Using the k Nearest Problems for Adaptive Multicriteria Planning

Grigorios Tsoumakas, Dimitris Vrakas, Nick Bassiliades, and Ioannis Vlahavas

Department of Informatics, Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{greg, dvrakas, nbassili, vlahavas}@csd.auth.gr
http://lpis.csd.auth.gr/

Abstract. This paper concerns the design and development of an adaptive planner that is able to adjust its parameters to the characteristics of a given problem and to the priorities set by the user concerning plan length and planning time. This is accomplished through the implementation of the k nearest neighbor machine learning algorithm on top of a highly adjustable planner, called HAP. Learning data are produced by running HAP offline on several problems from multiple domains using all value combinations of its parameters. When the adaptive planner HAP_{NN} is faced with a new problem, it locates the k nearest problems, using a set of measurable problem characteristics, retrieves the performance data for all parameter configurations on these problems and performs a multicriteria combination, with user-specified weights for plan length and planning time. Based on this combination, the configuration with the best performance is then used in order to solve the new problem. Comparative experiments with the statistically best static configurations of the planner show that HAP_{NN} manages to adapt successfully to unseen problems, leading to an increased planning performance.

1 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 problem, when the value of one or more parameters of the planner is 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 given problem.

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.

Some promising past approaches towards this goal, followed the methodology of utilizing Machine Learning in order to infer rules for the automatic configuration of planning systems [1],[2]. However, these approaches exhibited two important problems. The first one is that they used a fixed policy for what can be considered as a good solution to a planning problem and didn't allow users to specify their own priorities concerning the speed of the planner and the quality of the plans, which are frequently contradictious. The second one is that learning is very computationally expensive and thus extending the knowledge base of the planner is a non-trivial task.

This paper presents a different approach to adaptive planning that is based on instance-based learning in order to deal with the two aforementioned problems. Specifically, the k nearest neighbor machine learning algorithm is implemented on top of the HAP highly adjustable planner. Learning data are produced by running HAP offline on 30 problems from each one of 15 domains (i.e. 450 problems) using 864 combinations of values for its 7 parameters. When the adaptive planner HAP_{NN} is faced with a new problem, it retrieves the steps and time performance data for all parameter configurations of the k nearest problems and performs a multi-criteria combination, with user-specified weights. The best configuration is then used for running the planner on the new problem. Most importantly, the planner can store new problems and train incrementally from them, making the system highly extensible.

The performance of HAP_{NN} was thoroughly evaluated through experiments that aimed at showing the behavior of the adaptive system in new problems. The results showed that the system managed to adapt quite well and the use of different weights for steps and time had the expected effect on the resulting plan length and planning time of the adaptive planner.

The rest of the paper is organized as follows: Section 2 overviews related work combining Machine Learning and Planning. The planning system used for the purposes of our research and the problem analysis done for deciding the problem attributes are presented in Section 3 and 4 respectively. Section 5 describes in detail the methodology we followed for designing the adaptive planner. The experimental results are presented and discussed in the Section 6 and finally, Section 7 concludes the paper and poses future research directions.

2 Related Work

Machine learning has been exploited extensively in the past to support Planning systems in many ways. There are three main categories of approaches based on the phase of planning that learning is applied to and the consequent type of knowledge that is acquired.

Domain knowledge is utilized by planners in pre-processing phases in order to either modify the description of the problem in a way that will make it easier for solving it or make the appropriate adjustments to the planner to best attack the problem [1].

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 [3].

Finally, optimization knowledge is utilized after the production of an initial plan, in order to transform it in a new one that optimizes certain criteria, e.g. number of steps or resources usage [4].

A concise survey of related work on learning-powered adaptive planners can be found in [2]. Furthermore, a very detailed and analytical survey of past approaches on Machine Learning and Planning has been presented in [5].

3 The HAP Planner

The proposed methodology has been applied to HAP (Highly Adjustable Planner), a customizable planning system, embodying the search modules of the BP planner [6], the heuristics of AcE [7] and several add-ons that improve the speed and the accuracy of the planner. The customization of the system is feasible through the 7 planning parameters, outlined in Table 1, which can be set by the user.

The first one refers to the planning direction, which can be either backward (0) or forward (1). The second parameter allows the user to select one of the three available heuristic functions in order to use it as a guide during the search. The third parameter sets the values for the weights used during planning in the weighted $A*$ search technique. The fourth parameter sets the penalty put on states violating pre-computed fact orderings, while the next one sets the size of the planning agenda (maximum number of states in the frontier set). The last two parameters enable or disable techniques for overcoming plateaus in the search space and simplifying the definition of subproblems, respectively. More details about the planning parameters and their possible setups can be found in [2].

Table 1. The seven planning parameters and their valuesets

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}

4 Problem Characteristics

The purpose of this research effort was to discover 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 features that could influence the choice of parameters of HAP.

Our main concern was to select 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 from the PDDL files, which are the default input to planning systems, and their values can be calculated during the standard parsing process. We also included a small number of attributes which are 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 but the overhead is negligible compared to the total planning time.

A second concern which influenced the selection of attributes was the fact that the attributes 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 whether it is a large problem or a not. This is a general solution followed in all situations where a problem attribute depends nearly linearly on the size of the problem.

Taking all the above into consideration we resulted in a large set of 35 measurable characteristics, which can be divided in three categories: The first category refer 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). The last category contains attributes that can be instantiated after the calculation of the heuristic functions, such as the estimated distance between the initial state and the goals. The list of the attributes and a more detailed analysis on their purpose can be found in [2].

5 The $\mathbf{HAP_{NN}}$ Adaptive Multi-criteria Planner

$\mathrm{HAP_{NN}}$, is an extension of HAP that implements the k Nearest Neighbor (kNN) machine learning algorithm in order to learn the necessary knowledge for autotuning its planning parameters to best fit the morphology of each planning problem. This section presents the process of preparing the learning data for the kNN algorithm, the adaptation functionality of the planner when faced with a new problem and its offline incremental training capability.

5.1 Preparing the Training Data

Training data were produced by running the HAP planner on 450 planning problems (30 problems from each one of 15 domains) using all 864 combinations of values for its 7 planning parameters. For each run of HAP, we recorded the features of the problem, the performance of the planner (steps of the resulting plan and required planning time) and the configuration of parameters. This process is illustrated in Figure 1.

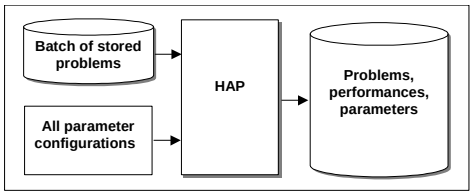

Fig. 1. Preparing the training data

The training data were 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.

One issue that had to be dealt is how to record the cases where HAP failed to find a solution due to memory or time limitations. Note here that an upper limit of 60 seconds was imposed on all runs of the planner. In such cases a special number (999999), was recorded for both plan steps and planning time.

5.2 Online Planning Mode

Given a new planning problem, HAP_{NN} first calculates the values of the problem characteristics. Then the kNN algorithm is engaged in order to retrieve the ids of the k nearest problems from the problems file. k is an input parameter of HAP_{NN} whose default value is set to 7 (see section 6.1). In the implementation of kNN 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 2-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.

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 if the plan isn't very good. Since, these two criteria (fast planning, short plans) are contradicting, it is up to the domain expert to set up his/her priorities. HAP_{NN} has the advantage of letting the user express his/her 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 straightforward Weighted Average method is used to obtain an overall score from steps and time. This requires the normalization of the criteria. For each problem and planner configuration, we normalize time and steps 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..k) and planner configuration j (j=1..864).
- First, we find the shortest plan and minimum planning time for each problem among the tested planner configurations:

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

- Then, we 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 had not managed to find a solution, the normalized values of steps and time were set to zero.

$$S_{ij}^{norm} = \begin{cases} 0 & \text{if } S_{ij} = 999999 \\ \frac{S_i^{min}}{S_{ij}} & \text{otherwise} \end{cases} \qquad T_{ij}^{norm} = \begin{cases} 0 & \text{if } T_{ij} = 999999 \\ \frac{T_i^{min}}{T_{ij}} & \text{otherwise} \end{cases}$$

- Subsequently HAP_{NN} calculates an overall score as the average of the normalized criteria weighted by the user-specified weights:

$$Score_{ij} = ws * S_{ij}^{norm} + wt * T_{ij}^{norm}$$

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, HAP uses the parameter configuration j (j=1..864) with the largest C:

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

The whole process for the online planning mode of HAP_{NN} is depicted in Figure 2. 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.

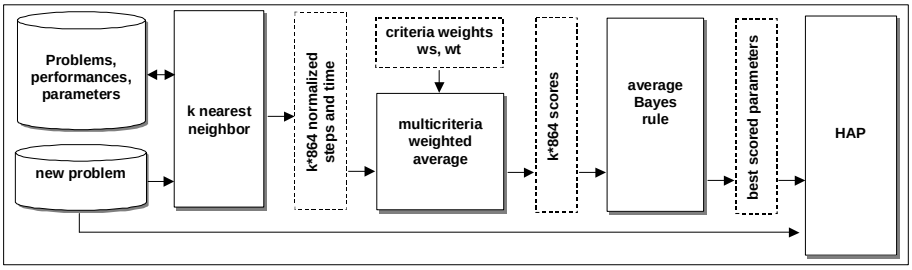

Fig. 2. Online planning mode

The computational cost of training the HAP_{NN} planner is zero, as no training is involved in lazy learning approaches such as the kNN algorithm. However, there is some cost involved during classification, which is however negligible (1 second on a typical Pentium III system at 1Ghz), and can be reduced using a suitable data indexing structure. In contrast, past rule learning approaches [1], [2] exhibit a very large training time (a few hours on a typical Pentium III system at 1Ghz) and a negligible classification time (20 milliseconds on a typical Pentium III system at 1Ghz). Our approach sacrifices a small amount of response time, but gains tremendously in training performance. This way it solves the impractical problems of rule learning approaches, like incremental training and training with user-specified weights for steps and time.

5.3 Offline Incremental Mode

HAP_{NN} can be trained incrementally with each new planning problem that arises. Specifically, the planner stores each new planning problem that it examines, 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 as before.

The incremental training capability is an important feature of HAP_{NN}, stemming from the use of the kNN 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. In comparison, rule-based adaptive planning approaches, require the recomputation of the rule-base, which is a computationally expensive task.

6 Experimental Results

The experiments presented here focus at evaluating the generalization of the adaptive planner's knowledge to new problems and the effect of the weight settings to the resulting plan length and time. These issues are discussed in the following subsections.

For the purpose of the experiments all the runs of HAP 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.

6.1 Evaluating the adaptation of the planner

Examining the problem of learning to adapt HAP to new problems from the viewpoint of a machine learner we notice that it is quite a hard problem. The number of available problems (450) is small, especially compared to the number of problem attributes (35). Since the training data were limited, a proper strategy should be followed for evaluating the planner performance.

For the above reason, we decided to perform 10-fold cross-validation. We 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 (405 problems) were merged into a training set. The training set was used for finding the k nearest problems, and the test set for measuring the adaptive planner's performance. Specifically, we calculated the sum of the average normalized steps and time. In order to evaluate the learning approach, we calculated the same metric for all 864 static planner configurations based on the training set and chose the one that performs best for comparison on the test set. This is even better than having an expert choose the default parameter configuration for the planner. We also calculated the same metric with the best configuration that an "oracle" adaptive planner could achieve if it would always use the best configuration on the test set. 3 sets of weights were used at each run: a) ws=1, wt=1, b) ws=2, wt=1 and c) ws=1, wt=2. The results of each run, were averaged and thus a proper estimation was obtained, which is presented in Figure 3.

We notice that for all sets of weights and all numbers of nearest neighbors the adaptive planner exceeded the best static planner configuration. The average difference for all three settings and for the best average adaptive planner (k=7) was 0.274 which can be translated as an approximate 14% average gain combining both steps and time. If we notice the performance of the oracle planner we

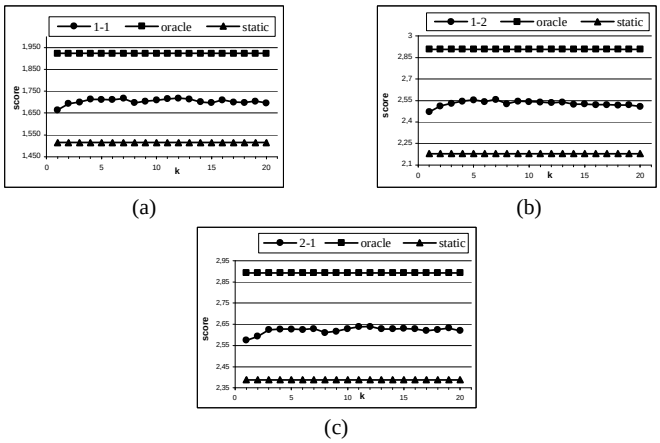

Fig. 3. Average score of static, adaptive and oracle HAP for a) w_s=1 and w_t=1, b) w_s=1 and w_t=2, and c) w_s=2 and w_t=1

can see that the adaptive planner has still the potential to improve with the use of more training problems, but it managed to reach approximately half the gain in performance of an "oracle" planner.

6.2 Evaluating the effect of weights

In order to evaluate the effect that the change of weights have in the resulting plans we produced the graphs depicted in Figure 4 that show the average normalized steps and time respectively for the three different weight settings.

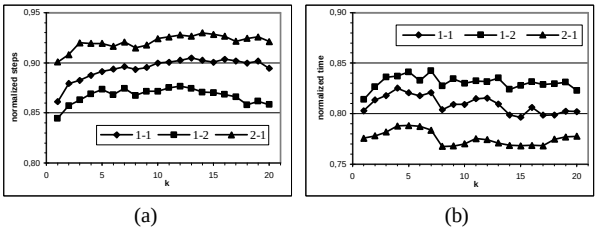

Fig. 4. Average normalized steps (a) and time (b) for three different weight settings

Figure 4a shows that giving more weight to steps (2-1), reduces the average steps of the adaptive planner, in comparison with giving equal weights to both steps and time (1-1). In addition giving more weight to time, further increases the steps in comparison to equal weight setting. Similar conclusions can be drawn from Figure 4b, which concerns planning time. These graphs empirically show that tuning the weights has the user-desired effect on the planner behavior.

7 Conclusions and Future Work

This work has presented a methodology for multicriteria adaptive planning, using the k nearest neighbor algorithm on top of a highly adjustable planner. The planner consults past runs on similar problems and selects the most promising configuration. The results show that the planner manages to adapt quite well to new problems. One very interesting aspect is the capability of the planner to also adapt to user preferences. The priorities of users for steps and time are quantified through two respective weights. Experimental results show that the use of weights results to tuning the planner towards the preferences of the users.

In the future we intend to explore the performance of the proposed methodology various other interesting learning problems for the planning community, like learning from a single domain, learning from easy problems of a domain and adapting to unknown domains. We will also investigate the exploitation of feature selection and weighting techniques to enhance the performance of the kNN algorithm. It is widely known that kNN is prone to irrelevant attributes and the large dimensionality of our problem (35) with respect to the small training set (450) may give rise to overfitting and reduce the potential performance of our methodology.

Acknowledgements

This work is partly funded from the eCONTENT FP5 European Programme under the EUROCITIZEN project, contract No. 22089.

References

1. Vrakas, D., Tsoumakas, G., Bassiliades, N., Vlahavas, I.: Learning rules for Adaptive Planning. In: Proceedings of the 13th International Conference on Automated Planning and Scheduling, Trento, Italy (2003) 82–91
2. Vrakas, D., Tsoumakas, G., Bassiliades, N., Vlahavas, I.: Rule Induction for Automatic Configuration of Planning Systems. Technical report, Dept. of Informatics, Aristotle University of Thessaloniki (2003)
3. Carbonell, J., Knoblock, C.A., Minton, S.: PRODIGY: An integrated architecture for planning and learning. In: Architectures for Intelligence. Volume K. VanLehn, ed. Lawrence Erlbaum Associates (1991) 241–278
4. Ambite, J., Knoblock, C., Minton, S.: Learning Plan Rewriting Rules. In: Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling Systems, AAAI Press (2000) 3–12
5. Zimmerman, T., Kambhampati, S.: Learning-Assisted Automated Planning: Looking Back, Taking Stock, Going Forward. AI Magazine **24** (2003) 73–96
6. Vrakas, D., Vlahavas, I.: Combining progression and regression in state-space heuristic planning. In: Proceedings of the 6th European Conference on Planning. (2001) 1–12
7. Vrakas, D., Vlahavas, I.: A heuristic for planning based on action evaluation. In: Proceedings of the 10th International Conference on Automated Planning and Scheduling. (2002) 61–70