# LLAMA: Learning LAMA

**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. Finally, we implemented the most suitable model in the LAMA planner and we called it LLAMA (for Learning LAMA). The main difference respect to LAMA is that we use more appropriate cost-sensitive heuristics during the anytime phase.

## Introduction

In the last IPC[1] (2011), the approach employed 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/`

Based on (Virseda, Borrajo, and Alcázar 2013), this work is focused on improving the efficiency of real-cost heuristics. With this purpose, we learn from existing domain-independent heuristics *per domain*. We follow a procedure similar to the one proposed by Arfaee *et al.* (2011), although in our case we obtain the training instances from small problems (as commonly done in the learning track of the IPC) instead of learning per problem and from initial ad-hoc heuristics.

In order to minimize the error derived from 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 a specific domain, we use machine learning techniques to try to infer useful combinations of heuristics. We extract learning instances from solutions to small 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 can be used later as the new domain-dependent heuristic; it computes at each state the values of several selected heuristics and returns a linear 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 document describes our approach and gives the details of the employed components and techniques.

## 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 easy 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 $H = h_1, h_2, \dots, h_m$ 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 $t(n) = \langle h_1(n), h_2(n), \dots, h_m(n) \rangle$ 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 $\pi = (a_1, a_2, \dots, a_n)$ is the solution to a training problem $p \in P$, and $S_\pi = (s_0, s_1, s_2, \dots, s_n)$ 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: $\langle h_1(s_i), \dots, h_m(s_i), c(s_i) \rangle$.

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.

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 ← $\emptyset$;
    foreach problem $\in P$ do
        solution_path ← apply($M$,problem);
        foreach node $\in$ solution_path do
            instance ← compute_instance(node,$H$);
            instance_set ← instance_set $\cup$ 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.

## Planner Components

This section describes the elements involved in the planner implementation. This includes the chosen heuristics, the methods used to generate the training instances and the machine learning methods.

### 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).

### 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.

## Experimentation

This section includes some experimentation that justifies our choices of parameters. It describes 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, obtained from the optimal track of IPC'11. 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. Results are also consistent across domains too. Accuracy is higher and time is smaller for all learning methods with the training sets obtained with FDSS.

Even if the FDSS method generates fewer instances, the results obtained are similar to the LAMA11 method. Looking at the quality of the solutions of the instances solved by both methods, we can observe that LAMA11 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 methods produce similar training examples *a priori*, which leads us to 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 we will assume it is true. Hence, the new heuristic values of each search node $n$ will be obtained using a linear equation of the form:

| Instance-generating method | Classifier | Correlation | Time(ms) |
|---|---|---|---|
| FDSS | AR | 0.9263 | 94.29 |
| | GP | 0.9462 | 7,657.14 |
| | LR | 0.9451 | 74.29 |
| | M5P | 0.9505 | 362.86 |
| | M5Rules | 0.9500 | 543.57 |
| | REPT | 0.9451 | 65.71 |
| | SMOreg | 0.9450 | 970.00 |
| LAMA11 | AR | 0.9195 | 127.14 |
| | GP | 0.9337 | 12,121.40 |
| | LR | 0.9291 | 98.57 |
| | M5P | 0.9344 | 669.29 |
| | M5Rules | 0.9329 | 1,140.00 |
| | REPT | 0.9387 | 93.57 |
| | SMOreg | 0.9207 | 1683.57 |
| MHFS | AR | 0.9271 | 130.00 |
| | GP | 0.9445 | 20,095.00 |
| | LR | 0.9399 | 101.43 |
| | M5P | 0.9495 | 736.43 |
| | M5Rules | 0.9492 | 1,201.43 |
| | REPT | 0.9479 | 94.29 |
| | SMOreg | 0.9384 | 2,495.00 |

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

$$h_R(n) = w_1h_1(n) + w_2h_2(n) + \ldots + w_mh_m(n) + k$$

where $w_i$ is the weight associated to $h_i$ ($w_i = 0$ if $h_i$ was not selected by Attribute Selection, and we would not need to compute their values when searching) and $k$ is a constant. We set $h_R$ to zero if $h_R < 0$. Regarding $k$, some clarifications must be made. The heuristic function $h_R$ will be used in all nodes. Since the function of an inadmissible heuristic is to help discriminate between nodes, adding or subtracting a constant has no effect at all. In this case, though, $h_R$ may have negative values due to negative weights and/or a negative $k$. This means that taking $k$ out of $h_R$ will affect the cases in which $h_R < 0$ (either before or after the removal of $k$), which may affect the ranking of the nodes during search. Therefore, $k$ is necessary to ensure a consistent behavior.

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

An additional advantage of using Attribute Selection is that in most cases it does not select more than one "expensive" heuristic, 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 assess 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 have been used for all the versions for the same reasons as in the previous section. The scores were computed as in the IPC. The metric used in the IPC is quality-oriented; nevertheless, in this work we will report the parameters: quality, expanded nodes and time. When computing the score for time, all the instances solved by a planner in less than one second are assumed to be solved in exactly one second. This is so because below one second the factors that affect the total time may be others than the efficiency of the search process. Also, due to the way the score is computed, too low values may skew the results favorably in favor of configurations that can solve some instances in very little time. All scores are computed using Equation 1:

$$score_{p,r} = \begin{cases} \frac{best\_v}{v_{p,r}}, & \text{if solution found} \\ 0, & \text{if no solution found} \end{cases} \tag{1}$$

where $r$ is a configuration (heuristic), $p$ is a problem, $best\_v$ is the best value found by any configuration for $p$ and $v_{p,r}$ is the value of $r$ for $p$.

Table 2 shows the quality, 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 despite the fact that the FDSS instance-generating method generates fewer learning instances. The best instance-generating method with respect to the evaluated metrics 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.

| Heuristic | Quality score | Expanded nodes score | Time score | Coverage |
|---|---|---|---|---|
| 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 |
| Combination | Quality score | Expanded nodes score | Time score | Coverage |
| 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 2: Results regarding quality, number of expanded nodes, execution time and number of solved problems for individual heuristics and combination of heuristics. The instance-generating methods appear in parentheses.

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 results with respect the quality score are 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.

Overall, the comparison between the linear combination of heuristics and their use in an alternation list favors the latter. This was at least to be to some extent expected, because: 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; and second, they introduce diversity, which tends to be beneficial in most planning domains where plateaus may hinder the search process.

## Implementation Details

After the experimentation we are now in position to design a competitive planner. From our observations, the use of cost-sensitive heuristics leads to an improvement in quality, but it is still lackluster in terms of coverage. For this reason, we adopt a strategy similar to LAMA's. LAMA uses an anytime scheme composed by several phases, in which first unit-cost heuristics are used (in particular the FF and LandmarkCount heuristics) along with greedy best first search, deferred evaluation and preferred operators, and after founding the first solution a series of searches are sequentially performed, in which greedy enhancements are disabled and Weighted A$^*$ is used with a decreasing weight.

The main drawback of LAMA's strategy is that using the FF and LandmarkCount heuristics after the first solution is found is unjustified. Because of this, LLAMA learns which combination of heuristics is the most appropriate per domain and uses that combination instead of the FF and LandmarkCount heuristics once the first solution is found. This achieves the same coverage as LAMA but aims to improve the quality score thanks to the learning phase.

LLAMA's anytime phase is identical to LAMA's except for the employed heuristics. The heuristics are used in an alternating queue instead of using the linear combination, as Table 2 shows that the combination in an alternation queue performs much better.

## 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.

[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.

[Virseda, Borrajo, and Alcázar 2013] Virseda, J.; Borrajo, D.; and Alcázar, V. 2013. Learning heuristic functions for cost-based planning. In *Preprints of the ICAPS'13 PAL Workshop on Planning and Learning*.

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