The last subsection describes advantages and disadvantages of LAMP compared to past related approaches.

4.1 Collection of training data

The training data concern the performance of the planning system under consideration on a number of planning problems using all combinations of values for its planning parameters. The whole process of recording the training data is illustrated in Figure 1.

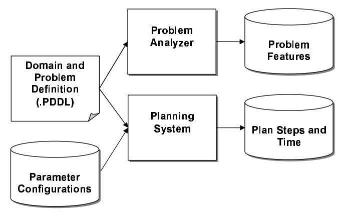


Figure 1. The process of training data collection.

For each run of the planner we record the features of the problem, the performance of the planner (steps of the resulting plan and required planning time) and the configuration of parameters. The latter two are recorded straightforwardly as they are the input and output of the planning system. For the efficient calculation of the problem features we developed a problem analyzer that takes as input the PDDL files of the planning problem and outputs the values of the features.

The training data were organized as a multi-relational data set, consisting of 2 primary tables, *problems* (M rows) and *parameters* (N rows), and a relation table *performances* ($M * N$ rows), in order to save storage space, enhance the search for the k nearest neighbors and speed up the retrieval of the corresponding performances.

4.2 Selection of the optimal configuration

Given a new planning problem p_n, LAMP first calculates the values of the problem features using the problem analyzer. Then the kNN algorithm [1] is engaged in order to retrieve the *ids* of the k nearest problems from the *problems* table. In the implementation of kNN we use the Manhattan measure with the normalized values of the problem attributes to calculate the distance between p_n and all problems p_x in the *problems* table:

$$\delta(p_n, p_x) = \sum_f \frac{|p_n(f) - p_x(f)|}{max(f) - min(f)} \qquad (1)$$

where f is a problem feature, $p_n(f)$ and $p_x(f)$ the values of this feature for problems p_n and p_x respectively and $max(f)$ and $min(f)$ are the maximum and minimum values of this feature.

The *ids* of the k nearest problems are then used by LAMP to retrieve the corresponding plan steps and planning time for all possible configurations from the *performance* table. The next step is to combine these performances in order to suggest a single parameter configuration with the optimal performance.

Optimal is however susceptible to user preferences, i.e. a shorter plan is usually preferred than a longer one, but there are cases (e.g. real time systems) where the planner must respond promptly even at the expense of the quality of the resulting plan. Since, these two criteria (fast planning, short plans) are contradictory, it is up to the users to set up their priorities.

LAMP has the advantage of letting the users express their priorities through two parameters: w_s (weight of steps) and w_t (weight of time). The overall planner performance is calculated as a multi-criteria combination of the steps and time based on these weights. Specifically, the widely used Weighted Sum method [5] is applied to obtain an overall score from the two criteria, which must first be normalized.

Let S_{ij} be the number of plan steps and T_{ij} be the required time to build it for problem i (i=1..k) and planner configuration j (j=1..N). First, we find the shortest plan and minimum planning time for each problem among the tested planner configurations:

$$S_i^{min} = \min S_{ij} \qquad T_i^{min} = \min T_{ij}$$

Then, we normalize the results by dividing the minimum plan length and minimum planning time of each run with the corresponding problem value.

$$S_{ij}^{norm} = \frac{S_i^{min}}{S_{ij}} \qquad T_{ij}^{norm} = \frac{T_i^{min}}{T_{ij}}$$

Subsequently LAMP calculates an overall score as the average of the normalized criteria weighted by the user-specified weights:

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

Finally, LAMP averages the planner configuration scores across the k nearest problems and outputs the one with the largest average. The whole process is illustrated in Figure 2.

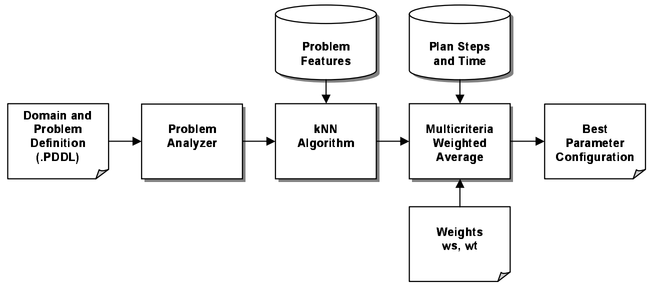

Problem Features
Plan Steps and Time
Domain and Problem Definition (.PDDL)
Problem Analyzer
kNN Algorithm
Multicriteria Weighted Average
Best Parameter Configuration
Weights ws, wt

Figure 2. Predicting the best parameter configuration for a problem.

4.3 Advantages and disadvantages

LAMP allows users to specify their priorities for steps and time through the values of the corresponding weights. As it is shown from the experimental results, setting the weights in favor of either of the criteria has the desired effect on the planner performance. This is a very useful feature of LAMP that is feasible due to the use of a lazy learning approach. An eager learning algorithm would require users to wait until a new model is built for the specific weights, before it can be used to predict the optimal planner configuration. This would increase the complexity of the approach and lead to an inflexible system.

Lazy learning offers another important advantage. It is not necessary to train a different model when a new bunch of training data are produced. This makes it possible to incrementally enrich LAMP with new problems and thus enhance its predictive performance. For each new problem, the training data collection process is followed off-line and the produced data are added to the *problems* and *performances* tables. LAMP can then immediately take advantage of this new information, as it does not require re-training.

In addition, it is worth noting that LAMP can actually output a vector of average scores for all parameter configurations instead of a single 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 of the configurations for non time-critical applications is to run the planner with the top performing configurations sequentially or in parallel and obtain the smallest plan.

The disadvantages of LAMP are increased storage needs and time during prediction. The system must locate the k nearest problems of a new problem, which requires a scan of all records in the *problem* table and calculation of distances, and then retrieve the performances through k disk accesses at the *performance* table in the worst case. However, there exist efficient indexing techniques for reducing the complexity of kNN queries in large databases. In any case, the classification time is only a fraction of the total planning time for real-world planning problems.

5 FEATURE WEIGHTING

A known limitation of lazy learning algorithms of the k-nearest neighbor (kNN) family is their sensitivity to irrelevant, redundant, interacting and noisy features. As we were uncertain about the utility of the features that were extracted from planning problems by the domain-expert analysis, we decided to adopt a feature pre-processing method.

Wettschereck et. al [12], argue that feature weighting methods tend to outperform feature selection algorithms for tasks where some features are useful but less important than others. In the same study it has been shown that feature weighting methods that use performance feedback to assign weight settings require less pre-processing, perform better in the presence of interacting features and generally require less training data to learn good settings. For the above reasons that hold in our application domain, we decided to employ a performance feedback feature weighting method.

Specifically, LAMP performs feature weighting based on the adaptation of the Relief algorithm for attribute estimation in regression [7]. RReliefF's estimate $W[A]$ of the quality of an attribute A is given by the following equation:

$$W[A] = \frac{P_{dC|dA} P_{dA}}{P_{dC}} - \frac{(1 - P_{dC|dA}) P_{dA}}{1 - P_{dC}} \tag{2}$$

where P_{dA} is the probability that the nearest instances to an instance have different values for attribute A, P_{dC} is the probability that the nearest instances to an instance have different predictions and $P_{dC|dA}$ is the probability that the nearest instances to an instance with different values for attribute A have different predictions.

To approximate these probabilities, the RReliefF algorithm randomly selects m instances R_i from the training set and for each one of them it finds its k nearest instances I_j. For each of the nearest instances it updates the probability estimates by considering the difference of the values of each attribute A between R_i and I_j and the differences in predictions.

However, in our case the prediction is associated with a vector of scores for the different configurations, instead of a single number. To deal with this problem we employed *Pearson's product-moment correlation coefficient* (r) that measures the linear correlation of two such vectors X, Y of size n:

$$r_{XY} = \frac{n\sum XY - (\sum X)(\sum Y)}{\sqrt{[n\sum X^2 - (\sum X)^2][n\sum Y^2 - (\sum Y)^2]}} \quad (3)$$

This returns a number between -1 and 1, where 1 indicates positive correlation, -1 negative correlation and 0 no correlation. Within RReliefF we estimate the difference in prediction as $1 - r_{XY}$, so that perfectly correlated configuration scores have 0 difference and perfectly uncorrelated ones have a difference of 1.

6 EXPERIMENTAL SETUP

For the evaluation of the learning performance of LAMP, an actual domain-independent planning system along with the descriptions of several planning problems are required in order to collect the necessary data. The next two subsections deal exactly with these issues. The first describes the planning systems and the following the planning problems that were used.

6.1 The planning systems

LAMP was used for the automatic configuration of LPG [4] and HAP [10], two publicly available state-of-the-art planning systems from two different research groups working on domain-independent planning. LPG is a planning system that performs stochastic local search in temporal Action Graphs. It can be customized through a number of parameters, but for the purposes of this research we selected the 4 with the maximum impact on the performance of the planner, which are outlined in Table 1. HAP is a planning system that performs a classical search in the space of states. It uses various heuristic mechanisms in order to enhance this search. The system can be configured through the 7 planning parameters that are outline in Table 1.

Table 1. The planning parameters of LPG and HAP and their value sets.

Planner	Parameter	Value Set
LPG	Heuristic	{1, 2}
	Restarts	{25, 50, 75}
	Search Steps	{100, 500, 1000}
	IChoice	{1, 2, 3, 4}
HAP	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}

6.2 The planning problems

For the production of training data we run LPG and HAP using all possible parameter configurations on a total of 450 planning problems, which correspond to 30 planning problems from each of the 15 planning domains, outlined in Table 2. Some problems were not solved by any configuration of LPG and HAP. These were excluded from learning as they do not offer any information. The actual number of problems per domain that were used for training LAMP for each planner are given in the last two columns of Table 2.

Table 2. Planning domains, their sources and the number of problems solved by at least one configuration for each of the two planners.

Domain	Source	LPG	HAP
Assembly	New domain	30	29
Blocks-world (3 operators)	Bibliography	30	30
Blocks-world (4 operators)	AIPS 98, 2000	30	30
Driver	AIPS 2002	30	30
Ferry	FF collection	28	28
Freecell	AIPS 2000, 2002	9	30
Gripper	AIPS 98	30	30
Hanoi	Bibliography	17	28
Sokoban	New domain	15	28
Logistics	AIPS 98, 2000	30	30
Miconic-10	AIPS 2000	28	30
Mystery	AIPS 98	29	30
Tsp	FF collection	30	30
Windows	New domain	30	30
Zeno	AIPS 2002	29	30
	Total	395	443

7 RESULTS AND DISCUSSION

This section evaluates a) the usefulness of LAMP in boosting the performance of the planning systems with and without feature weighting, and b) the effect of the multicriteria weights on the performance of the planner in terms of plan steps and planning time.

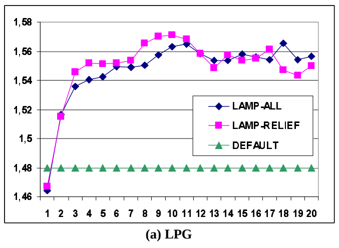
7.1 Evaluating the performance

10-fold cross-validation was used to accurately evaluate the automatic configuration of LPG and HAP by LAMP. Specifically the original sets of problems (395 for LPG and 443 for HAP) were split into 10 problem sets of equal size. The training set for each fold was used to calculate the feature weights using our adaptation of RReliefF and to find the k nearest problems for each of the problems in the test set, using a) the weights selected by RReliefF, b) all weights equal to 1.

The default configuration of each planner served as a baseline for comparison with its automatic configuration as predicted by LAMP. We used equal weights w_s and w_t for the two performance metrics and varied the number of nearest neighbors from 1 to 20. We recorded the score of the default and predicted configurations on all problems and calculated the averages. Figure 3 shows the average score of LAMP with and without the feature weighting method for the different number of nearest neighbors along with the score of the default configurations for LPG and HAP.

A first conclusion stemming from the experimental results is that LAMP manages to achieve a good boost in performance compared to that of the default configurations of the planners, especially for values of k greater than 2. The best results were noticed for a value of k equal to 10 and the feature weights that were calculated by RReliefF. This led to a gain of 6% in the performance of LPG and 10% in the performance of HAP.

The results show that LAMP manages to achieve better performance in the automatic configuration of HAP than of LPG. A reason behind this could be the fact that the training data for HAP are



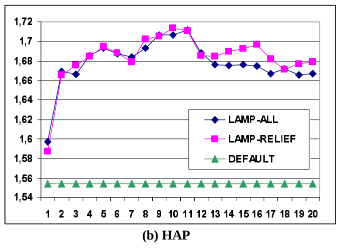


Figure 3. Average score of (a) LPG and (b) HAP using their default configurations and the automatic configurations selected by LAMP with and without the RReliefF feature weighting method.

10% more than that of LPG. Another reason is that HAP with 7 tunable parameters is more flexible than LPG that only has 4. It seems plausible that the larger the space of configurations, the better the adaptation. However, a larger configuration space imposes a higher computational cost for training data collection.

A second conclusion from the experimental results is that RReliefF manages to increase the performance of LAMP on average, but it does not seem to make a statistically significant contribution. This probably means that Pearson's product-moment correlation coefficient is not a suitable statistic for this kind of problems. The truth is that the large number of degrees of freedom (70 for LPG and 862 for HAP) make the coefficient unreliable, especially as it is insensitive to whether the differences of the vectors concern the best performing configurations, that actually influence the selection of the best configuration, or not.

7.2 Evaluating the multicriteria weights

In order to evaluate the effect of the multicriteria weights on the performance of the planners, we run LAMP with three different sets of weights: a) w_s=1, w_t=1, b) w_s=1, w_t=2, and c) w_s=2, w_t=1. Figure 4 compares the average normalized steps and time for the three different weight settings for all nearest neighbors.

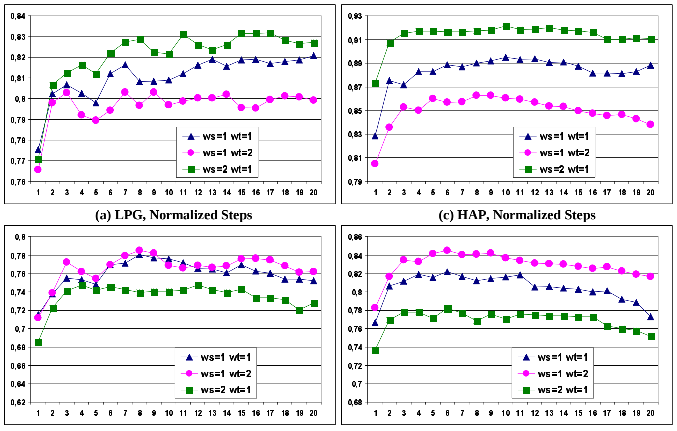


Figure 4. Normalized steps and time of LPG and HAP.

An important conclusion from these graphs is that setting the weights in favor of planning speed or plan quality has the desired effect on the performance of the planners. We notice that normalized steps are increased for larger weight to steps and decreased for larger weight to time. The same applies to normalized time.

8 CONCLUSIONS AND FUTURE WORK

This work combines two important areas of Artificial Intelligence. It utilizes Machine Learning to adapt domain independent Planning systems to a) the given planning problem that they have to solve and b) the preferences of the users in terms of plan quality and execution speed. The experimental results showed that LAMP generalizes successfully from past runs of at least two well-performing planning systems and manages to boost their performance on a variety of planning problems.

In the future we plan to find paths to further improve the performance of LAMP. A first thing to research into is an effective feature weighting or feature selection method, taking into account the vector of scores. A related line of research involving Knowledge Engineering for Planning is the extraction of more informative features from the planning problems. Finally we intend to investigate the effect of a weighted distance kNN approach.

ACKNOWLEDGEMENTS

This work is partly funded from the eCONTENT FP5 European Programme under the EUROCITIZEN project, contract No. 22089. Grigorios Tsoumakas is supported by an ECCAI Travel Award.

REFERENCES

[1] D. Aha, D.W. Kibler, and M.K. Albert, 'Instance-based learning algorithms', *Machine Learning*, **6**, 37–66, (1991).

[2] J. Carbonell, C. A. Knoblock, and S. Minton, 'PRODIGY: An integrated architecture for planning and learning', in *Architectures for Intelligence*, ed., K. VanLehn, pp. 241–278. Lawrence Erlbaum Associates, (1991).

[3] E. Fink, 'How to solve it automatically: Selection among problem-solving methods', in *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems, AIPS98*, pp. 128–136, (1998).

[4] A. Gerevini and I. Serina, 'LPG: a Planner based on Local Search for Planning Graphs with Action Costs', in *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems, AIPS02*, pp. 13–22, (2002).

[5] C.L. Hwang and K. Youn, *Multiple Attribute Decision Making - Methods and Applications: A State of the Art Survey*, Springer-Verlag, New York, USA, 1981.

[6] S. Minton, 'Automatically Configuring Constraint Satisfaction Programs: A Case Study', *Constraints*, **1**(1/2), 7–43, (1996).

[7] M. Robnik-Sikonja and I. Kononenko, 'An adaption of Relief for attribute estimation in regression', in *Proceedings of 14th International Conference on Machine Learning, ICML97*, pp. 296–304, (1997).

[8] G. Tsoumakas, D. Vrakas, N. Bassiliades, and I. Vlahavas, 'Using the k nearest problems for adaptive multicriteria planning', in *Proceedings of the 3rd Hellenic Conference on Artificial Intelligence, SETN04*, pp. 132–141, Samos, Greece, (2004).

[9] R. Vilalta and Y. Drissi, 'A perspective view and survey of meta-learning', *Artificial Intelligence Review*, **18**(2), 77–95, (2002).

[10] D. Vrakas. The Highly Adjustable Planner (HAP). http://lpis.csd.auth.gr/systems/hap/, September 2002.

[11] D. Vrakas, G. Tsoumakas, N. Bassiliades, and I. Vlahavas, 'Learning rules for Adaptive Planning', in *Proceedings of the 13th International Conference on Automated Planning and Scheduling, ICAPS03*, pp. 82–91, Trento, Italy, (2003).

[12] D. Wettschereck, D.W. Aha, and Mohri T., 'A Review and Empirical Analysis of Feature Weighting Methods for a Class of Lazy Learning Algorithms', *Artificial Intelligence Review*, **11**, 273–314, (1997).