

Algorithm Runtime Prediction: Methods & Evaluation (Extended Abstract)*.

Frank Hutter **Lin Xu** **Holger H. Hoos** **Kevin Leyton-Brown**
University of Freiburg University of British Columbia
fh@cs.uni-freiburg.de {xulin730, hoos, kevinlb}@cs.ubc.ca

Abstract

Perhaps surprisingly, it is possible to predict how long an algorithm will take to run on a previously unseen input, using machine learning techniques to build a model of the algorithm’s runtime as a function of problem-specific instance features. Such models have many important applications and over the past decade, a wide variety of techniques have been studied for building such models. In this extended abstract of our 2014 AI Journal article of the same title, we summarize existing models and describe new model families and various extensions. In a comprehensive empirical analysis using 11 algorithms and 35 instance distributions spanning a wide range of hard combinatorial problems, we demonstrate that our new models yield substantially better runtime predictions than previous approaches in terms of their generalization to new problem instances, to new algorithms from a parameterized space, and to both simultaneously.

1 Introduction

NP-complete problems are ubiquitous in AI. Luckily, while these problems may be hard to solve on worst-case inputs, it is often feasible to solve even large problem instances that arise in practice. Less luckily, state-of-the-art algorithms often exhibit extreme runtime variation across instances from realistic distributions, even when problem size is held constant, and conversely the same instance can take dramatically different amounts of time to solve depending on the algorithm used [Gomes *et al.*, 2000]. There is little theoretical understanding of what causes this variation. Over the past decade, a considerable body of work has shown how to use supervised machine learning methods to build regression models that provide approximate answers to this question based on given algorithm performance data. We refer to such models as *empirical performance models (EPMs)*. These models are useful in a variety of practical contexts:

- **Algorithm selection.** This classic problem of selecting the best from a given set of algorithms on a per-instance

*This paper is an invited extended abstract of our 2014 AI Journal article [Hutter *et al.*, 2014b]

basis [Rice, 1976; Smith-Miles, 2009; Kotthoff, 2014] has been successfully addressed by using EPMs to predict the performance of all candidate algorithms and selecting the one predicted to perform best [Brewer, 1995; Lobjois and Lemaître, 1998; Fink, 1998; Howe *et al.*, 2000; Nudelman *et al.*, 2003; Roberts and Howe, 2007; Xu *et al.*, 2008; Kotthoff *et al.*, 2012].

- **Parameter tuning and algorithm configuration.** EPMs are useful for these problems in at least two ways. First, they can model the performance of a parameterized algorithm dependent on the settings of its parameters; in a sequential model-based optimization process, one alternates between learning an EPM and using it to identify promising settings to evaluate next [Jones *et al.*, 1998; Bartz-Beielstein *et al.*, 2005; Hutter *et al.*, 2011]. Second, EPMs can model algorithm performance dependent on both problem instance features and algorithm parameter settings; such models can then be used to select parameter settings with good predicted performance on a per-instance basis [Hutter *et al.*, 2006].
- **Generating hard benchmarks.** An EPM for one or more algorithms can be used to set the parameters of existing benchmark generators in order to create instances that are hard for the algorithms in question [Leyton-Brown *et al.*, 2009].
- **Gaining insights into instance hardness and algorithm performance.** EPMs can be used to assess which instance features and algorithm parameter values most impact empirical performance. Some models support such assessments directly [Ridge and Kudenko, 2007; Mersmann *et al.*, 2013; Hutter *et al.*, 2014a]. For other models, generic feature selection methods, such as forward selection, can be used to identify a small number of key model inputs (often fewer than five) that explain algorithm performance almost as well as the whole set of inputs [Leyton-Brown *et al.*, 2009; Hutter *et al.*, 2013].

While these applications motivate our work, we do not discuss them in detail in our article; instead, we focus on the models themselves. The idea of modeling algorithm runtime is no longer new; however, we have made substantial recent progress in making runtime prediction methods more general, scalable and accurate. After a comprehensive review of past

work on runtime prediction from many separate communities, our AI Journal article [Hutter *et al.*, 2014b] makes four new contributions:

1. We describe new, more sophisticated modeling techniques (based on random forests and approximate Gaussian processes) and methods for modeling runtime variation arising from the settings of a large number of (both categorical and continuous) algorithm parameters.
2. We introduce new instance features for propositional satisfiability (SAT), travelling salesperson (TSP) and mixed integer programming (MIP) problems—in particular, novel probing features and timing features—yielding comprehensive sets of 138, 121, and 64 features for SAT, MIP, and TSP, respectively.
3. To assess the impact of these advances and to determine the current state of the art, we performed what we believe is the most comprehensive evaluation of runtime prediction methods to date. Specifically, we evaluated all methods of which we are aware on performance data for 11 algorithms and 35 instance distributions spanning SAT, TSP and MIP and considering three different problems: predicting runtime on novel instances, novel parameter configurations, and both novel instances *and* configurations.
4. Techniques from the statistical literature on survival analysis offer ways to better handle data from runs that were terminated prematurely. While these techniques were not used in most previous work—leading us to omit them from our broad empirical evaluation—we show how to leverage them to achieve further improvements to our best-performing model, random forests.

In this extended abstract of our AI Journal article, we provide a high-level description of the modeling techniques (Section 3) and instance features (Section 4) and give some exemplary empirical results (Section 5).

2 Problem Definition

We describe a problem instance by a list of m features $\mathbf{z} = [z_1, \dots, z_m]^\top$, drawn from a given *feature space* \mathcal{F} . These features must be computable by a piece of problem-specific code (usually provided by a domain expert) that efficiently extracts characteristics for any given problem instance (typically, in low-order polynomial time w.r.t. the size of the given problem instance). We define the *configuration space* of a parameterized algorithm with k parameters $\theta_1, \dots, \theta_k$ with respective domains $\Theta_1, \dots, \Theta_k$ as a subset of the cross-product of parameter domains: $\Theta \subseteq \Theta_1 \times \dots \times \Theta_k$. The elements of Θ are complete instantiations of the algorithm’s k parameters, and we refer to them as *configurations*. We note that parameters can be numerical (continuous- or real-valued) or categorical (with finite unordered domain, as, e.g., for parameters that govern which of several heuristics to use).

Given an algorithm \mathcal{A} with configuration space Θ and a distribution of instances with feature space \mathcal{F} , an EPM is a stochastic process that defines a probability distribution over performance measures for each combination of a parameter

configuration $\theta \in \Theta$ of \mathcal{A} and a problem instance with features $\mathbf{z} \in \mathcal{F}$. The prediction of an entire distribution allows us to assess the model’s *confidence* at a particular input. While this is essential in some applications (e.g., in model-based algorithm configuration), many previous models do not quantify uncertainty. We thus chiefly evaluate our models in terms of their mean predictions.

3 Modeling Techniques

We evaluated the following algorithm runtime prediction methods from previous work:

- **RR**: ridge regression with quadratic basis function expansion and forward selection as in early versions of SATzilla [Leyton-Brown *et al.*, 2009; Xu *et al.*, 2008];
- **SP**: SPORE-FoBa, a variant of RR that performs forward-backward feature selection [Huang *et al.*, 2010];
- **NN**: neural networks (as used by Smith-Miles and van Hemert [2011]); and
- **RT**: regression trees (as used by Bartz-Beielstein and Markon [2004]).

Out of these models, only regression trees natively handle categorical inputs. In order to apply the other model types to construct EPMs for algorithms with categorical parameters, we used a 1-in-K encoding (also known as 1-hot encoding). We also used two methods newly in the context of runtime prediction:

- **GP**: approximate Gaussian processes [Rasmussen and Williams, 2006], equipped with a new kernel for categorical parameters; and
- **RF**: random forests [Breiman, 2001], adapted with a new method to quantify predictive uncertainties and a new method for choosing split points to yield linear interpolations (and uncertainty estimates that grow with distance to observed data points) in the limit of an infinite number of trees.

4 Features

Instance features are inexpensively computable, problem-dependent characteristics that distinguish instances from one another. In our AI Journal paper we describe a large set of 138, 121, and 64 features for SAT, MIP, and TSP, respectively. In particular, we review many existing features and also introduce various new features, including the following two general classes:

- **Probing features**: we execute a cheap algorithm for the actual problem instance and keep track of several statistics; e.g., counting the number of clauses a DPLL SAT solver learns in 2 seconds;
- **Timing features**: we measure the time the computation of various features take; this timing information can be a very useful feature in itself.

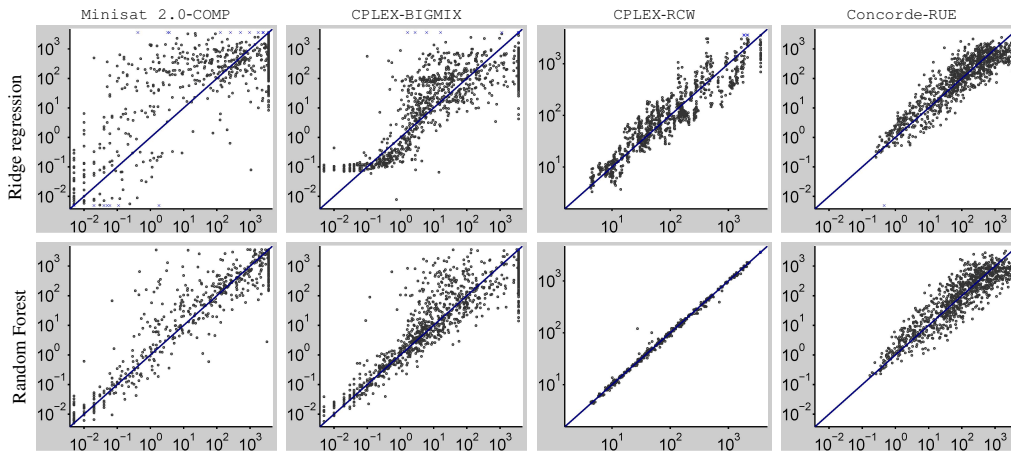


Figure 1: Visual comparison of models for runtime predictions on previously unseen test instances. The data sets used in each column are shown at the top. The x -axis of each scatter plot denotes true runtime and the y -axis 2-fold cross-validated runtime as predicted by the respective model; each dot represents one instance. Predictions above 3 000 or below 0.001 seconds are denoted by a blue cross rather than a black dot.

Domain	RMSE					
	RR	SP	NN	PP	RT	RF
Minisat 2.0-COMPETITION	1.01	1.25	0.62	0.92	0.68	0.47
CryptoMinisat-INDU	0.94	0.99	0.94	0.9	0.91	0.72
tnm-RANDSAT	1.01	1.05	0.94	0.93	1.22	0.88
CPLEX-BIGMIX	$2.7 \cdot 10^8$	0.93	1.02	1	0.85	0.64
CPLEX-CORLAT	0.49	0.52	0.53	0.46	0.62	0.47
Concorde-RUE	0.41	0.43	0.43	0.42	0.59	0.45

Table 1: Quantitative comparison of models for runtime predictions on previously unseen instances (for 6 representative benchmark sets out of 35 in the full article). We report 10-fold cross-validation performance. Lower RMSE values are better (0 is optimal). Boldface indicates performance not statistically significantly different from the best method in each row based on a Wilcoxon signed rank test.

5 Empirical Results

We now summarize some results that are representative of our findings about runtime prediction on (a) new instances and (b) new instances *and* configurations.

5.1 Predictions for New Instances

Table 1 provides quantitative results for all benchmarks, and Figure 1 visualizes some results in more detail. At the broadest level, we can conclude that most of the methods were able to capture enough information about algorithm performance on training data to make meaningful predictions on test data, most of the time: easy instances tended to be predicted as being easy, and hard ones as being hard. Take, for example the case of predicting the runtime of *Minisat 2.0* on a heterogeneous mix of SAT competition instances (see the leftmost column in Figure 1 and the top row of Table 1). *Minisat 2.0* runtimes varied by almost six orders of magnitude, while predictions made by the better models were rarely off by more than one order of magnitude (outliers may draw the eye in the

scatterplot, but quantitatively, the RMSE for predicting \log_{10} runtime was low—*e.g.*, 0.47 for random forests, which means an average misprediction of a factor of $10^{0.47} < 3$). While the models were certainly not perfect, note that even the relatively poor predictions of ridge regression tended to be accurate within about an order of magnitude, which was sufficient to enable the portfolio-based algorithm selector *SATzilla* [Xu *et al.*, 2008] to win five medals in each of the 2007 and 2009 SAT competitions.

In our experiments, random forests were the overall winner among the different methods, yielding the best predictions in terms of all our quantitative measures. We attribute their strong performance on highly heterogeneous data sets to the fact that, as a tree-based approach, they can model very different parts of the data separately; in contrast, the other methods allow the fit in a given part of the space to be influenced more by data in distant parts of the space. Indeed, the ridge regression variants made extremely bad predictions for some outlying points on *CPLEX-BIGMIX*. For the more homogeneous MIP data sets, either random forests or projected processes performed best, often followed closely by ridge regression. In terms of computational requirements, random forests were also among the cheapest methods, taking between 0.1 and 11 seconds to learn a model.

5.2 Predictions for New Instances and New Configurations

We now examine the most interesting case, where test instances and algorithm parameter configurations were both previously unseen. Table 2 provides quantitative results of model performance based on $n = 10\,000$ training data points, and Figure 2 visualizes performance. Overall, we note that the best models generalized to new configurations *and* to new instances almost as well as to either alone. On the most heterogeneous data set, *CPLEX-BIGMIX*, we once again witnessed extremely poorly predicted outliers for the ridge regression variants, but in all other cases, the models captured the large spread in

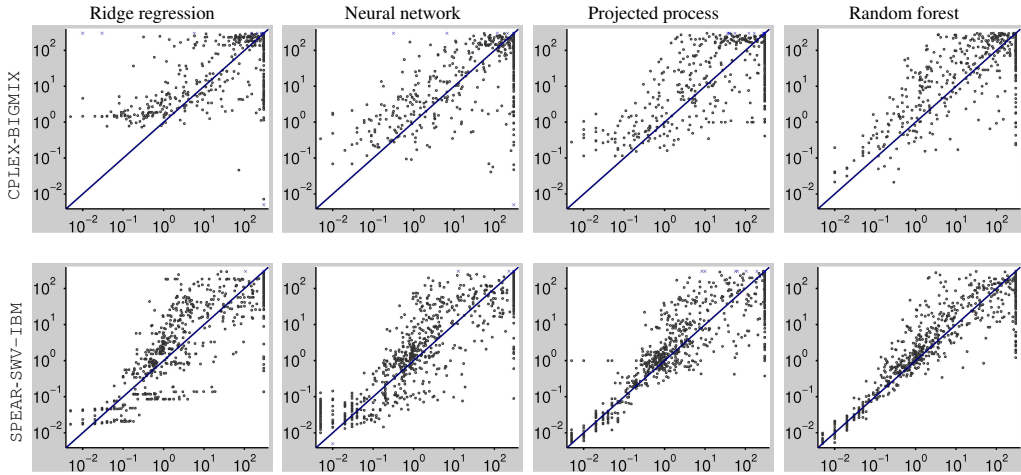


Figure 2: Visual comparison of models for runtime predictions on pairs of previously unseen test configurations and instances. In each scatter plot, the x -axis shows true runtime and the y -axis cross-validated runtime as predicted by the respective model. Each dot represents one combination of an unseen instance and parameter configuration. Predictions above 3 000 or below 0.001 seconds are denoted by a blue cross rather than a black dot.

Domain	RMSE					
	RR	SP	NN	PP	RT	RF
CPLEX-BIGMIX	$> 10^{100}$	4.5	0.68	0.78	0.74	0.55
CPLEX-CORLAT	0.53	0.57	0.56	0.53	0.67	0.49
CPLEX-REG	0.17	0.19	0.19	0.19	0.24	0.17
CPLEX-RCW	0.1	0.12	0.12	0.12	0.12	0.09
CPLEX-CR	0.41	0.43	0.42	0.42	0.52	0.38
CPLEX-CRR	0.35	0.37	0.37	0.39	0.43	0.32
SPEAR-IBM	0.58	11	0.54	0.52	0.57	0.44
SPEAR-SWV	0.58	0.61	0.63	0.54	0.55	0.44
SPEAR-SWV-IBM	0.65	0.69	0.65	0.65	0.59	0.45

Table 2: Root mean squared error (RMSE) obtained by various models for runtime predictions on unseen instances and configurations. Boldface indicates the best average performance in each row. Models were based on 10 000 data points.

formed best on the most heterogeneous data sets. Figure 2 also shows some qualitative differences in predictions: for example, ridge regression, neural networks, and projected processes sometimes overpredicted the runtime of the shortest runs, while the tree-based methods did not have this problem. Random forests performed best in all cases.

Finally, Figure 3 qualitatively compares true runtimes to those predicted by random forests and ridge regression using the heterogeneous data set SPEAR-SWV-IBM. We note that the true heatmaps are very similar to those predicted by random forests, while the non-tree-based methods (here: ridge regression) only captured instance hardness, failing to distinguish good from bad configurations (even in the simplest case of predictions for training instances and training configurations).

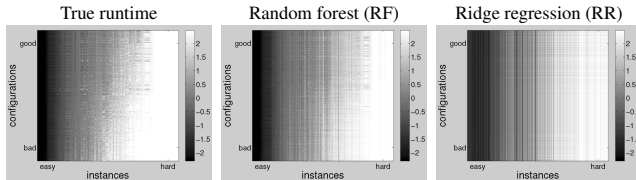


Figure 3: True and predicted runtime matrices for dataset SPEAR-SWV-IBM, for previously unseen test instances and test configurations. For example, the left heatmap shows the true runtimes for the cross product of 500 test configurations of SPEAR and the 685 test instances of the SWV-IBM benchmark set. Darker greyscale values represent faster runs, *i.e.*, instances on the right side of each heatmap are hard (they take longer to solve), and configurations at the top of each heatmap are good (they solve instances faster).

runtimes (above 5 orders of magnitude) quite well. As in the experiments in Section 5.1, the tree-based approaches, which model different regions of the input space independently, per-

6 Conclusion

In this invited extended abstract of our AI Journal paper [Hutter *et al.*, 2014b], we summarized existing and new techniques for predicting algorithm runtime and evaluated their performance in a comprehensive empirical analysis. Particularly noteworthy is the rather good predictability of runtime for new problem instances *and* new configurations of parameterized algorithms. We encourage the interested reader to consult our full journal article for a complete account of our methods and findings. Overall, in this article, we show that the performance prediction methods we studied are fast, general, and achieve good, robust performance. We hope they will be useful to a wide variety of researchers who seek to model algorithm performance for algorithm analysis, scheduling, algorithm portfolio construction, automated algorithm configuration, and other applications. The Matlab source code for our models, the data and source code to reproduce our experiments, and an online appendix containing additional experimental results, are available at <http://www.cs.ubc.ca/labs/beta/Projects/EPMs>.

References

- [Bartz-Beielstein and Markon, 2004] T. Bartz-Beielstein and S. Markon. Tuning search algorithms for real-world applications: a regression tree based approach. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC'04)*, pages 1111–1118, 2004.
- [Bartz-Beielstein *et al.*, 2005] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC'05)*, pages 773–780, 2005.
- [Breiman, 2001] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [Brewer, 1995] E. A. Brewer. High-level optimization via automated statistical modeling. In *Proceedings of the 5th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPOPP-95)*, pages 80–91, 1995.
- [Fink, 1998] E. Fink. How to solve it automatically: Selection among problem-solving methods. In *Proceedings of the Fourth International Conference on AI Planning Systems*, pages 128–136. AAAI Press, 1998.
- [Gomes *et al.*, 2000] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1):67–100, 2000.
- [Howe *et al.*, 2000] A. E. Howe, E. Dahlman, C. Hansen, M. Scheetz, and A. Mayrhauser. Exploiting competitive planner performance. In *Recent Advances in AI Planning (ECP'99)*, LNCS, pages 62–72. 2000.
- [Huang *et al.*, 2010] L. Huang, J. Jia, B. Yu, B. Chun, P. Maniatis, and M. Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Proceedings of the 23rd Conference on Advances in Neural Information Processing Systems (NIPS'10)*, pages 883–891, 2010.
- [Hutter *et al.*, 2006] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP'06)*, LNCS, pages 213–228, 2006.
- [Hutter *et al.*, 2011] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th Conference on Learning and Intelligent Optimization (LION'11)*, LNCS, pages 507–523, 2011.
- [Hutter *et al.*, 2013] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In *Proceedings of the 7th Conference on Learning and Intelligent Optimization (LION'13)*, LNCS, pages 364–381, 2013.
- [Hutter *et al.*, 2014a] F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on Machine Learning (ICML'14)*, pages 754–762, June 2014.
- [Hutter *et al.*, 2014b] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206(0):79–111, January 2014.
- [Jones *et al.*, 1998] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.
- [Kotthoff *et al.*, 2012] L. Kotthoff, I. P. Gent, and I. Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Commun.*, 25(3):257–270, 2012.
- [Kotthoff, 2014] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.
- [Leyton-Brown *et al.*, 2009] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4):1–52, 2009.
- [Lobjois and Lemaître, 1998] L. Lobjois and M. Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 353–358, 1998.
- [Mersmann *et al.*, 2013] O. Mersmann, B. Bischl, H. Trautmann, M. Wagner, J. Bossek, and F. Neumann. A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence (AMAI)*, pages 32 pages, published online, 2013.
- [Nudelman *et al.*, 2003] E. Nudelman, K. Leyton-Brown, G. Andrew, C. Gomes, J. McFadden, B. Selman, and Y. Shoham. Satzilla 0.9. Solver description, 2003 SAT Competition, 2003.
- [Rasmussen and Williams, 2006] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.
- [Rice, 1976] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.
- [Ridge and Kudenko, 2007] E. Ridge and D. Kudenko. Tuning the performance of the MMAS heuristic. In *Proceedings of the International Workshop on Engineering Stochastic Local Search Algorithms (SLS'2007)*, LNCS, pages 46–60, 2007.
- [Roberts and Howe, 2007] Mark Roberts and Adele Howe. Learned models of performance for many planners. In *ICAPS 2007 Workshop AI Planning and Learning*, 2007.
- [Smith-Miles and van Hemert, 2011] Kate Smith-Miles and Jano van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 61:87–104, 2011.
- [Smith-Miles, 2009] K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6:1–6:25, 2009.
- [Xu *et al.*, 2008] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, June 2008.