Towards Adaptive Heuristic Planning through Machine Learning

Dimitris Vrakas, Grigorios Tsoumakas, and Ioannis Vlahavas

Dept. of Informatics, Aristotle University of Thessaloniki, 54006 Thessaloniki, Greece
{dvrakas, greg, vlahavas}@csd.auth.gr

Abstract. In domain independent heuristic planning there is a number of planning systems with very good performance on some problems and very poor on others. Few attempts have been made in the past to explain this phenomenon. In this paper we use machine learning techniques to discover knowledge hidden in the dynamics of the planning process that would relate specific characteristics of a planning problem with specific properties of a planning system that lead to good or bad performance. By this, we aim at shedding light to some of the dark areas of heuristic planning and develop an adaptive planner that would be able to optimize its configuration according to the problem at hand.

1 Introduction

In domain independent heuristic planning there are a number of planning systems that perform some better and some worse on a number of toy and real-world planning domains. However few of them have really explained why and how they work well on some cases and bad on others.

To answer these questions we employ machine learning techniques and attempt to discover this knowledge from planning data. The benefits of such an approach are dual: i) There is the prospect of discovering interesting knowledge about the field of domain independent heuristic planning, that could assist in the construction of better planning systems ii) If such knowledge exists, an adaptive planner can be constructed that would tune its parameters depending on the characteristics of the problem at hand and thus outperform a single planner on average for a variety of planning domains and problems.

Trying to learn knowledge from data of a scientific field with a strong experimental facet like that of heuristic planning, is also very interesting from the view of a machine learner. The application domain is in this case an evolving research field, rather than an established business model with clearly defined properties. Trying to discover knowledge in such a domain involves a lot of risk and uncertainty.

Therefore, we decided to follow as close as possible a standard process for data mining, in order to avoid common pitfalls and minimize the probability of venturing towards the wrong direction in the first place. A well known standard is the Cross Industry Standard Process for Data Mining (CRISP-DM) [1]. We

followed a simplified edition of the phases of CRISP-DM version 1.0, adapted to our data mining project.

The rest of the paper is organized as follows. Section 2, presents related work on the use of Machine Learning in Planning. Section 3 discusses the details of the methodology we followed for collecting and preparing the data for the learning process. Section 4 describes the actual data mining task along with comments on obtained results. Finally section 5 concludes the paper and poses future research directions.

2 Related Work

Machine Learning has been exploited in the past in Planning, mainly in order to learn control rules. The PRODIGY Architecture [8] 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.

Approaches towards exploiting domain and problem characteristics in a pre-planning phase have been presented in the past by Fox and Long [6], [3]. They have also done a lot of research in the area of state analysis and it's use by automated planning systems, such as STAN [5] and Hybrid STAN [2].

Hoffman [4] discusses the matter of when a specific planner will behave well and when not by performing domain analysis. He created a 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.

3 Data Collection and Preparation

We need data about characteristics of planning domains and problems, along with data about the parameters of a planner and its performance. These initial data are collected by i) a process of domain and problem analysis by the planning experts and ii) the development of an adjustable planner.

3.1 Planning Domain and Problem Analysis

In order to decide what characteristics of a planning domain are important and could influence the choice of parameters of a planning system, we carried out an analysis that resulted in seventeen measurable characteristics, which can be easily extracted from the operators file. These are summarized in Table 1.

We further performed an analysis at the finer level of planning problem. This led to 34 measurable characteristics of each problem, which are summarized in Table 2.

Table 1. Domain model attributes

Name	Meaning
num_pred	Number of predicates
avg_par_pred	Average number of parameters per predicate
std_par_pred	Standard deviation of parameters per predicate
dyn_stat_pred	Number of dynamic predicates divided by total number of predicates (static and dynamic). A predicate is dynamic, if it appears in the add or delete list of at least one operator.
num_op	Number of operators
avg_par_op	Average number of parameters per operator
std_par_op	Standard deviation of parameters per operator
avg_prec_op	Average number of preconditions per operator
std_prec_op	Standard deviation of preconditions per operator
avg_dyn_pred	Average ratio between dynamic and total preconditions
std_dyn_pred	Standard dev. of ratio between dynamic and total preconditions
avg_eff_op	Average number of add effects per operator
std_eff_op	Standard deviation of add facts per operator
avg_del_op	Average number of del effects per operator
std_del_op	Standard deviation of delete facts per operator
avg_imp_op	Average number of implied facts per operator. A fact f is implied by an operator O, if it exists in the operator's delete list, e.g. $f \in prec(O) \bigcap del(O)$.
std_imp_op	Standard deviation of implied facts per operator

3.2 Highly Adjustable Planner

We developed a highly adjustable planning platform, called HAP, which is customizable by the user through a number of parameters. These concern the type of search, the quality of the heuristic and several other characteristics that affect the planning process. The HAP system is based on the BP planning system [9] and uses an extended version of the ACE heuristic [10].

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 e.t.c., there are implementations for both directions. The *direction* of search is the first parameterized characteristic of HAP used in tests, with the following acceptable values: a) 0 (Regression or Backward chaining) and b) 1 (Progression or Forward chaining).

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, 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 five different assignments for the weights: a) w_1=1 and w_2=0, b) w_1=1 and w_2=1, c) w_1=2 and w_2=1, d) w_1=3 and w_2=1 and e) w_1=4 and w_2=1.

Table 2. Problem Attributes

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 the number of facts per predicate
A10	Average number of actions per operator
A11	Standard deviation of the number of actions per operator
A12	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
A13	Standard deviation of the number of mutual exclusions per fact
A14	Ratio between useless and total facts in the initial state. A fact of the initial state is useless if it can be safely removed without affecting the planning process
A15	Number of orderings among the goals of the problem. An ordering between goals g1 and g2 exists (denoted as ob(g1,g2)), if goal g1 must be achieved before g2
A16	Ratio between number of goal orderings and total number of goals
A17	Average distance of all actions for the forward direction
A18	Standard deviation of distances of all actions for the forward direction
A19	Estimated actions needed to reach goals starting from the initial state moving forward
A20	Branching factor of the initial state for the forward direction
A21	Number of orderings among the facts of the initial state. These are similar to the goal orderings but are used by regression planners
A22	Ratio between number of orderings between facts of the Initial state and the total number of facts in the initial state
A23	Average distance of all actions for the backward direction
A24	Standard deviation of distances of all actions for the backward direction
A25	Estimated number of actions needed to reach the nitial state starting from the goals moving backwards
A26	Branching factor of the goals for the backward direction
A27	A15-A21
A28	A16-A22
A29	A17-A23
A30	A18-A24
A31	A19-A25
A32	A20-26
A33	A19/A25
A34	A20/A26

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 ∞, 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 four different settings for the size of the agenda: a) 1, b) 10, c) 100 and d) 1000.

The OB and OB-R functions introduced in BP and ACE respectively, are also adopted by HAP in order to search the states of the search 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, returned by the heuristic function, is increased by *violation_penalty*, a constant number defined 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.

One of the most important problems that a planner faces when it regresses the goals of the problem either during the search or during the computation of the heuristic function is the fact that the goals of the problems usually form a set of states, rather than a complete state specification. Regression planners, usually deal with this problem by relaxing the regressibility criteria, since each state in the space may contain facts that are not explicitly defined. For planners constructing their heuristic function backwards, there are two ways to overcome this problem: they either use the same idea as the regression planners, i.e. they relax their regression criteria, or they enrich the goal state with additional facts that are not mutual exclusive with the goals. Both these techniques are supported by HAP and it is up to the user to select whether the goals should be enriched by configuring the *goal_enrichment* parameter. Acceptable values for this parameter are: a) 0 (goals are not enriched) and b) 1 (goals are enriched for heuristic construction).

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:

$$dist(A) = \begin{cases} 1 & \text{if prec}(A) \subseteq I \\ 1 + \sum_{X \in MPS(prec(A))} dist(X) & \text{otherwise} \end{cases}$$

where MPS(S) returns a set of actions, with near minimum accumulated cost, achieving S.

```
Function MPS(S)
Input: set of facts S
Output: set of actions achieving S with near min accumulated dist
Set G = ∅
Set 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 maximizes val
    Set G = G ⋃ A'
    Set S = S − add(A') ⋂ S
Until S = ∅
Return G
```

Apart from the initial heuristic function described above, HAP embodies two variations, which in general, are finer grained. The general idea behind these variations, lays 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, among 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_add list in the MPS function instead of the add list but only in the second part of the function, which updates state S. So the command $SetS = S - add(A') \bigcap S$ is updated to $SetS = S - enriched\_add(A') \bigcap S$.

The second variation of the heuristic function pushes the above ideas one step further. The enriched_add list is not only used, instead of add list, for updating state S, but also in the first part of function MPS, which ranks the candidate actions. So additionally, it updates the command $val(A) = dist(A)/|add(A) \bigcap S|$ to $val(A) = dist(A)/|enriched\_add(A) \bigcap S|$.

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.

HAP was run using all possible combinations of the above mentioned parameters for a number of problems of six domains used in the AIPS (Artificial Intelligence Planning and Scheduling) conference planning competitions. These domains are: gripper from AIPS-98, blocks world, logistics and mic-10 from AIPS-00 and zeno and driverlog from AIPS-02. The time limit which we imposed on HAP was 60 seconds. If no solution was found within 60 seconds, then HAP stopped searching and continued with the next problem. For each of these runs a record of data regarding HAP's parameters, the problem's characteristics and the domain's characteristics along with the resulting plan steps and time to be completed was produced. A dash was used in place of the number of steps and time to indicate that a plan was not found within the time limit.

3.3 Data Preparation

The data regarding the number of steps and the actual time that HAP took to find each solution is not really very useful to us. What we need is a way to discriminate between good and bad plans. Furthermore, the time to find a solution is not a machine independent measure. This brings up another problem, since we ran our experiments on several different workstations in parallel due to the increased computational power that was required.

To deal with the above issues we decided to normalize the original *steps* and *time* attributes by dividing them with the minimum steps and time respectively for each problem. These transformed attributes *norm_steps* and *norm_time* are independent of the machine used to get the measurements. In addition they are independent of the actual values of *steps* and *time*, rather they emphasize on the relation of these values with the best value for each problem. Therefore, they can be safely used for discrimination between good and bad plans.

Now that we have an objective measure of the quality of plans with respect to steps and time, we can proceed by categorizing the records of HAP executions into two discrete classes, *good* and *bad*. The normalized versions of steps and time attributes are further transformed into the attributes *steps_quality* and *time_quality* according to the following rules: a) If *norm_steps* is less or equal to 1.1 then assign the value *good*, otherwise assign the value *bad* and b) If (*norm_time*) is less or equal to 1.5 then assign the value *good*, otherwise assign the value *bad*. For example, if the smallest number of steps is 10 and the smallest time is 0.5 seconds then a plan with 11 steps that was found in 1.5 seconds has a value of *good* for *steps_quality* and a value of *bad* for *time_quality*.

A noteworthy issue with respect to data cleaning is that some of the executions of HAP finished after the imposed time limit of 60 seconds without finding a plan. This is a usual situation for some hard planning problems. In these cases the initial data contained a dash in place of the number of steps and amount of time. These records were cleaned by removing them from the data set that would be used for inducing knowledge about the steps and by assigning the value of *bad* to the *time_quality* attribute in the data set that would be used for inducing knowledge about the time.

The next step was to integrate the data about domain characteristics and the data about problem characteristics and planner parameters using a full join based on the domain attribute of each problem. This led to a huge data set consisting of 63 attributes (domain_name, problem_name, 17 domain characteristics, 34 problem characteristics, 8 planner parameters, steps_quality, time_quality).

From these attributes we excluded the name of the planning domain and problem. Although knowledge regarding specific domains could potentially be of interest to planning experts, we decided to focus on the larger picture of inducing a general model that would be applicable to any unseen domain.

4 Modelling and Evaluation

We selected the modelling technique of decision trees, and more specifically the J48.PART algorithm, which is a variation of the state-of-the-art c4.5rules [7] program implemented in Java within the WEKA [11] toolkit. The reason for choosing this technique was the initial requirement of interpretability of the model. Decision trees and especially rules extracted from decision trees offer knowledge in a format that is comprehensible and intuitive to people.

In order to obtain accurate estimates of the model quality, we decided to use cross-validation. However, as data were limited in terms of different domains and problems, 10-fold cross-validation would leave too few data out for testing. Therefore, we decided to use 4-fold cross-validation in order to have 25% of data available for testing at each fold. The experiments were run on a Pentium 3 at 1Ghz and 256Mb of RAM allocated to Java. It took about 45 minutes to build the model from the first fold and another 3*45 minutes to finish the evaluation.

Using the 73795 instances of HAP executions and the *steps_quality* as the target attribute J48.PART gave 619 rules and an average accuracy of 96%. We present some of the most important rules here along with comments of experts:

1. IF $num\_op \leq 3$ AND $A30 > 0.56$ AND $equal\_estimation = 0$
   THEN $steps\_quality = good$
   Comments: Although we were initially surprised by this rule, with a second thought it seemed reasonable, but yet difficult to explain. With a closer look at the semantics of A30 and the tendency of people to encode domains, with forward chaining in their minds, we speculate that a positive number in A30 indicates nevertheless that the heuristics are quite informative and therefore the search algorithm should "trust" it till the end.
2. IF $A8 \leq 7.6$ AND $A4 \leq 9$ AND $direction = 0$ AND $w_2 = 1$
   THEN $steps\_quality = good$
   Comments: It is known that A^* returns shorter plans, in general, than Best first. However, if the problem is too complex then with A^* the planner may not be able to find a solution at all, due to time and memory limitations. This rule takes these into account and suggests A^* only when the problem is not very complex. As far as the direction is concerned, it was known from our experience that backward search often results in shorter plans, when the domain description is not biased against it.

3. a) IF $A6 > 350$ AND $A18 < 1.9$ AND $goal\_enrichment = 0$ AND ($w_1 = 2$ OR $w_1 = 3$) THEN $steps\_quality = good$
   b) IF $A6 > 350$ AND $A18 < 1.9$ AND $goal\_enrichment = 1$ THEN $steps\_quality = bad$
   Comments: From our experiments it was clear that when we chose to enrich the goals, the heuristic function tended to become less informative, especially when the graph built by the heuristic was quite short. The standard deviation of the estimations given by the heuristic is closely related to the length of the graph.
4. IF $A3 \leq 211$ AND $A6 > 44$ AND $A7 > 676$ AND $A13 > 2.4$ AND $A29 \leq 0.62$ AND $direction = 0$ THEN $steps\_quality = bad$
   Comments: This can also be explained by our comment for the previous rule, since complex domains are usually designed in a way that votes for progression planners.

Using the 73795 instances of HAP executions and using the *time_quality* as the target attribute J48.PART gave 566 rules and an average accuracy of 97%. We present some of the most important rules here along with comments of experts:

1. IF $A30 > 1.67$ AND $direction = 0$ THEN $time\_quality = good$
   Comments: Once again $A30 > 0$ indicates that the heuristics are quite informative and it is not very difficult to find a solution. What's more, a large number for A30 may also indicate the existence of paths that start from the initial state and head quite away from the goals, which is clearly a reason for choosing backward chaining.
2. IF $A23 \leq 2.7$ AND $A27 > -4$ AND $A31 \leq -4$ AND $w_2 = 0$ AND $direction = 0$ THEN $time\_quality = good$
   Comments: The above indicate that the heuristic for backward chaining is more informative and it can make better use of orderings so it will reach a solution quite quickly.
3. IF $avg\_par\_op > 2$ AND $num\_pred \leq 9$ AND $A2 > 8$ AND $A19 > 6$ AND $A29 > 0.3$ AND $w_1 = 1$ AND $w_2 = 1$ THEN $time\_quality = bad$
   Comments: As stated before, A^* is not generally a wise choice when our first priority is speed. A^* returns better plans but it needs more time since it has to perform more search. Especially in big and complex domains it is almost impossible to find a solution in a reasonable amount of time.

5 Conclusions

This paper reported on ongoing work, which aims at using machine learning techniques in order to explain why all the heuristic planners behave well in certain problems and not well in others. More specifically, it tries to find the hidden relations between the characteristics of the problems and the parameters of planning.

To accomplish this, we used a data-mining tool on a very large data set created by instances of execution of a highly adjustable planner. The outcome

was over a thousand of high quality rules connecting measurable characteristics of each problem with proper and improper configurations of the planning system.

It is in our direct future plans to refine the data mining methodology we used, and investigate the applicability of the resulting model, by encoding it in a planning system. This will result in a planning system able to automatically fine-tune itself to the specific characteristics of each problem.

Acknowledgements

This project has been partially supported by SUN Microsystems, grant number EDUD-7832-010326-GR.

References

1. P. Chapman, J. Clinton, T. Khabaza, T. Reinartz, and R. Wirth. The crisp-dm process model. http://www.crisp-dm.org, 1999.
2. Maria Fox and Derek Long. Hybrid stan: Identifying and managing combinatorial sub-problems in planning. In *Proceedings of the UK Planning and Scheduling SIG Workshop*, December 2000.
3. Maria Fox, Derek Long, Steven Bradley, and James McKinna. Using model checking for pre-planning analysis. In *Proceedings of the AAAI Symposium on Model-based Validation of Intelligence*, March 2001.
4. J. Hoffman. Local search topology in planning benchmarks: An empirical analysis. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Seattle, Washington, USA, August 2001.
5. Derek Long and Maria Fox. Efficient implementation of the plan graph in stan. *JAIR*, 10:87–115, 1998.
6. Derek Long and Maria Fox. Automatic synthesis and use of generic types in planning. Technical report, 1999.
7. Ross J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo, 1993.
8. Manuela Veloso, Jaime Carbonell, Alicia Perez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.
9. Dimitris Vrakas and Ioannis Vlahavas. Combining progression and regression in state-space heuristic planning. In *Proceedings of the 6th European Conference on Planning*, pages 1–12, 2001.
10. Dimitris Vrakas and Ioannis Vlahavas. A heuristic for planning based on action evaluation. In Donia Scott, editor, *10th International Conference on Artificial Intelligence: Methodology, Systems, Applications. LNAI 2443*, pages 61–70. Springer-Verlag, 2002.
11. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.