Learning heuristic functions for cost-based planning

Jesús Virseda and **Daniel Borrajo** and **Vidal Alcázar**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. 28911 Leganés (Madrid). Spain
jvirseda@pa.uc3m.es, dborrajo@ia.uc3m.es, valcazar@inf.uc3m.es

Abstract

In the last International Planning Competition (IPC 2011), the most efficient planners in the satisficing track were planners that used unit-cost heuristics. These heuristics ignore the real cost of the actions and return instead an estimate of the plan length to the goal. The main advantage of these heuristics compared with real-cost heuristics is that they solve a greater number of problems (also known as coverage), which has a high impact on the IPC score. However, *a priori* heuristics that predict the real cost should find solutions of better quality. To increase the effectiveness of real-cost heuristics and reduce the impact of their drawbacks without losing quality, we study the use of machine learning techniques to automatically obtain good combinations of those heuristics per domain. In particular, regression techniques are used to predict the real cost from any state to the goal. We use the heuristic estimations and the real costs obtained from solving easy problems as attributes. Later, we feed those instances to several machine learning techniques to obtain prediction models. All learned models approximate the real value with high correlation. Then, we implemented the most suitable model in a planner and evaluated it on harder problems. With this new planner we can solve 56 more problems than using the best real-cost heuristics for each domain separately. Our approach is also better regarding solution quality.

Introduction

In the last IPC[1] (2011), the approach followed by most planners was heuristic forward search. Heuristic planners search in a state space guided by one or more heuristic functions. Heuristic functions can take into account the real cost of actions or assume that all actions have unitary cost. The former are real-cost heuristics, whereas the latter are unit-cost heuristics. Overall, real-cost heuristics find solutions of better quality and unit-cost heuristics find solutions expanding fewer nodes. Therefore, unit-cost heuristics often solve more problems under time and memory constraints.

A common solution to the downfalls of both kinds of heuristics is using anytime schemes that employ both types of heuristics. Generally, the first solution is found using unit-cost heuristics with greedy search algorithms and subsequent solutions are found using real-cost heuristics with more conservative algorithms (Richter and Westphal 2010).

[1] http://ipc.icaps-conference.org/

Here we focus on improving the efficiency of real-cost heuristics. We are inspired by the work of Arfaee *et al.* (2011). The authors propose learning a new heuristic starting with a very weak one taking into account particular characteristics of the problem and iteratively improving its accuracy. This scheme can be improved since: it must be done per problem, the initial heuristic may be too weak to solve the problem (and thus alternative methods to generate training instances must be employed), and the learning process is often on the order of days. Instead, we learn from existing domain-independent heuristics *per domain*, obtaining training instances from small problems.

In order to minimize the error generated by the use of a single heuristic, we study whether a combination of more than one real-cost heuristic function can be useful to improve the performance of the planner. Given that it is hard to know *a priori* which heuristic combination will work well for each state, problem and domain, we use machine learning techniques. We extract learning instances from solutions to some problems in each domain. The instances will be composed of the values that each domain-independent heuristic returns for each state and the real cost to the goal. Then, we use two approaches based on machine learning techniques to find a useful combination of heuristics. First, we generate a regression model, which will be used later as the new domain-dependent heuristic; it computes at each state the values of several selected heuristics and returns a combination of the former values for that state. Second, we use an attribute selection technique to select a subset of heuristics to be used in an alternating queue, as previous works have shown that this way of combining heuristic estimators is overall very effective (Röger and Helmert 2010). For the experimentation we use Fast Downward (Helmert 2011), a planning framework that implements several state-of-the-art heuristics, and WEKA (Witten and Frank 2005), an environment with multiple machine learning techniques. Our approach is an offline learning technique, as the real cost to achieve the goals must be known beforehand to create the training instances.

The rest of the paper formalizes the planning task, describes our approach, gives the details of the employed components and techniques, presents the experimental results, compares our approach with the related work and puts forward the conclusions and future work.

Propositional Planning

A planning task is defined as a tuple Π=*(S,A,I,G)*, where *S* is a set of atomic propositions, *A* is the set of grounded actions derived from the operators of the domain, $I \subseteq S$ is the initial state, and $G \subseteq S$ is the set of goal propositions. Each action $a \in A$ is defined as a tuple *(pre(a), add(a), del(a), c(a))* (preconditions, add effects, delete effects and cost), such that *pre(a), add(a), del(a)* $\subseteq S$ and $c(a)$ is a fixed positive real-valued cost. In this paper, we only consider satisficing planning; that is, a solution does not have to be optimal with respect to a given metric.

Description of the Approach

Our approach involves two phases: training and testing. The training part is also divided into two parts: gathering the training instances and learning models from them. The training instances are obtained computing the values of a set of heuristics (given as input) and the real cost to the goal of a set of states. The states are those that appear along the solution paths of simple problems solved by different methods. Then, regression models are built using different machine learning techniques. The aim of the models is to predict the real cost of the solution by combining heuristics.

The testing phase implements the chosen regression model in a planner and compares its performance against different state-of-the-art approaches. Different combinations of heuristic functions are studied.

Training

Given a set of training problems P in a domain D, a set of heuristic functions $H = h_1, h_2, \ldots, h_m$ and a machine learning technique L, the training phase returns a regression model R. A regression model $R : T \rightarrow \mathbb{R}$ takes as input a tuple $t(n) = \langle h_1(n), h_2(n), \ldots, h_m(n) \rangle$ and returns a real number, the combined heuristic value of node n according to the regression model R. Each $h_j(n), j = 1..m$ is the heuristic value of heuristic h_j for node n.

We first solve a set of simple problems to obtain training instances for the learning process. We keep the solutions of those problems; in particular, for each state along the solution plan we store the value returned by each heuristic $h_j \in H$ for that node, as well as the cost from that node to the goal according to the computed solution. Suppose $\pi = (a_1, a_2, \ldots, a_n)$ is the solution to a training problem $p \in P$, and $S_\pi = (s_0, s_1, s_2, \ldots, s_n)$ is the set of states in the solution path, such that $s_0 = I$, s_n is a goal state ($s_n \subseteq G$), and a_i is applicable in the state s_{i-1}, generating state s_i. For each state $s_i \in S_\pi$ and for each heuristic $h_j \in H$, we compute $h_j(s_i)$. Also, we compute $c(s_i) = \sum_{k=i}^{n} c(a_k)$. Then, for each $s_i \in S$ of each solution of problems in P, we generate a training instance of the form: $\langle h_1(s_i), \ldots, h_m(s_i), c(s_i) \rangle$.

There are several ways of computing the solutions of the problems during the training phase. Ideally, an optimal planner should be used to ensure that the solution plan is optimal. With an optimal solution plan, the cost to the goal for each state along the solution path is guaranteed to be h^*, the perfect heuristic value using the real actions costs. Using the optimal solution avoids introducing noise in the training instances due to imprecisions in the estimation of the cost to the goal. It is not guaranteed though that using optimal solutions will yield more accurate models; other methods, such as the use of suboptimal planners or random walks from the goal, may also be valid alternatives. This will be explored in Section Experimentation.

Once the training instances are generated, several machine learning techniques are used to compute different regression models. This is done per domain, so there will be several models for each domain. Prior to learning, we perform attribute selection to avoid the use of correlated attributes that may not contribute to the overall process. Finally, we estimate the accuracy and correlation of the models to compare them and select the most suitable one. Algorithm 1 shows how the whole training process is performed.

Algorithm 1: Description of the training process.

```
input  : solving_method, M
         heuristic_set, H
         problem_set, P
         attribute_selection, AS
         learning_technique, L
output: regression_model, R
begin
    instance_set ← ∅;
    foreach problem ∈ P do
        solution_path ← apply(M,problem);
        foreach node ∈ solution_path do
            instance ← compute_instance(node,H);
            instance_set ← instance_set ∪ instance;
    instance_set ←apply(AS,instance_set);
    return R ← apply(L,instance_set);
end
```

Testing

We test our approach in each of the domains used in the training phase. The problems used in the testing phase are more challenging than those used for training. To asses the viability of the learning process, the best regression model in each domain is used as the heuristic function of the planner. In particular, we compare the score of each heuristic against the score obtained by using the learned model as heuristic. This is done both in terms of coverage and IPC score.

Experimental Setting

This section describes the elements involved in the experimentation. This includes the chosen heuristics, the sets of problems employed in the training and testing phases, the methods used to generate the training instances, the machine learning methods and the details of the testing environment.

Heuristic Functions

The following heuristic functions were used in our setting:

Additive heuristic (Add) (Bonet and Geffner 2001) is the sum of the accumulated costs of the paths to the goal

propositions in the relaxed problem (a delete-free version of the problem).

Blind heuristic returns the cost of the cheapest applicable action for non-goal states and 0 for goal states.

Causal graph heuristic (CG) (Helmert 2004) is the sum of the costs of the paths in the domain transition graphs which are necessary to consider to reach the goal propositions.

Context-enhanced additive heuristic (CEA) (Helmert and Geffner 2008) is the causal graph heuristic modified to use pivots that define contexts relevant to the heuristic computation.

Fast Forward heuristic (FF) (Hoffmann 2003) is the cost of a plan that reaches the goals in the relaxed problem (a delete-free version of the problem).

Goal count heuristic is the number of unsatisfied goal propositions.

Landmark count heuristic (LM-Count) (Richter, Helmert, and Westphal 2008) is the sum of the costs of the minimum cost achiever of each unsatisfied or *required again* landmark. Landmarks are computed using the RHW method; disjunctive landmarks were taken into account.

Landmark-cut heuristic (LM-Cut) (Helmert and Domshlak 2010) is the sum of the costs of each disjunctive action landmark that represents a cut in a justification graph towards the goal propositions.

Max heuristic (Bonet and Geffner 2001) is the maximum of the accumulated costs of the paths to the goal propositions in the relaxed problem (a delete-free version of the problem).

Planning Domains and Problem Sets

All the domains in the seventh IPC (2011) have been used in the experimentation. The domains are the following: Barman, Elevators, Floortile, Nomystery, Openstacks, Parcprinter, Parking, Pegsol, Scanalyzer, Sokoban, Tidybot, Transport, Visitall and Woodworking.

Each domain has two sets of problems: those used in the optimal track, and those used in the satisficing track. The problems of the optimal track are designed to be easier than the problems of the satisficing track. Thus, we used the problems in the optimal track of each domain for training, and the problems of the satisficing track of the same domain for testing.

Generation of Training Instances

Initial experiments showed that using optimal solutions does not guarantee more accurate models. Hence, three methods were used to generate the training instances. Each method has been tested in isolation; that is, the set of training instances obtained with each method was used to learn different prediction models.

FDSS optimal solution is the solution found by the optimal version of Fast Downward Stone Soup (FDSS) (Helmert, Röger, and Karpas 2011), winner of the optimal track at IPC'11.

LAMA11 best solution is the solution that LAMA11 (Richter and Westphal 2010), winner of the satisficing track at IPC'11, finds. The solution is not guaranteed to be optimal, although the solutions are expected to be close to the optimal one due to the anytime scheme that LAMA11 uses. The FF and the landmark count heuristics are used during the search process.

Multi-Heuristic First Solution (MHFS) is the first solution found employing all the studied heuristics in the alternation open list implemented by Fast Downward (Röger and Helmert 2010). Besides the choice of heuristics, greedy best-first search with regular evaluation and no preferred operators was used. We selected this scheme because we found interesting to compute the solution paths employing the same heuristics that will be used afterwards as attributes in the learning process.

Machine Learning methods

The machine learning techniques used to compute the models were the following:

Attribute Selection obtains a subset of relevant features. We employed this technique because some heuristics yield very similar values in some domains, so including all of them may not be useful in the learning process. Also, the computation of several heuristics can be expensive, so removing uninteresting or correlated heuristics may increase the performance of the planner. We used Correlation-based Feature Selection (Hall 1998). This attribute selection method is independent from the regression learning method used afterwards.

Regression Analysis is used to compute the prediction models. The following techniques have been used with 10 fold cross-validation:

- **Linear Regression** (LR) models are linear functions that minimize the sum of squared residuals of the model.
- **M5P** (Quinlan 1992) models are regression trees that approximate the value of the class. This method is more flexible than Linear Regression because M5P can capture non-linear relations.
- **M5Rules** (Quinlan 1992) is similar to M5P, but generates rules instead of regression trees.
- **SVMreg** (Shevade et al. 2000) implements Support Vector Machines for regression.

Testing Environment

The learned model was implemented as a novel heuristic in Fast Downward (Helmert 2011). To test our system, the time and memory constraints were the same as in the IPC: 6GB of RAM and 1800 seconds for the execution. The machines the planners were tested on were Linux computers with a 2.93 GHz 64-bits AMD processor and 8GB of RAM. To compute the solutions of the set of training problems a time limit of 600 seconds was used.

Experimentation

This section describes the results obtained in both phases: training and testing. The results in the training phase are used

to compare the accuracy of the different regression models. The results in the testing phase are used to compare the approach against state-of-the-art heuristics.

Results on Training

To obtain the set of training instances, three problem solving methods were proposed: FDSS, LAMA11 and MHFS. The total number of training problems was 280, 20 problems per domain. FDSS solved only 181 problems, whereas LAMA11 and MHFS solved all problems. Since FDSS solved fewer problems, the models obtained with this approach employ fewer training instances.

To analyze the accuracy of the four described regression methods, we show the average correlation and average computation time of the model of each instance-generating method and regression technique over all domains in Table 1. As we can see, all four regression techniques have high accuracy, but Linear Regression is noticeably faster. Accuracy is higher and time is smaller for all methods with the training sets obtained with FDSS. This is to be expected, given that the regression methods can approximate the function more accurately when there are fewer input points to be approximated.

Instance-generating method	Classifier	Correlation	Time(ms)
FDSS	LR	0.9451	73.57
	M5P	0.9505	346.43
	M5Rules	0.9500	497.86
	SVMreg	0.9450	987.14
LAMA11	LR	0.9291	104.29
	M5P	0.9344	566.43
	M5Rules	0.9329	964.29
	SVMreg	0.9207	1,703.57
MHFS	LR	0.9399	104.29
	M5P	0.9495	627.86
	M5Rules	0.9492	1,071.43
	SVMreg	0.9384	2,567.14

Table 1: Average of the correlation and computation time of the models over the 20 domains for each instance-generating method and regression technique.

Even if the FDSS method generates fewer instances, the results obtained are similar to the LAMA method. Looking at the quality of the solutions of the instances solved by both methods, we can observe that LAMA is usually very close in quality (less than 10% worse than the optimal cost on average in all domains except for *nomystery*). This means that the instances solved by both produce likely similar training examples, and thus we can deduce that fewer instances may lead to similar results in terms of accuracy as long as these instances are representative enough.

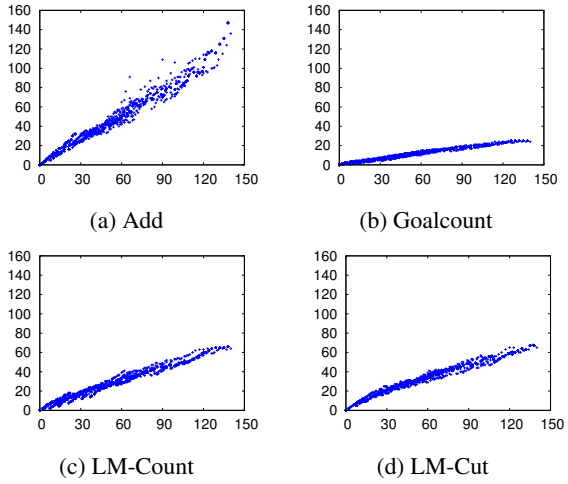
As linear regression is simpler and its accuracy during training is similar to the rest of the regression techniques, all models used from this point on will be the ones computed with it. Of course, good accuracy during training does not guarantee good results in the testing phase, but for the sake of simplicity we assume this is so. Hence, the new heuristic values will be obtained using a linear equation of the form:

$$h_R(n) = w_1 h_1(n) + w_2 h_2(n) + \ldots + w_i h_i(n) + k$$

where h_i are the heuristics selected by attribute selection, w_i the weights for each h_i, and k a constant. We set h_R to zero if $h_R < 0$.

To further justify the adequateness of linear regression, we present a detailed example in the Floortile domain. In this domain 1330 instances were obtained from solving the 20 problems with MHFS. Correlation-based Feature Selection chooses the Additive, Goalcount, Landmark Count and Landmark Cut heuristic functions as the most relevant. We can see the data distribution of these heuristics in Figure 1. All four heuristics have a distinctive linear shape, evidencing why linear regression approximates well the real cost to the goal by weighting the values obtained from the selected heuristics. The obtained Linear Regression model is shown in Equation 1.

$$\begin{aligned} h_R(n) &= 0.3676 * \text{Add}(n) \\ &+ 1.6692 * \text{Goalcount}(n) \\ &+ 0.2429 * \text{LM-count}(n) \\ &+ 0.5490 * \text{LM-cut}(n) \\ &- 4.9717 \end{aligned} \quad (1)$$



(a) Add (b) Goalcount

(c) LM-Count (d) LM-Cut

Figure 1: Data distribution in the Floortile domain (real cost in x-axis and heuristic values in y-axis) of the heuristics chosen by Correlation-based Feature Selection using MHFS.

The heuristic values are not normalized, so the weight is not proportional to the relevance of the heuristic. For instance, Add heuristic yields much higher values than Goalcount, so Goalcount will have higher weights than Add in most cases to compensate for it.

Table 2 summarizes the weights (and thus the selected heuristics by attribute selection) obtained with linear

regression for each domain after learning with the set of instances obtained with MHFS. Here we can see that the heuristic that has been selected more often is FF, followed by Goalcount and LM-Cut. Interestingly enough, the heuristics that LAMA11 uses are FF and LM-Count.

An additional advantage of using Attribute Selection is that it seldom chooses more than one "expensive" heuristic in most cases, because highly correlated heuristics tend to have a similar computational cost. This avoids cases in which computing several expensive heuristics does not improve over using only one of them, which is important to decrease the time spent evaluating states. An alternative could have been using learning algorithms that can take into account the cost of computing the value of an attribute (Núñez 1991), although after Attribute Selection this may be redundant and would force us to use a reduced set of learning techniques.

Results on Testing

To asses the effectiveness of our approach, the learned models were implemented in Fast Downward. In all cases, only linear regression was used. We tested two different configurations for each instance-generating method (FDSS, LAMA11 and MHFS): the linear combination of weighted heuristics as the only heuristic function of the planer (LR); and an alternation multiple queue (Röger and Helmert 2010) that uses the heuristics selected during the learning process (ASH), instead of using the learned model. The motivation behind ASH is that alternation queues are often better than the sum of heuristics (Röger and Helmert 2010). These new planners were compared with all the studied heuristics and the combination of the FF and LM-Count heuristic in an alternation queue, as done in LAMA11. Greedy best-first search with regular evaluation and no preferred operators has been used for all the versions. The scores were computed as in the IPC, using Equation 2.

$$score_p = \begin{cases} \frac{best\_v}{v_p}, & \text{if solution found} \\ 0, & \text{if no solution found} \end{cases} \quad (2)$$

where $best\_v$ is the best value found by any configuration for problem p and v_p is the value for the configuration to be compared. The relevant parameters of the score are cost, number of expanded nodes and time. When computing the score for the time, all instances solved by a planner in less than one second were assumed to be solved in exactly one second.

Table 3 shows the cost, expanded nodes and time scores and the number of solved problems for each heuristic, the FF/LM-Count heuristic combination with an alternation queue and all our approaches. The performance of the FDSS and LAMA11 instance-generating methods is similar, probably because the solutions found by LAMA11 are close to the optimal ones, and in spite of the FDSS instance-generating method generating fewer learning instances. The best instance-generating method is MHFS, in both the LR and ASH combination methods. This is due to the way MHFS obtains the solutions. The role of heuristics is more important when computing the first suboptimal solution, than when finding subsequent (or optimal) solutions by exploring the search space more exhaustively. It is more likely that the best heuristics in the problem were accurate along the first solution path, as they succeeded guiding the search.

LR with the MHFS instance-generating method can solve 185 problems, 24 problems more than LM-Count, the best single heuristic. This approach can solve a problem more than the FF/LM-Count combination. The quality score is similar. ASH(MHFS) can solve 217 problems, 33 problems more than the FF/LM-Count combination (the one used in LAMA11). ASH(MHFS) is also the best configuration in terms of quality. Regarding time, the LM-Count heuristic is the best one, despite solving fewer problems. A similar behaviour can be seen with respect to the number of expanded nodes, where the FF/LM-Count combination is slightly better than ASH(MHFS) with worse coverage.

Approach	Cost score	Expanded nodes score	Time score	# sol.
Add	118.71	50.96	25.62	143
Blind	31.00	0.23	0.68	31
CG	127.18	37.42	44.07	152
CEA	121.51	62.29	24.36	145
FF	108.31	44.48	23.69	132
Goalcount	108.18	27.23	30.30	119
LM-Count	137.56	51.40	**62.53**	161
LM-Cut	98.70	44.74	14.01	114
Max	65.90	23.41	18.94	70
FF,LM-Count	150.79	**104.06**	42.96	184
LR (*FDSS*)	120.05	71.10	17.20	150
ASH (*FDSS*)	156.60	88.17	37.15	182
LR (*LAMA11*)	113.02	65.09	13.81	144
ASH (*LAMA11*)	152.53	76.49	24.46	184
LR (*MHFS*)	150.18	81.70	30.86	185
ASH (*MHFS*)	**192.33**	101.38	52.04	**217**

Table 3: Results regarding cost, number of expanded nodes, execution time and number of solved problems. The instance-generating methods appear in parentheses.

Table 4 and Table 5 show in detail, respectively, the number of solved problems and the cost score per domain by each single heuristic, the FF/LM-Count combination and all our approaches. It is noteworthy that the *a priori* best heuristics are not always chosen, or, if they are chosen, they may not perform as well as expected. For example, using the best heuristics in Barman (FF and LM-Count, which solve 5 and 4 problems respectively) leads to only 5 problems solved; but using Goalcount along with the former heuristics allows the planner to solve all the problems, noticeably more than the sum of problems solved by the three heuristics. Another interesting example is Woodworking, where the FF/LM-Cut combination proves to be much more useful than the FF/LM-Count combination or any single heuristic, doubling the number of problems solved.

Overall, the comparison between the linear combination of heuristics and their use in an alternation list favors the latter. This was at least to some extent expected to be so, because:

Domain	Add	Blind	CG	CEA	FF	Goalcount	LM-Count	LM-Cut	Max	*constant*
Barman					1.08	27.32	0.63			-27.38
Elevators			0.38	0.38	0.47		0.57			-1.33
Floortile	0.37					1.67	0.24	0.55		-4.97
Nomystery			0.17		0.65			0.26		-0.39
Openstacks					1.48	0.17				-1.09
Parcprinter								1.04	0.18	-11,378.34
Parking					0.78		0.51		0.64	-0.63
Pegsol						0.72				-0.09
Scanalyzer					0.49			0.86		0.61
Sokoban					1.22	3.37				0.82
Tidybot					1.58					-0.39
Transport	0.52			0.57		16.05				-66.17
Visitall						0.72		1.33	0.12	-1.82
Woodworking		2.72			0.24	0.66		0.74		-6.36
# of times	2	1	2	2	9	8	4	6	3	

Table 2: Matrix of (non-normalized) weights employed in Linear Regression model for each heuristic on each domain using the instance-generating method MHFS. The last row shows the number of times the heuristic was selected by linear regression.

first, alternation lists exploit effectively the strengths of the more informative heuristics in the instance while paying only a linear amount of time as penalty; second, it introduces diversity, which tends to be beneficial in most planning domains where plateaus may hinder the search process.

Related Work

Our work is based on (Arfaee, Zilles, and Holte 2011) with several important differences. First, they used combinatorial search problems with a single goal state and weak domain-dependent heuristics, whereas we learn from state-of-the-art planning heuristics. Going from specific domains to domain-independent planning is not trivial, so this should be seen as a significant contribution. When they used a planning domain, the Blocksworld domain, they restricted to problems with a single reachable goal state. Second, our generation of the learning instances does not depend on the chosen heuristics. While Arfaee et al. adapt the size of the problems to be solvable by their initial heuristics, we choose an already existing benchmark suite to minimize any bias the training sets may introduce. In fact, we do not see that we are choosing/generating examples, we just take those from another IPC track. It is widely known that the selection of the learning examples often has a high impact on the performance of the final system; in this case, we are exposing ourselves to such a situation and still obtain good results. Finally, our method requires much less time to generate the instances and learn the model.

Several ways of combining heuristics in the same planner have been proposed (Röger and Helmert 2010), among which the use of alternation queues has proved to be the most successful one. We provide in this paper an automatic domain-independent procedure for selecting the right heuristics to be used on those alternation queues.

Other recent works learn in a similar fashion (Xu, Fern, and Yoon 2010). But, they learn weighted action-selection rules to guide a greedy best first search algorithm. They usually need to implement matching mechanisms for each planner they use. The advantage of our work is its simplicity and flexibility, as the learning process is straightforward and it can be used by any planner with minimal modifications.

Planner portfolios that use some learning process to choose the parameters of the final configuration are another alternative to our approach. Two recent examples are Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011) and PbP (Gerevini, Saetti, and Vallati 2009), which choose several configurations of FD and a set of different planners with a given amount of allocated time per planner (plus some macro-actions) respectively. A comparison with such planners however is out of the scope of this work, as we focused on working only with cost-based heuristics to study their behavior in isolation. Additionally, a comparison with PbP would introduce some noise in the results, as the framework would not be the same and hence the differences in the implementation of the planners may influence the performance of the heuristics.

Conclusions and Future Work

In this work, we have proposed the use of a domain-independent learning algorithm to automatically acquire sets of heuristics that are relevant for each domain. The results show that the resulting domain-dependent heuristics greatly enhance the performance of current powerful heuristics for cost-based planning, by configuring state-of-the-art planners for each domain.

In future work, we will focus on exploiting other features of the planning instances to achieve more accurate estimations. Examples of such features may be the size of the

Domain	Add	Blind	CG	CEA	FF	Goalcount	LM-Count	LM-Cut	Max	FF,LM-Count	LR (*FDSS*)	ASH (*FDSS*)	LR (*LAMA11*)	ASH (*LAMA11*)	LR (*MHFS*)	ASH (*MHFS*)
Barman	0	0	0	0	5	0	4	0	0	5	8	20	0	3	8	20
Elevators	1	0	6	5	0	4	0	0	0	1	6	6	7	8	10	10
Floortile	8	0	2	8	7	0	0	7	12	7	4	4	5	4	4	4
Nomystery	6	3	7	7	10	6	14	7	5	13	7	9	8	10	7	7
Openstacks	0	0	0	0	0	7	20	0	0	20	13	13	7	7	13	13
Parcprinter	12	0	15	13	7	0	13	18	3	14	20	19	17	18	19	19
Parking	15	0	20	19	20	0	0	6	0	20	7	5	8	20	20	20
Pegsol	20	17	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Scanalyzer	20	4	20	20	18	20	20	12	5	20	18	20	19	18	20	19
Sokoban	18	4	18	12	18	13	8	18	18	18	18	18	18	18	18	18
Tidybot	17	2	16	18	13	19	19	9	6	14	9	9	11	14	14	16
Transport	12	0	14	9	0	9	18	0	0	3	0	3	8	9	14	15
Visitall	3	0	3	3	4	20	20	6	0	20	10	16	7	15	8	17
Woodworking	11	1	11	11	10	1	5	11	1	9	10	20	9	20	10	19

Table 4: Number of test problems solved by each heuristic on each domain. Cells colored in dark gray are the heuristics chosen to Selected Heuristics and Linear Regression models using MHFS. The instance-generating methods appear in parentheses.

Domain	Add	Blind	CG	CEA	FF	Goalcount	LM-Count	LM-Cut	Max	FF,LM-Count	LR (*FDSS*)	ASH (*FDSS*)	LR (*LAMA11*)	ASH (*LAMA11*)	LR (*MHFS*)	ASH (*MHFS*)
Barman	-	-	-	-	4.5	-	4.0	-	-	4.5	7.3	19.6	-	2.7	7.2	19.5
Elevators	0.7	-	4.6	3.7	-	4.0	-	-	-	0.7	3.7	5.1	5.3	6.5	8.2	9.3
Floortile	6.8	-	1.7	6.7	6.0	-	-	6.3	11.9	5.4	2.7	3.2	4.2	3.0	2.7	3.4
Nomystery	5.5	3.0	6.4	6.4	9.3	5.6	13.8	6.6	4.7	12.6	6.6	8.5	7.4	9.5	6.6	6.6
Openstacks	-	-	-	-	-	7.0	16.9	-	-	13.9	8.3	9.4	3.2	3.3	8.3	9.4
Parcprinter	11.9	-	14.9	12.9	6.9	-	12.9	17.9	3.0	13.8	19.8	18.9	16.9	17.8	18.8	18.9
Parking	12.5	-	15.5	15.5	14.4	-	-	4.5	-	18.9	5.6	4.2	6.5	16.2	16.9	18.5
Pegsol	13.8	17.0	14.7	13.8	14.4	15.0	13.7	17.3	18.3	13.1	12.9	13.6	13.2	13.6	15.1	15.0
Scanalyzer	17.8	4.0	17.0	17.8	15.2	17.4	16.7	11.1	4.6	17.4	16.1	18.6	16.5	16.2	17.7	17.1
Sokoban	15.2	4.0	15.0	9.9	15.3	12.5	6.6	15.2	16.6	13.3	13.5	13.7	13.6	13.7	13.4	13.9
Tidybot	14.2	2.0	13.5	15.4	11.0	17.5	15.0	7.8	5.6	11.3	7.9	7.5	8.7	12.0	12.0	13.9
Transport	8.6	-	12.2	7.6	-	8.0	16.8	-	-	2.4	-	2.8	6.7	7.7	10.6	11.9
Visitall	0.3	-	0.3	0.3	0.7	20.0	16.4	1.3	-	14.8	5.2	12.2	1.2	10.9	2.0	15.8
Woodworking	10.8	1.0	10.9	10.8	9.9	1.0	4.2	10.2	1.0	8.1	9.9	18.7	8.9	18.7	9.9	18.5

Table 5: Cost score of test problems solved by each heuristic on each domain. Cells colored in dark gray are the heuristics chosen by Selected Heuristics and Linear Regression models using MHFS. The instance-generating methods appear in parentheses.

task, the number of deletes of the operators of the problem, the existence of replenishable resources,...

An important characteristic that has been left out in this work is the cost of computing a heuristic. In further work we will try to predict not only the accuracy of a combination of heuristics, but also the "reward"; that is, the ability of the heuristic to guide the search taking into account the time it takes to compute it. As mentioned before, this can be done in a straightforward way using learning algorithms that can include the costs directly in the learning examples, like C4.5 (Núñez 1991), although it would be interesting to define such a measure to assess its viability.

We will also analyze whether a similar learning process can be performed online. This is important to create a domain-independent technique able to capture information particular to a planning task (as opposed to our current method, in which we need learning examples prior to the search process). Similar works have already been done before (Thayer, Dionne, and Ruml 2011), although they did not use general learning techniques as we do in this paper.

Regarding the FD framework and its different configurations, it would also be interesting to analyze the impact of anytime schemes (that may even include unit-cost versions of the heuristics, as LAMA does) and portfolio techniques similar to FD Stone Soup.

Acknowledgements

This work has been partially supported by the INNPACTO program from the Spanish government associated to the MICINN project IPT-370000-2010-008, the MITYC project TSI-090100-2011-33 and by the MICINN projects TIN2008-06701-C03-03 (as a FPI grant) and TIN2011-27652-C03-02.

References

[Arfaee, Zilles, and Holte 2011] Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artif. Intell* 175(16-17):2075–2098.

[Bonet and Geffner 2001] Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

[Gerevini, Saetti, and Vallati 2009] Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: Pbp. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, (ICAPS 2009)*. AAAI.

[Hall 1998] Hall, M. 1998. *Correlation-based Feature Selection for Machine Learning*. Ph.D. Dissertation, Waikato University.

[Helmert and Domshlak 2010] Helmert, M., and Domshlak, C. 2010. Landmarks, critical paths and abstractions: What's the difference anyway? In Brim, L.; Edelkamp, S.; Hansen, E. A.; and Sanders, P., eds., *Graph Search Engineering*, number 09491 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[Helmert and Geffner 2008] Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *ICAPS*, 140–147. AAAI.

[Helmert, Röger, and Karpas 2011] Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. *In ICAPS 2011 Workshop on Planning and Learning* 28–35.

[Helmert 2004] Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *ICAPS*, 161–170. AAAI.

[Helmert 2011] Helmert, M. 2011. The fast downward planning system. *CoRR* abs/1109.6051.

[Hoffmann 2003] Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)* 20:291–341.

[Núñez 1991] Núñez, M. 1991. The use of background knowledge in decision tree induction. *Machine Learning* 6:231–250.

[Quinlan 1992] Quinlan, J. R. 1992. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, 343–348.

[Richter and Westphal 2010] Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39:127–177.

[Richter, Helmert, and Westphal 2008] Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In Fox, D., and Gomes, C. P., eds., *AAAI*, 975–982. AAAI Press.

[Röger and Helmert 2010] Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *ICAPS*, 246–249. AAAI.

[Shevade et al. 2000] Shevade, S. K.; Keerthi, S. S.; Bhattacharyya, C.; and Murthy, K. 2000. Improvements to the SMO algorithm for SVM regression. 1188–1193.

[Thayer, Dionne, and Ruml 2011] Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *ICAPS*. AAAI.

[Witten and Frank 2005] Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann.

[Xu, Fern, and Yoon 2010] Xu, Y.; Fern, A.; and Yoon, S. W. 2010. Iterative learning of weighted rule sets for greedy search. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *ICAPS*, 201–208. AAAI.