An Evaluation of Machine Learning in Algorithm Selection for Search Problems

Lars Kotthoff [a,*], Ian P. Gent [a] and Ian Miguel [a]

[a] *School of Computer Science*
Jack Cole Building, North Haugh
St Andrews
KY16 9SX
United Kingdom
E-mail: {*larsko,ipg,ianm*}*@cs.st-andrews.ac.uk*

Machine learning is an established method of selecting algorithms to solve hard search problems. Despite this, to date no systematic comparison and evaluation of the different techniques has been performed and the performance of existing systems has not been critically compared with other approaches. We compare the performance of a large number of different machine learning techniques from different machine learning methodologies on five data sets of hard algorithm selection problems from the literature. In addition to well-established approaches, for the first time we also apply statistical relational learning to this problem. We demonstrate that there is significant scope for improvement both compared with existing systems and in general. To guide practitioners, we close by giving clear recommendations as to which machine learning techniques are likely to achieve good performance in the context of algorithm selection problems. In particular, we show that linear regression and alternating decision trees have a very high probability of achieving better performance than always selecting the single best algorithm.

Keywords: Algorithm Selection, Machine Learning, Combinatorial Search

1. Introduction

The technique of portfolio creation and algorithm selection has recently received a lot of attention in areas of artificial intelligence that deal with solving computationally hard problems [26, 39]. The current state of the art is such that often there are many algorithms and systems for solving the same kind of problem; each with different performance characteristics on a particular problem.

Recent research has focussed on creating *algorithm portfolios*, which contain a selection of state of the art algorithms. To solve a particular problem with a portfolio, the suitability of each algorithm in the portfolio for the problem at hand is assessed in a preprocessing step. This step often involves some kind of machine learning, as the actual performance of each algorithm on the given, unseen problem is unknown.

The algorithm selection problem was first described many decades ago by Rice [30] and numerous systems that employ machine learning techniques have been developed since [26, 28, 37, 39]. While there has been some small-scale work to compare the performance of different machine learning algorithms (e.g. [28]), there has been no comparison of the machine learning methodologies available for algorithm selection and large-scale evaluation of their performance to date.

The systems that perform algorithm selection usually justify their choice of a machine learning methodology (or a combination of several) with their performance compared with one of the algorithms selected from and do not critically assess the real performance – could we do as well or even better by using just a single algorithm instead of having to deal with portfolios and complex machine learning?

This paper presents a comprehensive comparison of machine learning paradigms and techniques for tackling algorithm selection. It evaluates the performance of a large number of different techniques on data sets used in the literature. We furthermore compare our results with existing systems and to a simple "winner-takes-all" approach where the best overall algorithm is always selected. We demonstrate that this approach performs quite well in practice, a result that we were surprised by. Based on the results of these extensive experiments and additional statistical simulations, we give recommendations as to which machine learning techniques should be considered when performing algorithm selection.

[*]Corresponding Author.

The aim of the investigation presented here is not to establish a set of machine learning algorithms that are best in general for algorithm selection and should be used in all cases, but rather to provide guidance to researchers with little experience in algorithm selection. In any particular scenario, an investigation similar to the one presented here can be performed to establish the best machine learning method for the specific case if the resources for doing so are available.

2. Background

We are addressing an instance of the algorithm selection problem [30] – given variable performance among a set of algorithms, choose the best candidate for a particular problem instance. Machine learning is an established method of addressing this problem [5, 24]. Given the performance of each algorithm on a set of training problems, we try to predict the performance on unseen problems.

An algorithm portfolio [11, 23] consists of a set of algorithms. A subset is selected and applied sequentially or in parallel to a problem instance, according to some schedule. The schedule may involve switching between algorithms while the problem is being solved (e.g. [21, 36]). We consider the problem of choosing the best algorithm from the portfolio (i.e. a subset of size 1) and using it to solve the particular problem instance to completion. In this context, the widest range of machine learning techniques are applicable. Some of the techniques are also applicable in other contexts – performance predictions can easily be used to devise a schedule with time allocations for each algorithm in the portfolio, which can then be applied sequentially or in parallel. Therefore some of our results are also relevant to other approaches.

There have been many systems that use algorithm portfolios in some form developed over the years and an exhaustive list is beyond the scope of this paper. Smith-Miles [34] presents a survey of many different approaches. One of the earliest systems was Prodigy [3], a planning system that uses various machine learning methodologies to select from search strategies. PYTHIA [37] is more general and selects from among scientific algorithms. MULTI-TAC [25] tailors constraint solvers to the problems they are to tackle. Borrett et al. [2] employed a sequential portfolio of constraint solvers. More recently, Guerri and Milano [12] use a decision-tree based technique to select among a portfolio of constraint- and integer-programming based solution methods for the bid evaluation problem. In the area of hard combinatorial search problems, a highly successful approach in satisfiability (SAT) is SATzilla [39]. In constraint programming, CP-Hydra uses a similar approach [26]. The AQME system [28] performs algorithm selection for finding satisfying assignments for quantified Boolean formulae.

Silverthorn and Miikkulainen [33] describe a different approach to portfolio performance prediction. They cluster instances into latent classes (classes that are unknown before and only emerge as the clustering takes place) and choose the best algorithm for each class. The ISAC system [18] is similar in that it clusters problem instances and assigns algorithms to each cluster, but it does not use a latent approach. The classes are only implicit in the clusters. Stern et al. [35] use a Bayesian model to manage portfolios and allow for changes to the algorithms from the portfolio and problem instance characteristics. Hydra [40] configures algorithms before adding them to the portfolio. The aim is to compose portfolios where the algorithms complement each other.

There are many different approaches to using machine learning for algorithm selection. Often, the method of choice is not compared with other approaches. Justification of the authors' decision usually takes the form of demonstrated performance improvements over a single algorithm of the ones being selected from. Other approaches use ensembles of machine learning algorithms to provide good performance [20].

There are a few publications that do explicitly compare different machine learning algorithms. Xu et al. [39] mention that, in addition to the chosen ridge regression for predicting the runtime, they explored using lasso regression, support vector machines and Gaussian processes. Cook and Varnell [4] compare different decision tree learners, a Bayesian classifier, a nearest neighbour approach and a neural network. Leyton-Brown et al. [22] compare several versions of linear and non-linear regression. Guo and Hsu [13] explore using decision trees, naïve Bayes rules, Bayesian networks and meta-learning techniques. Gebruers et al. [6] compare nearest neighbour classifiers, decision trees and statistical models. Hough and Williams [16] use decision tree ensembles and support vector machines. Bhowmick et al. [1] investigate alternating decision trees and various forms of boosting, while Pulina and Tacchella [27] use decision trees, decision rules, logistic regression and nearest neighbour approaches and Roberts and Howe [32] evaluate 32 different Machine Learning algorithms for predicting the runtime. Silverthorn and Miikkulainen [33] compare the performance of different latent class models. Gent et al., Kotthoff et al. [8, 20] compare 18 classification algorithms.

Of these, only [6, 8, 13, 16, 20, 27, 33] quantify the performance of the different methods they used. The other comparisons give only qualitative evidence. None of the publications that give quantitative evidence are as comprehensive as this study, neither in terms of data sets nor Machine Learning algorithms.

3. Algorithm selection methodologies

In an ideal world, we would know enough about the algorithms in the portfolio to formulate rules to select a particular one based on certain characteristics of a problem to solve. In practice, this is not possible except in trivial cases. For complex algorithms and systems, like the ones mentioned above, we do not understand the factors that affect the performance of a specific algorithm on a specific problem well enough to make the decisions the algorithm selection problem requires with confidence.

As outlined above, a common approach to overcoming these difficulties is to use machine learning. Several machine learning methodologies are applicable here. We present the most prevalent ones below. We use the term "methodology" to mean a kind of machine learning algorithm that can be used to achieve a certain kind of prediction output. In addition to these, we use a simple majority predictor that always predicts the algorithm from the portfolio with the largest number of wins, i.e. the one that is fastest on the largest subset of all training instances, ("winner-takes-all" approach) for comparison purposes. This provides an evaluation of the real performance improvement over manually picking the best algorithm from the portfolio. For this purpose, we use the WEKA [14] `ZeroR` classifier implementation.

3.1. Case-based reasoning

Case-based reasoning informs decisions for unseen problems with knowledge about past problems. An introduction to the field can be found in [31]. The idea behind case-based reasoning is that instead of trying to construct a theory of what characteristics affect the performance, examples of past performance are used to infer performance on new problems.

The main part of a case-based reasoning system is the case base. We use the WEKA `IBk` nearest-neighbour classifier with 1, 3, 5 and 10 nearest neighbours considered as our case-based reasoning algorithms. The case base consists of the problem instances we have encountered in the past and the best algorithm from the portfolio for each of them – the set of training instances and labels. Each case is a point in n-dimensional space, where n is the number of attributes each problem has. The nearest neighbours are determined by calculating the Euclidean distance. While this is a very weak form of case-based reasoning, it is consistent with the observation above that we simply do not have more information about the problems and algorithms from the portfolio that we could encode in the reasoner.

The attraction of case-based reasoning, apart from its conceptual simplicity, is the fact that the underlying performance model can be arbitrarily complex. As long as the training data is representative (i.e. the case base contains problems similar to the ones we want to solve with it), the approach will achieve good performance.

We use the AQME system [28] as a reference system that uses case-based reasoning to compare with. AQME uses a nearest-neighbour classifier to select the best algorithm.

3.2. Classification

Intuitively, algorithm selection is a simple classification problem – label each problem instance with the algorithm from the portfolio that should be used to solve it. We can solve this classification problem by learning a classifier that discriminates between the algorithms in the portfolio based on the characteristics of the problem. A set of labelled training examples is given to the learner and the learned classifier is then evaluated on a set of test instances.

We use the WEKA

- `AdaBoostM1`,
- `BayesNet`,
- `BFTree`,
- `ConjunctiveRule`,
- `DecisionTable`,
- `FT`,
- `HyperPipes`,
- `J48`,

- `J48graft`,
- `JRip`,
- `LADTree`,
- `LibSVM` (with radial basis and sigmoid function kernels),
- `MultilayerPerceptron`,
- `OneR`,
- `PART`,
- `RandomForest`,
- `RandomTree` and
- `REPTree`

classifiers. Our selection is large and inclusive and contains classifiers that learn all major types of classification models. In addition to the WEKA classifiers, we used a custom classifier that assumes that the distribution of the class labels for the test set is the same as for the training set and samples from this distribution without taking features into account.

We consider the classifier presented by Gent et al. [7] as a reference system from the literature to compare with. They use a decision tree induced by the `J48` algorithm.

3.3. Regression

Instead of considering all algorithms from the portfolio together and selecting the one with the best performance, we can also try to predict the performance of each algorithm on a given problem independently and then select the best one based on the predicted performance measures. The downside is that instead of running the machine learning once per problem, we need to run it for each algorithm in the portfolio for a single problem.

The advantage of this approach is that instead of trying to learn a model of a particular portfolio, the learned models only apply to individual algorithms. This means that changing the portfolio (i.e. adding or removing algorithms) can be done without having to retrain the models for the other algorithms. Furthermore, the performance model for a single algorithm might be not as complex and easier to learn than the performance model of a portfolio.

Regression is usually performed on the runtime of an algorithm on a problem. Xu et al. [39] predict the logarithm of the runtime because they "have found this log transformation of runtime to be very important due to the large variation in runtimes for hard combinatorial problems."

We use the WEKA

- `GaussianProcesses`,
- `LibSVM` (ε and ν),
- `LinearRegression`,
- `REPTree` and
- `SMOreg`

learners to predict both the runtime and the logarithm of the runtime. Again we have tried to be inclusive and add as many different regression learners as possible regardless of our expectations as to their suitability or performance.

We use a modified version of SATzilla [39] (denoted SATzilla$'$) to compare with. SATzilla uses a form of ridge regression to predict a performance measure related to the runtime of an algorithm on a problem.

3.4. Statistical relational learning

Statistical relational learning is a relatively new discipline of machine learning that attempts to predict complex structures instead of simple labels (classification) or values (regression) while also addressing uncertainty. An introduction can be found in [10]. For algorithm selection, we try to predict the performance ranking of the algorithms from the portfolio on a particular problem.

We consider this approach promising because it attempts to learn the model that that is most intuitive for humans. In the context of algorithm portfolios, we do not care about the performance of individual algorithms, but the relative performance of the algorithms in the portfolio. While this is not relevant for selecting the single best algorithm, many approaches use predicted performance measures to compute schedules according to which to run the algorithms in the portfolio (e.g. [9, 26, 28]). We also expect a good model of this sort to be much more robust with respect to the inherent uncertainty of empirical performance measurements.

We use the support vector machine SVM^{rank} instantiation[1] of SVM^{struct} [17]. It was designed to predict ranking scores. Instances are labelled and grouped according to certain criteria. The labels are then ranked within each group. We can use the system unmodified for our purposes and predict the ranking score for each algorithm on each problem. We left the parameters at their default values and used a value of 0.1 for the convergence parameter ε except in cases where the model learner did not converge within an hour. In these cases, we set $\varepsilon = 0.5$.

[1] `http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html`

To the best of our knowledge, statistical relational learning has never before been applied to algorithm selection.

4. Evaluation data sets

We evaluate and compare the performance of the approaches mentioned above on five data sets of hard algorithm selection problems taken from the literature. We take three sets from the training data for SATzilla 2009. This data consists of SAT instances from three categories – hand-crafted, industrial and random. They contain 1181, 1183 and 2308 instances, respectively. The SATzilla authors use 91 attributes for each instance and select a SAT solver from a portfolio of 19 solvers[2]. We compare the performance of each of our methodologies to a modified version of SATzilla that only outputs the predictions for each problem without running a presolver or doing any of the other optimisations and denote this system SATzilla$'$. While the effectiveness of such optimisations has been shown in some cases, most systems do not use them (e.g. [26, 28]). We adjusted the timeout values reported in the training data available on the website to 3600 seconds after consultation with the SATzilla team as some of the reported timeout values are incorrect.

The fourth data set comes from the QBF Solver Evaluation 2010[3] and consists of 1368 QBF instances from the main, small hard, 2QBF and random tracks. 46 attributes are calculated for each instance and we select from a portfolio of 5 QBF solvers. Each solver was run on each instance for at most 3600 CPU seconds. If the solver ran out of memory or was unable to solve an instance, we assumed the timeout value for the runtime. The experiments were run on a machine with a dual 4 core Intel E5430 2.66 GHz processor and 16 GB RAM. We compare the performance to that of the AQME system.

Our last data set is taken from [7] and selects from a portfolio of two solvers for a total of 2028 constraint problem instances from 46 problem classes with 17 attributes each. We compare our performance to the classifier described in the paper.

For each data set, some of the attributes are cheap to compute while others are extremely expensive. In practice, steps are usually taken to avoid the expensive attributes; see for example [7], who explicitly eliminate them. More details can be found in the referenced publications.

We chose the data sets because they represent algorithm selection problems from three areas where the technique of algorithm portfolios has attracted a lot of attention recently. For all sets, reference systems exist that we can compare with. Furthermore, the number of algorithms in the respective portfolios for the data sets is different.

It should be noted that the systems we are comparing against are given an unfair advantage. They have been trained on at least parts of the data that we are using for the evaluation. Their performance was assessed on the full data set as a black box system. The machine learning algorithms we use however are given disjoint sets of training and test instances.

5. Methodology

The focus of our evaluation is the performance of the machine learning algorithms. Additional factors that would impact the performance of an algorithm selection system in practice are not taken into account. These factors include the time to calculate problem features and additional considerations for selecting algorithms, such as memory requirements.

We furthermore do not assess the impact of techniques such as using a presolver to allow the machine learning to focus on problems that take a long time to solve. While this technique has been used successfully by Xu et al. [39], most approaches in the literature do not use such techniques (e.g. [21, 26, 28]). Therefore, our results are applicable to a wide range of research.

We measured the performance of the machine learning algorithms in terms of misclassification penalty. The misclassification penalty is the additional CPU time we need to solve a problem instance if not choosing the best algorithm from the portfolio, i.e. the difference between the CPU time the selected algorithm required and the CPU time the fastest algorithm would have required. This is based on the intuition that we do not particularly care about classifying as many instances correctly as possible; we rather care that the instances that are important to us are classified correctly. The wider the performance between the best and worst algorithm for an instance, the more important it is to us. If the selected algorithm was not able to solve the problem, we assumed the timeout value minus the fastest CPU time to be the misclassification penalty. This only

[2]`http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

[3]`http://www.qbflib.org/index_eval.php`

gives a weak lower bound, but we cannot determine the correct value without running the algorithm to completion.

For the classification learners, we attached the maximum misclassification penalty as a weight to the respective problem instance during the training phase. The intuition is that instances with a large performance difference between the algorithms in the portfolio are more important to classify correctly than the ones with almost no difference. We use the weight as a means of biasing the machine learning algorithms towards these instances. The maximum misclassification penalty is the maximum possible gain – if the default choice is the worst performer, we can improve the solve time by this much by selecting the best performer. This weight ensures that the optimisation objective of the classification learners is the same as the objective we are using in the evaluation – minimising the additional time required because of misclassifications.

The handling of missing attribute values was left up to the specific machine learning system. We estimated the performance of the learned models using ten-fold stratified cross-validation [19]. The performance on the whole data set was estimated by summing the misclassification penalties of the individual folds.

For each data set, we used two sets of features – the full set and the subset of the most predictive features. We used WEKA's `CfsSubsetEval` attribute selector with the `BestFirst` search method with default parameters to determine the most predictive features for the different machine learning methodologies. We treated SVM^{rank} as a black box algorithm and therefore did not determine the most predictive features for it.

We performed a full factorial set of experiments where we ran each machine learning algorithm of each methodology on each data set. We also evaluated the performance with thinned out training data. We randomly deleted 25, 50 and 75$\%$ of the problem-algorithm pairs in the training set. We thus simulated partial training data where not all algorithms in the algorithm portfolio had been run on all problem instances. The missing data results in less comprehensive models being created.

To evaluate the performance of the algorithm selection systems we compare with, we ran them on the full, unpartitioned data set. The misclassification penalty was calculated in the same way as for the machine learning algorithms.

5.1. Machine learning algorithm parameters

We tuned the parameters of all machine learning algorithms to achieve the best performance on the given data sets. Because of the very large space of possible parameter configurations, we focussed on the subset of the parameters that is likely to affect the generalisation error. Tuning the values of all parameters would be prohibitively expensive. The total number of evaluated configurations was 19,032.

Our aim was to identify the parameter configuration with the best performance on *all* data sets. Configurations specific to a particular data set would prevent us from drawing conclusions as to the performance of the particular machine learning algorithm in general. It is very likely that the performance on a particular data set can be improved significantly by carefully tuning a machine learning algorithm to it (cf. [7]), but this requires significant effort to be invested in tuning for each data set.

Our intention for the results presented in this paper is twofold. On one hand, the algorithms that we demonstrate to have good performance can be used with their respective configurations as-is by researchers wishing to build an algorithm selection system for search problems. On the other hand, these algorithm configurations can serve as a starting point for tuning them to achieve the best performance on a particular data set. The advantage of the former approach is that a machine learning algorithm can be chosen for a particular task with quantitative evidence for its performance already available.

In many approaches in the literature, machine learning algorithms are not tuned at all if the performance of the algorithm selection system is already sufficient with default parameters. Many researchers who use machine learning for algorithm selection are not machine learning experts.

We used the same methodology for tuning as for the other experiments. For each parameter configuration, the performance in terms of misclassification penalty with the full set of parameters on each data set was evaluated using ten-fold stratified cross-validation. We determined the best configurations by calculating the intersection of the set of best configurations on each individual data set. For four algorithms, this intersection was empty and we used the configurations closest to the best one to determine the best overall configuration. This was the case for the classification algorithms `BFTree`, `DecisionTable`, `JRip` and `PART`. For all other algorithms, there was at least one configuration that achieved the best performance on all data sets.

We found that for most of the machine learning algorithms that we used, the default parameter values already gave the best performance across all data sets. Furthermore, most of the parameters had very little or no effect; only a few made a noticeable difference. For SVM^{rank}, we found that only a very small number of parameter configurations were valid across all data sets – in the majority of cases, the configuration would produce an error. We decided to change the parameter values from the default for the six case-based reasoning and classification algorithms below.

AdaBoostM1 We used the `-Q` flag that enables resampling.
DecisionTable We used the `-E acc` flag that uses the accuracy of a table to evaluate its classification performance.
`IBk` with 1, 3, 5 and 10 neighbours We used the `-I` flag that weights the distance by its inverse.
J48 We used the flags `-R -N 3` for reduced error pruning.
JRip We used the `-U` flag to prevent pruning.
PART We used the `-P` flag to prevent pruning.

We were surprised that the use of pruning decreased the performance on unseen data. Pruning is a way of preventing a learned classifier from becoming too specific to the training data set and generalising poorly to other data. One possible explanation for this behaviour is that the concept that the classifier learns is sufficiently prominent in even relatively small subsets of the original data and pruning over-generalises the learned model which leads to a reduction in performance.

6. Experimental results

We first present and analyse the results for each machine learning methodology and then take a closer look at the individual machine learning algorithms and their performance.[4]

The misclassification penalty in terms of the majority predictor for all methodologies and data sets is shown in Figure 1. The results range from a misclassification penalty of less than 10% of the majority predictor to almost 650%. In absolute terms, the difference to always picking the best overall algorithm can be from an improvement of more than 28 minutes per problem to a decrease in performance of more than 41 minutes per problem.

At first glance, no methodology seems to be inherently superior. The "No Free Lunch" theorems, in particular the one for supervised learning [38], suggest this result. We were surprised by the good performance of the majority predictor, which in particular delivers excellent performance on the industrial SAT data set. The SVM^{rank} relational approach is similar to the majority predictor when it delivers good performance.

Many publications do not compare their results with the majority predictor, thus creating a misleading impression of the true performance. As our results demonstrate, always choosing the best algorithm from a portfolio without any analysis or machine learning can *significantly outperform* more sophisticated approaches.

Some of the machine learning algorithms perform worse than the majority predictor in some cases. There are a number of possible reasons for this. First, there is always the risk of overfitting a trained model to the training set such that it will have bad performance on the test set. While cross-validation somewhat mitigates the problem, it will still occur in some cases. Second, the set of features we are using may not be informative enough. The feature sets are however what is used in state of the art algorithm selection systems and able to inform predictions that provide performance improvements in some cases.

Figure 2 shows the misclassification penalty in terms of a classifier that learns a simple rule (`OneR` in WEKA) – the data is the same as in Figure 1, but the reference is different. This evaluation was inspired by Holte [15], who reports good classification results even with simple rules. On the QBF and SAT-IND data sets, there is almost no difference. On the CSP data set, a simple rule is not able to capture the underlying performance characteristic adequately – it performs worse than the majority predictor, as demonstrated by the improved relative performance. On the remaining two SAT data sets, learning a simple classification rule improves over the performance of the majority predictor.

The reason for including this additional comparison was to show that there is no simple solution to the problem. In particular, there is no single attribute that adequately captures the performance characteristics and could be used in a simple rule to reliably predict the

[4]Some of the results in a previous version of this paper have been corrected here. For an explanation of the issue, see `http://www.cs.st-andrews.ac.uk/~larsko/asc-correction.pdf`.

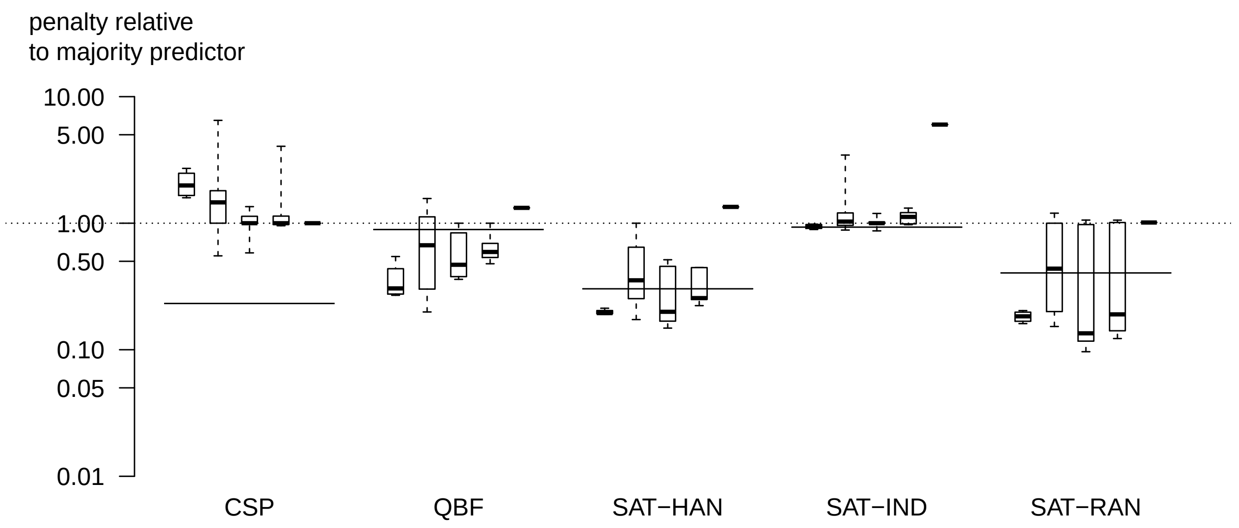

Fig. 1. Experimental results with full feature sets and training data across all methodologies and data sets. The plots show the 0th (bottom line), 25th (lower edge of box), 50th (thick line inside box), 75th (upper edge of box) and 100th (top line) percentile of the performance of the machine learning algorithms for a particular methodology (4 for case-based reasoning, 19 for classification, 6 for regression and 1 for statistical relational learning). The boxes for each data set are, from left to right, case-based reasoning, classification, regression, regression on the log and statistical relational learning. The performance is shown as a factor of the simple majority predictor which is shown as a dotted line. Numbers less than 1 indicate that the performance is better than that of the majority predictor. The solid lines for each data set show the performance of the systems we compare with ([7] for the CSP data set, [28] for the QBF data set and SATzilla′ for the SAT data sets).

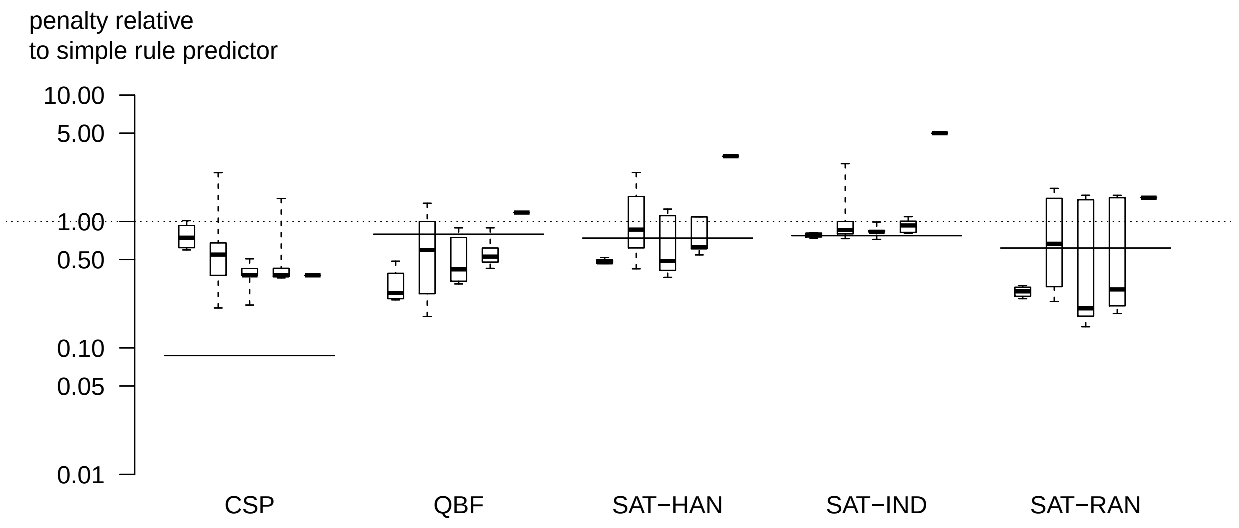

Fig. 2. Experimental results with full feature sets and training data across all methodologies and data sets. The boxes for each data set are, from left to right, case-based reasoning, classification, regression, regression on the log and statistical relational learning. The performance is shown as a factor of a classifier that learns a simple rule (`OneR` in WEKA) which is shown as a dotted line. Numbers less than 1 indicate that the performance is better than that of the simple rule predictor.

best solver to use. On the contrary, the results suggest that considering only a single attribute in a rule is an oversimplification that leads to a deterioration of overall performance. The decrease in performance compared to the majority predictor on some of the data sets bears witness to this.

To determine whether regression on the runtime or on the log of the runtime is better, we estimated the performance with different data by choosing 1000 bootstrap samples from the set of data sets and comparing the performance of each machine learning algorithm for both types of regression. Regression on the runtime has a higher chance of better performance – with a probability of $\approx$67% it will be better than regression on the log of the runtime on the full data set. With thinned out training data the picture is different however and regression on the log of the runtime delivers better performance. We therefore show results for both types of regression in the remainder of this paper.

6.1. *Most predictive features and thinned out training data*

Figure 3 shows the results for the set of the most predictive features. The results are very similar to the ones with the full set of features. A bootstrapping estimate as described above indicated that the probability of the full feature set delivering results better than the set of the most important features is $\approx$69%. Therefore, we only consider the full set of features in the remainder of this paper – it is better than the selected feature set with a high probability and does not require the additional feature selection step. In practice, most of the machine learning algorithms ignore features that do not provide relevant information anyway – either explicitly like `J48` by not including them in the generated decision tree, or implicitly like the regression techniques that set their factors to zero.

The effects of thinning out the training data were different across the data sets and are shown in Figure 4. On the industrial and random SAT data sets, the performance varied seemingly at random; sometimes increasing with thinned out training data for one machine learning methodology while decreasing for another one on the same data set. On the handcrafted SAT and QBF data sets, the performance decreased across all methodologies as the training data was thinned out while it increased on the CSP data set. Statistical relational learning was almost unaffected in most cases.

There is no clear conclusion to be drawn from these results as the effect differs across data sets and methodologies. They however suggest that deleting a proportion of the training data may improve the performance of the machine learning algorithms. At the very least, not running all algorithms on all problems because of resource constraints seems to be unlikely to have a large negative impact on performance as long as most algorithms are run on most problems.

The size of the algorithm portfolio did not have a significant effect on the performance of the different machine learning methodologies. For all data sets, each solver in the respective portfolio was the best one in at least some cases – it was not the case that although the portfolio sizes are different, the number of solvers the should be chosen in practice was the same or very similar. Figure 1 does not show a general increase or decrease in performance as the size of the portfolio increases. In particular, the variation in performance on the three SAT data sets with the same portfolio size is at least as big as the variation compared to the other two data sets with different portfolio sizes.

Our intuition was that as the size of the portfolio increases, classification would perform less well because the learned model would be more complex. At the same time, we expected the performance of regression to increase because the complexity of the learned models does not necessarily increase. In practice however the opposite appears to be the case – on the CSP data set, where we select from only 2 solvers, classification and case-based reasoning perform worse compared with the other methodologies than on the other data sets. It turned out however that the number of algorithms selected from the portfolio by the machine learning algorithms at all was small in all cases. As we compared only three different portfolio sizes, there is not enough data from which to draw definitive conclusions.

6.2. *Best machine learning methodology*

As it is not obvious from the results which methodology is the best, we again used bootstrapping to estimate the probability of being the best performer for each one. We sampled, with replacement, from the set of data sets and for each methodology from the set of machine learning algorithms used and calculated the ranking of the median and maximum performances across the different methodologies. Repeated 1000 times, this gives us the likelihood of an average and the best algorithm of each methodology being

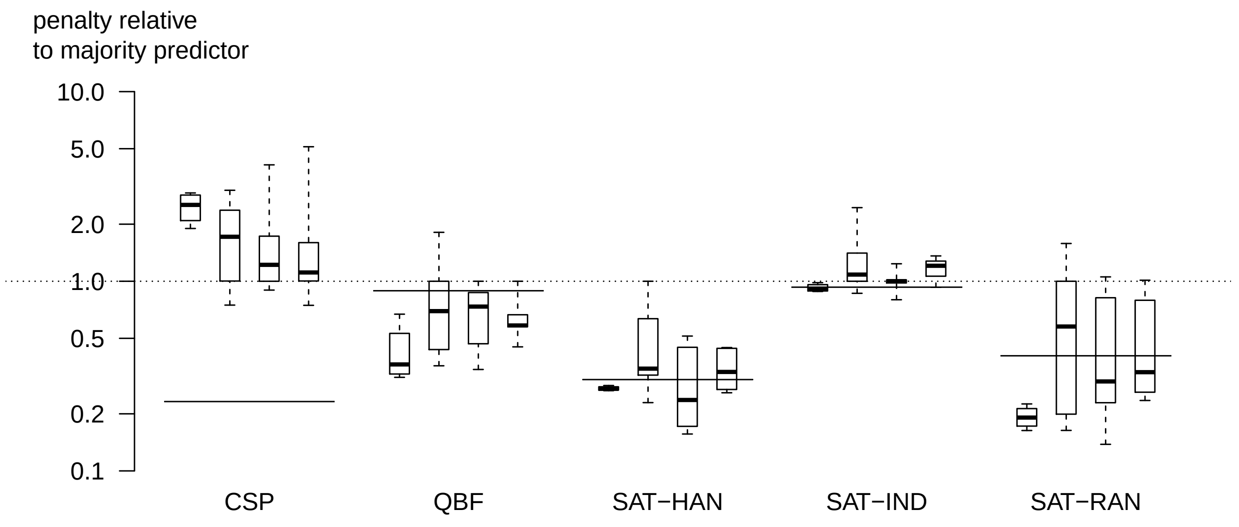

Fig. 3. Experimental results with reduced feature sets across all methodologies and data sets. The boxes for each data set are, from left to right, case-based reasoning, classification, regression and regression on the log. We did not determine the set of the most predictive features for statistical relational learning. The performance is shown as a factor of the simple majority predictor. For each data set, the most predictive features were selected and used for the machine learning.

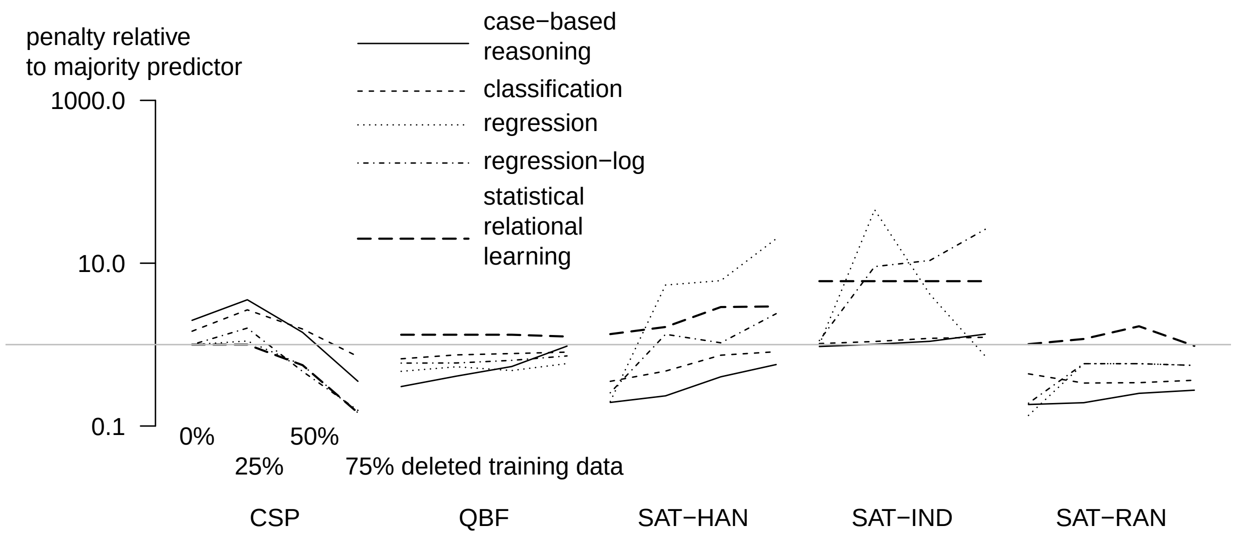

Fig. 4. Experimental results with full feature sets and thinned out training data across all methodologies and data sets. The lines show the median penalty (thick line inside the box in the previous plots) for 0%, 25%, 50% and 75% of the training data deleted. The performance is shown as a factor of the simple majority predictor which is shown as a grey line. Numbers less than 1 indicate that the performance is better than that of the majority predictor.

ranked 1st, 2nd and 3rd. We chose the median performance for comparison because there was no machine learning algorithm with a clearly better performance than all of the others and algorithms with a good performance on one data set would perform much worse on different data. We also include the performance of the best algorithm because the number of algorithms in each methodology is different. While we believe that we included algorithms that represent a representative variety of approaches within each methodology, the median performance of all algorithms of each methodology may in some cases obscure the performance of the best algorithm. This is especially possible for the methodologies that include a large number of different algorithms. We used the same bootstrapping method to estimate the likelihood that an average and the best machine learning algorithm of a certain methodology would perform better than the simple majority predictor. The probabilities are summarised in Table 1.

Based on the bootstrapping estimates, an average case-based reasoning algorithm is most likely to give the best performance and most likely to deliver good performance in terms of being better than the majority predictor. The picture is different when considering the best algorithm within each methodology rather than an average algorithm. Regression on the runtime is most likely to be the best performer here. Classification and regression on the log of the runtime are almost certain to be better than the majority predictor. The methodology that delivers good results in almost all cases is regression on the runtime. It has good probabilities of both a good ranking and being better than the majority predictor both when considering the best algorithm and an average algorithm. Only when a part of the training data is deleted it delivers relatively poor performance.

The best algorithm of each methodology is not necessarily the same on all data sets. This should be taken into account when considering the probabilities in parentheses in Table 1. For example, the numbers show that the best algorithm that does regression on the runtime has the highest probability of being the overall best performing algorithm. This does however require identifying that best algorithm first. Any one algorithm of that methodology has a much lower chance of being best (the probability not in parentheses), whereas a case-based reasoning algorithm is more likely to perform best.

We observe that the majority classifier still has a non-negligible chance of being as least as good as sophisticated machine learning approaches. Its advantages over all the other approaches are its simplicity and that no problem features need to be computed, a task that can further impact the overall performance negatively because of the introduced overhead.

6.3. Determining the best machine learning algorithm

When using machine learning for algorithm selection in practice, one has to decide on a specific machine learning algorithm rather than methodology. Looking at Figure 1, we notice that individual algorithms within the classification and regression methodologies have better performance than case-based reasoning. While having established case-based reasoning as the best overall machine learning *methodology*, the question remains whether an individual machine learning *algorithm* can improve on that performance.

The `GaussianProcesses` algorithm to predict the runtime has the best performance for the largest number of data sets. But how likely is it to perform well in general? Our aim is to identify machine learning algorithms that will perform well in general rather than concentrating on the top performer on one data set only to find that it exhibits bad performance on different data. It is unlikely that one of the best algorithms here will be the best one on new data, but an algorithm with good performance on all data sets is more likely to exhibit good performance on unseen data. We performed a bootstrap estimate of the probability of an individual machine learning algorithm being better than the majority predictor by sampling from the set of data sets as described above. The results are summarised in Table 2.

The results confirm our intuition – the two algorithms that *always* perform better than the majority predictor are never the best algorithms while the algorithms that have the best performance on at least one data set – `GaussianProcesses` predicting the runtime and `RandomForest` and `LibSVM` with radial basis function for classification – have a significantly lower probability of performing better than the majority predictor.

Some of the machine learning algorithms within the classification and regression methodologies have a less than 50% chance of outperforming the majority predictor and do not appear in Table 2 at all. It is likely that the bad performance of these algorithms contributed to the relatively low rankings of their respective methodologies compared with case-based reasoning, where all machine learning algorithms exhibit good performance. The fact that the individual algorithms with the best performance do not belong to the methodology

	rank with full training data			better than	rank 1 with deleted training data		
methodology	1	2	3	majority predictor	25%	50%	75%
case-based reasoning	**52%** (5%)	29% (10%)	25% (31%)	**80%** (80%)	**80%** (7%)	**70%** (16%)	**39%** (6%)
classification	2% (33%)	3% (**51%**)	5% (**32%**)	60% (**99%**)	6% (**61%**)	8% (**40%**)	14% (**43%**)
regression	33% (**60%**)	**32%** (35%)	**28%** (24%)	67% (96%)	3% (6%)	7% (9%)	35% (23%)
regression-log	8% (2%)	19% (5%)	24% (14%)	75% (**99%**)	1% (26%)	15% (35%)	6% (27%)
statistical relational learning	6% (0%)	16% (0%)	18% (0%)	0% (0%)	10% (0%)	0% (0%)	5% (0%)

Table 1

Probabilities for each methodology ranking at a specific place with regard to the median performance of its algorithms and probability that this performance will be better than that of the majority predictor. We also show the probabilities that the median performance of the algorithms of a methodology will be the best for thinned out training data. The numbers in parentheses show the probability of a methodology ranking at a specific place or being better than the majority predictor with regard to the *maximum* performance of its algorithms. All probabilities are rounded to the nearest percent. The highest probabilities for each rank are in **bold**.

with the highest chances of good performance indicate that choosing the best individual machine learning algorithm regardless of methodology gives better overall performance.

The good performance of the case-based reasoning algorithms was expected based on the results presented in Table 1. All of the algorithms of this methodology have a very high chance of beating the majority predictor. The nearest-neighbour approach appears to be robust with respect to the number of neighbours considered.

The good results of the two best algorithms seem to contradict the expectations of the "No Free Lunch" theorems. These theorems state that superior performance of an algorithm in one scenario must necessarily be paid for by inferior performance in other scenarios. It has been shown however that the theorems do not necessarily apply in real-world scenarios because the underlying assumptions may not be satisfied [29]. In particular, the distribution of the best algorithms from the portfolio to problems is not random – it is certainly true that certain algorithms in the portfolio are the best on a much larger number of problems than others. Xu et al. [39] for example explicitly exclude some of the algorithms in the portfolio from being selected in certain scenarios.

7. Conclusions

In this paper, we investigated the performance of five different machine learning methodologies and many machine learning algorithms for algorithm selection on five data sets from the literature. We compared the performance not only among these methodologies and algorithms, but also to existing algorithm selection systems. To the best of our knowledge, we presented the first large-scale, quantitative comparison of machine learning methodologies and algorithms applicable to algorithm selection. We furthermore applied statistical relational learning to algorithm selection for the first time.

We used the performance of the simple majority predictor as a baseline and evaluated the performance of everything else in terms of it. This is a less favourable evaluation than found in many publications, but gives a better picture of the real performance improvement of algorithm portfolio techniques over just using a single algorithm. This method of evaluation clearly demonstrates that simply choosing the best individual algorithm in all cases can achieve better performance than sophisticated (and computationally expensive) approaches.

Our evaluation also showed the performance in terms of a simple rule learner, evaluated the effects of using only the set of the most predictive features instead of all features, looked at the influence the size of the algorithm portfolio has on the relative performance of the machine learning methodologies and quantified performance changes with training with partial data sets.

We demonstrate that methodologies and algorithms that have the best performance on one data set do not necessarily have good performance on all data sets. A non-intuitive result of our investigation is that deleting parts of the training data can help improve the overall performance, although the results are not clear enough to draw definitive conclusions from them.

machine learning methodology	algorithm	better than majority predictor
classification	`LADTree`	**100%**
regression	`LinearRegression`	**100%**
case-based reasoning	`IBk` with 1 neighbour	81%
case-based reasoning	`IBk` with 5 neighbours	81%
case-based reasoning	`IBk` with 3 neighbours	80%
case-based reasoning	`IBk` with 10 neighbours	80%
classification	`DecisionTable`	80%
classification	`FT`	80%
classification	`J48`	80%
classification	`JRip`	80%
classification	`RandomForest`	80%
regression	`GaussianProcesses`	80%
regression-log	`GaussianProcesses`	80%
regression-log	`SMOreg`	80%
regression-log	`LibSVM` ε	80%
regression-log	`LibSVM` ν	80%
classification	`REPTree`	61%
classification	`LibSVM` radial basis function	61%
regression	`REPTree`	61%
regression	`SMOreg`	61%
classification	`AdaBoostM1`	60%
classification	`BFTree`	60%
classification	`ConjunctiveRule`	60%
classification	`PART`	60%
classification	`RandomTree`	60%
regression-log	`LinearRegression`	60%
regression-log	`REPTree`	60%
classification	`J48graft`	59%
regression	`LibSVM` ν	59%

Table 2

Probability that a particular machine learning algorithm will perform better than the majority predictor on the full training data. We only show algorithms with a probability higher than 50%, sorted by probability. All probabilities are rounded to the nearest percent.

Based on a statistical simulation with bootstrapping, we give recommendations as to which algorithms are likely to have good performance. We identify *linear regression* and *alternating decision trees* as implemented in WEKA as particularly promising types of machine learning algorithms. In the experiments done for this paper, they always outperform the majority predictor. We focussed on identifying machine learning algorithms that deliver good performance in general. It should be noted that neither of these algorithms exhibited the best performance on any of the data sets, but the machine learning algorithms that did performed significantly worse on other data.

We furthermore demonstrated that *case-based reasoning* algorithms are very likely to achieve good performance and robust with respect to the number of past cases considered. Combined with the conceptual simplicity of nearest-neighbour approaches, it makes them a good starting point for researchers who want to use machine learning for algorithm selection, but are not machine learning experts themselves.

These recommendations are not meant to establish a set of machine learning algorithms that are the best in general for algorithm selection, but to provide guidance to practitioners who are unsure what machine learning to use after surveying the literature. In many cases, the choice of a particular machine learning algorithm will be influences by additional constraints that are particular to the specific scenario and cannot be considered here.

Finally, we demonstrated that the default parameters of the machine learning algorithms in WEKA already achieve very good performance and in most cases no tuning is required. While we were able to improve the performance in a few cases, finding the better configuration carried a high computational cost. In practice, few researcher will be willing or able to expend lots of resources to achieve small improvements.

Acknowledgments

We thank the anonymous reviewers of this paper and a previous version of this paper for their feedback; especially for helpful suggestions on how to improve the evaluation. We furthermore thank Kristian Kersting for pointing out SVM^{struct} to us. Holger Hoos, Kevin Leyton-Brown and Lin Xu pointed out and helped to fix erroneous results regarding SATzilla in a previous version of this paper. They also pointed out the need to adjust the timeout values in the training data they present for SATzilla on their website.

This research was supported in part by EPSRC grant EP/H004092/1 and a grant from Amazon Web Services that provided for some of the computational resources used in the evaluation. Lars Kotthoff is supported by a SICSA studentship.

References

[1] Sanjukta Bhowmick, Victor Eijkhout, Yoav Freund, Erika Fuentes, and David Keyes. Application of machine learning in selecting sparse linear solvers. Technical report, Columbia University, 2006.

[2] James E. Borrett, Edward P. K. Tsang, and Natasha R. Walsh. Adaptive constraint satisfaction: The quickest first principle. In *ECAI*, pages 160–164, 1996.

[3] Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso. PRODIGY: an integrated architecture for planning and learning. *SIGART Bull.*, 2:51–55, July 1991.

[4] Diane J. Cook and R. Craig Varnell. Maximizing the benefits of parallel search using machine learning. In *Proceedings of the 14th National Conference on Artificial Intelligence AAAI-97*, pages 559–564. AAAI Press, 1997.

[5] Eugene Fink. How to solve it automatically: Selection among Problem-Solving methods. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 128–136. AAAI Press, 1998.

[6] Cormac Gebruers, Brahim Hnich, Derek Bridge, and Eugene Freuder. Using CBR to select solution strategies in constraint programming. In *Proc. of ICCBR-05*, pages 222–236, 2005.

[7] Ian P. Gent, Christopher A. Jefferson, Lars Kotthoff, Ian Miguel, Neil C.A. Moore, Peter Nightingale, and Karen Petrie. Learning when to use lazy learning in constraint solving. In *ECAI*, pages 873–878, August 2010.

[8] Ian P. Gent, Lars Kotthoff, Ian Miguel, and Peter Nightingale. Machine learning for constraint solver design – a case study for the alldifferent constraint. In *3rd Workshop on Techniques for implementing Constraint Programming Systems (TRICS)*, pages 13–25, 2010.

[9] Alfonso E. Gerevini, Alessandro Saetti, and Mauro Vallati. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*, pages 350–353, 2009.

[10] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[11] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[12] Alessio Guerri and Michela Milano. Learning techniques for automatic algorithm portfolio selection. In *ECAI*, pages 475–479, 2004.

[13] Haipeng Guo and William H. Hsu. A Learning-Based algorithm selection meta-reasoner for the Real-Time MPE problem. In *Australian Conference on Artificial Intelligence*, pages 307–318, 2004.

[14] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[15] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Mach. Learn.*, 11:63–90, April 1993.

[16] Patricia D. Hough and Pamela J. Williams. Modern machine learning for automatic optimization algorithm selection. In *Proceedings of the INFORMS Artificial Intelligence and Data Mining Workshop*, November 2006.

[17] Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM.

[18] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC Instance-Specific algorithm configuration. In *ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 751–756. IOS Press, 2010.

[19] Ron Kohavi. A study of Cross-Validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, pages 1137–1143. Morgan Kaufmann, 1995.

[20] Lars Kotthoff, Ian Miguel, and Peter Nightingale. Ensemble classification for constraint solver configuration. In *CP*, pages 321–329, September 2010.

[21] Michail G. Lagoudakis and Michael L. Littman. Algorithm selection using reinforcement learning. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 511–518, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[22] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 556–572, London, UK, 2002. Springer-Verlag.

[23] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim Mcfadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI*, pages 1542–1543, 2003.

[24] Lionel Lobjois and Michel Lemaître. Branch and bound algorithm selection by performance prediction. In *AAAI '98/IAAI '98: Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pages 353–358, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[25] Steven Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1:7–43, 1996.

[26] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[27] Luca Pulina and Armando Tacchella. A multi-engine solver for quantified boolean formulas. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pages 574–589, Berlin, Heidelberg, 2007. Springer-Verlag.

[28] Luca Pulina and Armando Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14 (1):80–116, 2009.

[29] R. Bharat Rao, Diana Gordon, and William Spears. For every generalization action, is there really an equal and opposite reaction? Analysis of the conservation law for generalization performance. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 471–479. Morgan Kaufmann, 1995.

[30] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[31] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1989.

[32] Mark Roberts and Adele E. Howe. Learned models of performance for many planners. In *ICAPS 2007 Workshop AI Planning and Learning*, 2007.

[33] Bryan Silverthorn and Risto Miikkulainen. Latent class models for algorithm portfolio methods. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[34] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41:6:1–6:25, January 2009.

[35] David H. Stern, Horst Samulowitz, Ralf Herbrich, Thore Graepel, Luca Pulina, and Armando Tacchella. Collaborative expert portfolio management. In *AAAI*, pages 179–184, 2010.

[36] Matthew J. Streeter and Stephen F. Smith. New techniques for algorithm portfolio design. In *UAI*, pages 519–527, 2008.

[37] Sanjiva Weerawarana, Elias N. Houstis, John R. Rice, Anupam Joshi, and Catherine E. Houstis. PYTHIA: A knowledge-based system to select scientific algorithms. *ACM Trans. Math. Softw.*, 22(4):447–468, 1996.

[38] David H. Wolpert. The supervised learning No-Free-Lunch theorems. In *Proc. 6th Online World Conference on Soft Computing in Industrial Applications*, pages 25–42, 2001.

[39] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res. (JAIR)*, 32:565–606, 2008.

[40] Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for Portfolio-Based selection. In *Twenty-Fourth Conference of the Association for the Advancement of Artificial Intelligence (AAAI-10)*, pages 210–216, 2010.