Improved Features for Runtime Prediction of Domain-Independent Planners

Chris Fawcett
University of British Columbia
fawcettc@cs.ubc.ca

Mauro Vallati
University of Huddersfield
m.vallati@hud.ac.uk

Frank Hutter
University of Freiburg
fh@informatik.uni-freiburg.de

Jörg Hoffmann
Saarland University
hoffmann@cs.uni-saarland.de

Holger H. Hoos
University of British Columbia
hoos@cs.ubc.ca

Kevin Leyton-Brown
University of British Columbia
kevinlb@cs.ubc.ca

Abstract

State-of-the-art planners often exhibit substantial runtime variation, making it useful to be able to efficiently predict how long a given planner will take to run on a given instance. In other areas of AI, such needs are met by building so-called *empirical performance models (EPMs)*, statistical models derived from sets of problem instances and performance observations. Historically, such models have been less accurate for predicting the running times of planners. A key hurdle has been a relative weakness in *instance features* for characterizing instances: mappings from problem instances to real numbers that serve as the starting point for learning an EPM. We propose a new, extensive set of instance features for planning, and investigate its effectiveness across a range of model families. We built EPMs for various prominent planning systems on several thousand benchmark problems from the planning literature and from IPC benchmark sets, and conclude that our models predict runtime much more accurately than the previous state of the art. We also study the relative importance of these features.

Introduction

The field of automated plan generation has significantly advanced through powerful, new domain-independent planners. These planners often exhibit dramatic runtime variation, and in particular no one planner dominates all others. Ideally, given an unseen problem instance, we should be able to automatically select "the right planner", which would be possible if we could *predict* how long a given planner will take to solve a given instance. Such predictions are possible using so-called *empirical performance models (EPMs)*, which are generally constructed as follows. First, a solver for a given problem (here: planning) is run on a large number of problem instances from a distribution or benchmark set of interest. For each instance, the solver's performance (e.g., runtime) is recorded; furthermore, a set of *instance features* is computed. Each instance feature is a real number that summarizes a potentially important property of the instance. Taken as a whole, the set of instance features constitutes a fixed-size instance "fingerprint". A predictive model is then learned as a mapping from instance features to solver performance (e.g., minimizing root mean squared error on the training set, with a penalty for model complexity). Finally, an EPM is evaluated on a separate (*test*) set of instances to ensure that it generalizes beyond its training data; in planning, an aspect of particular interest is whether the prediction generalizes *across domains*.

EPMs are well established within AI, with successes across a broad range of problems including, for example, SAT, MIP, and TSP (see, e.g., Brewer (1994); Leyton-Brown, Nudelman, and Shoham (2002); Leyton-Brown, Nudelman, and Shoham (2009); Nudelman et al. (2004); Xu et al. (2008); Hutter et al. (2014); Smith-Miles, van Hemert, and Lim (2010)). EPMs have also been considered in the planning literature. Fink (1998) predicted runtime using linear regression, based only on a problem size feature. Howe et al. (1999) built regression models based on five features to predict performance of six planners. Subsequent work by Roberts et al. (2008; 2009) included comprehensive features regarding PDDL statistics, considered additional planners, and explored more complex models. Most recently, Cenamor et al. (2012; 2013) grew the feature set to include statistics about the causal graph and domain transition graphs (Helmert 2006). These state-of-the-art EPMs for planning are generally able to predict whether a given planner would solve a given problem instance (at least on previously observed domains), but much less effective at predicting runtime.

One of the keys to building accurate EPMs lies in identifying a good set of instance features. Our work establishes a new, extensive set of instance features for planning, and investigates its effectiveness across a range of model families. In particular, we adopt features from SAT via a SAT encoding of planning, and for the first time in planning we extract features by *probing* the search. We build EPMs for variants of state-of-the-art planning systems, using an extensive benchmark set. Our main finding is that the resulting EPMs predict runtime more accurately than the previous state of the art. We also study the relative importance of our features, yielding insight into how to build simpler and faster models.

Features

Our new feature set derives 311 values from a given PDDL domain and instance file. Table 1 gives an overview, also listing the extraction costs associated with each feature group. For most instances, this cost was low, but for some instances the SAT encoding was prohibitively expensive. Most sig-

Class	# succ.	cost	Q50	Q90	# feat.
PDDL domain file	7333	trivial	0.048	0.075	18
PDDL instance file	7333	trivial	0.048	0.075	7
PDDL requirements	7333	trivial	0.048	0.075	24
LPG preprocessing	5228	cheap	0.075	3.230	6
Torchlight	4435	cheap	0.029	0.671	10
FDR translation	6590	moderate	0.365	33.205	19
FDR	6590	moderate	0.365	33.205	19
Causal&DT graph	4109	moderate	0.409	35.134	41
FD preprocessing	6654	moderate	0.477	40.722	8
FD probing	4948	moderate	1.419	34.295	16
SAT representation	6344	expensive	7.802	1800	115
Success & timing	n/a	various	n/a	n/a	28
Total	–	–	–	–	311

Table 1: Summary of our features, by class (the classes are explained in the text). "# succ." is the number of instances (out of 7571) for which the feature was successfully extracted. "cost" is the "cost class" of each extractor used in Tables 3 and 4. "Q50" and "Q90" are the median and 90% quantile of the extraction times (in CPU seconds) for the entire class, assuming that no other features have been extracted.

nificantly, we introduce so-called *probing features* for planning based on information gleaned from (short) runs of Fast Downward (Helmert 2006). We also employ instance features based on Torchlight, a recent tool for analyzing local search topology under h^+ (Hoffmann 2011). Furthermore, we employ SAT-based feature extractors, leveraging extensive prior work on EPMs in that area. Overall, our features fall into the following eight groups.

PDDL. We extend the 16 PDDL domain features, 3 instance features, and 13 language requirement features explored by Roberts et al. (2008) with 2 additional domain features covering the use and number of object types, and 4 additional instance features adding information about constants, "=" predicate usage, and function assignments in initial conditions. We also add 11 additional language requirement features from newer versions of the PDDL specification.

FDR. We use the translation and preprocessing tools built into Fast Downward to translate the given PDDL domain and instance into the finite domain representation used by Fast Downward (and for extracting our "FD probing" features). We gather features from the console output of the translation process (such as the number of removed operators and propositions and number of implied preconditions added), the finite domain representation created (such as the number of variables, number of mutex groups, and statistics over operator effects), as well as from the output of the preprocessing process (such as the percentage of variables, mutex groups and operators deemed necessary, whether the instance is deemed solvable in polynomial time, and the resulting task size).

Causal and DT graph. Using the finite domain representation created in the extraction of our "FDR" features, we extract the 41 features of the causal and domain transition graphs introduced by Cenamor et al. (2012; 2013).

LPG preprocessing. For this extractor, we run LPG-td (Gerevini, Saetti, and Serina 2003) until the end of its preprocessing. We extract features such as the number of facts, the number of "significant" instantiated operators and facts, and the number of mutual exclusions between facts. (As LPG is highly parameterized, it is difficult to design search probes as we do for FD; we leave this open for future work.)

Torchlight. We restrict ourselves to Torchlight's search-sampling variant, as this was most robust (highest success rate) across our instance set. We extract success (sample state proved to not be a local minimum) and dead-end percentages, and statistics on exit distance bounds as well as the preprocessing.

FD probing. We run Fast Downward for 1 CPU second, using the same heuristic settings as for LAMA, and measure features from the resulting planning trajectory, such as the number of reasonable orders removed, landmark discovery and composition, the number of graph edges, the number of initial and goal landmarks, and statistics on and relative improvement over the initial and final heuristic values.

SAT representation. We use the SAT translation of Mp (Rintanen 2012) to produce a CNF with a planning horizon of 10. If the creation of this instance is successful, we use the SAT feature extraction code of SATzilla 2012 (Xu et al. 2012) to extract 115 features from 12 classes: problem size features, variable-clause graph features, variable graph features, clause graph features, balance features, as well as features based on proximity to Horn formula, DPLL probing, LP-based, local search probing, clause learning, and survey propagation (see Hutter et al. (2014) for details on these classes).

Success & timing. For each of our seven extraction procedures, we record whether extraction was successful (a binary feature) and record as a numeric feature the CPU time required for extraction. The SAT feature extractors additionally report 10 more timing features for extraction time of various subcomponents.

Experiment Design and Methodology

All planners and feature extractors were run on a cluster with computing nodes equipped with two Intel Xenon X5650 hexacore CPUs with 12MB of cache and 24GB of RAM each, running Red Hat Enterprise Linux Server 5.3. Each of our runs was limited to a single core, and was given runtime and memory limits of 1800 CPU seconds and 8GB respectively, as used in the deterministic tracks of recent IPCs.

For each planner run, we recorded the overall result: success (found a valid plan or proved unsolvable), crashed, timed-out, ran out of memory, or unsupported instance/domain. Unsuccessful planner runs were assigned a runtime equal to the 1800 second cutoff.

Benchmarks and Planners. We selected variants of seven planners, based on their outstanding performance in various tracks of previous IPCs and/or the use of very different planning approaches (see Table 2).

Fast Downward has many configurations; we use the same default configuration as in previous work (Fawcett et al. 2011;

Planner	Literature	IPC
Arvand	Nakhost et al. (2011)	2011
Fast Downward	Helmert (2006), Seipp et al. (2012)	2011
FF	Hoffmann and Nebel (2001), Hoffmann (2003)	2002,2004
LAMA	Richter and Westphal (2008), Richter et al. (2011)	2008,2011
LPG	Gerevini, Saetti, and Serina (2003)	2002,2004
Mp	Rintanen (2012)	2011
Probe	Lipovetzky and Geffner (2011)	2011

Table 2: Overview of the planners in our experiments.

Seipp et al. 2012). We consider three versions of FF: the standard FF v2.3 (Hoffmann and Nebel 2001), FF-X which supports derived predicates (Thiébaux, Hoffmann, and Nebel 2005), and Metric-FF which supports action costs (Hoffmann 2003). We include the default version of LPG as well as a version (LPG-contra) allowing the instantiation of actions with the same objects for multiple parameters if permitted by the domain.

We gathered as many available PDDL planning instances as possible, with the only restriction being that they were supported (but not necessarily solvable) by at least one of the planners used in our study. Considering all different encodings available for the same planning problem, we ended up with a set of 7571 planning instances collected from the following sources: (i) IPC'98 and IPC'00; (ii) deterministic tracks of IPC'02 through IPC'11 and learning tracks of IPC'08 and IPC'11 (demo problems and learning problems, when provided by the organizers); (iii) FF benchmark library; (iv) Fast Downward benchmark library; (v) UCPOP Strict benchmarks, and; (vi) Sodor and Stek domains used by Roberts et al. (2008).

Building Empirical Performance Models. Using the EPM construction process outlined by Hutter et al. (2014), we performed experiments with different planners, different benchmark sets and different cross-validation approaches. Due to the fact that the runtimes of our planners varied from 0.01 CPU seconds to near the full 1800 second runtime cutoff, in all cases we trained our models to predict log-runtime rather than absolute runtime. This log-transformation has been previously demonstrated to be highly effective under similar circumstances (e.g., by Hutter et al. (2014)). We investigate two cross-validation approaches: *10-fold cross-validation* on a uniform random permutation of our instances (a standard method where nine slices are used for training and the tenth for testing), as well as *leave-one-domain-out* cross-validation where we train on instances from all domains but one, and test on the held-out domain, for each of the 180 domains in our instance set (the same methods were used by Cenamor et al. (2012)). First, we assessed the performance of various regression models (linear regression, neural networks, Gaussian processes, regression trees, and random forests) using the data by Roberts et al. (2008). While all models were competitive with the nearest neighbour approach by Roberts et al. (2008), consistent with the findings of Hutter et al. (2014), random forests performed best (somewhat better than the nearest neighbour approach). Since random forests are also automatic feature selectors, we used them for the remainder of the paper. Next, we describe experiments for our full 7571-instance set.

	RMSE of $\log_{10}$(runtime)								
Planner	None	DF	+IF	+LF	+NF-0	+Ce	+NF-1	+NF-2	All
Arvand	1.69	1.07	0.67	0.65	0.68	0.47	0.41	**0.39**	**0.37**
Fast Downward	1.64	0.98	0.6	0.58	0.61	0.38	0.37	**0.34**	**0.34**
FF v2.3	2.44	1.44	0.81	0.81	0.82	0.79	0.69	**0.67**	**0.64**
FF-X	2.41	1.45	0.73	0.73	0.75	0.72	0.62	0.59	**0.53**
LAMA	1.61	1.00	0.63	0.61	0.64	0.44	0.38	0.35	**0.32**
LPG	2.00	1.1	0.55	0.53	0.54	0.51	0.44	**0.41**	**0.42**
LPG-contra	2.00	1.11	0.55	0.55	0.57	0.53	**0.45**	**0.43**	**0.43**
Metric-FF v2.1	2.37	1.51	0.77	0.78	0.79	0.76	0.68	0.64	**0.61**
Mp	2.28	1.43	0.61	0.62	0.63	0.62	0.54	0.51	**0.41**
Probe	2.14	1.27	0.61	0.6	0.6	0.59	0.51	0.50	**0.45**

Table 3: Cross-validated RMSE of $\log_{10}$(runtime) for random forest models using feature subsets that are increasingly expensive to extract. The table shows 10-fold cross-validated performance using a uniform random permutation of our entire set of 7571 problem instances. Column 'None' gives the performance of a featureless mean predictor. The following three subsets contain the 32 features used by Roberts et al. (2008): PDDL domain features only (DF), DF plus PDDL instance features (IF), and IF plus PDDL requirements features (LF). In the remaining four subsets, NF-0 is comprised of the 32 previous PDDL features plus an additional 17 produced by our extractors, Ce additionally includes the graph features by Cenamor et al. (2012; 2013), NF-1 the features marked "cheap" in Table 1, and NF-2 the features marked "moderate". Finally, the All column also includes the features marked "expensive", and contains all of our 311 features. Bold text indicates the best result, or results that are not statistically significantly different from the best result across the cross-validation folds.

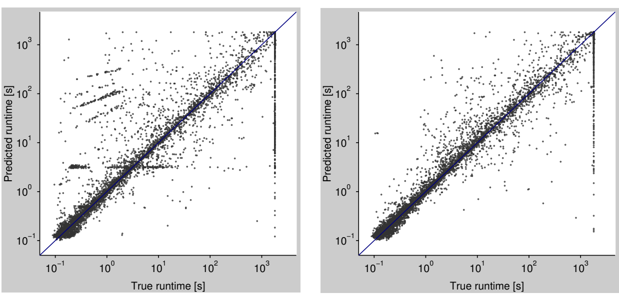
Experiments: Runtime Prediction

Table 3 gives detailed performance results for prediction using our random forest models, with 10-fold cross-validation on a uniform random permutation of our full set of 7571 problem instances. Table 4 shows results for the equivalent experiment using leave-one-domain-out cross-validation. Unlike for the "challenge 3" instance set used by Roberts et al. (2008), on this (harder) set of instances the PDDL domain features are no longer sufficient for achieving good prediction accuracy, and neither is the set of all PDDL features. Our features achieve substantially lower RMSE, even in the more challenging leave-one-domain-out scenario. We note further that in all scenarios our features achieve lower RMSE than the combined sets of features from Roberts et al. (2008) and Cenamor et al. (2012; 2013).

Figure 1 compares predicted vs. observed runtime across our entire set of problem instances using the models from Table 3 for the LAMA planner. These plots illustrate the improvement in accuracy gained from our larger feature set over the 32 PDDL features used by Roberts et al. (2008),

Planner	None	DF	+IF	+LF	+NF-0	+Ce	+NF-1	+NF-2	All
	RMSE of $\log_{10}$(runtime)								
Arvand	1.61	1.45	0.92	0.82	0.89	0.79	0.83	**0.65**	**0.61**
Fast Downward	1.55	1.14	0.61	0.58	0.63	0.45	0.46	**0.32**	**0.31**
FF v2.3	2.5	1.7	1.05	1.02	1.04	1.00	0.93	**0.82**	**0.71**
FF-X	2.47	1.75	1.05	1.01	1.03	1.01	**0.92**	**0.79**	**0.69**
LAMA	1.53	1.35	0.86	0.73	0.82	0.66	0.69	**0.54**	**0.49**
LPG	1.89	1.53	0.99	0.83	0.86	0.86	**0.55**	**0.59**	0.62
LPG-contra	1.89	1.52	1.00	0.9	0.89	0.91	**0.57**	**0.61**	**0.62**
Metric-FF v2.1	2.36	1.88	0.99	**0.93**	1.05	1.04	**0.95**	**0.90**	**0.78**
Mp	2.23	1.9	0.99	0.99	1.02	0.97	1.03	0.90	**0.47**
Probe	2.08	1.78	0.98	0.98	0.98	0.96	**0.82**	**0.84**	**0.69**

Table 4: Cross-validated RMSE of $\log_{10}$(runtime) obtained using the same process as for Table 3, but performing leave-one-domain-out cross-validation across all 180 problem domains. Note that means and statistical tests are taken over these 180 folds (i.e., domains with few and many problems have equal weight).



(a) LAMA, PDDL features (b) LAMA, All features

Figure 1: Scatter plots comparing predicted to actual runtime of LAMA on our full set of instances, using 10-fold cross-validation. (a) shows performance for a model using only the 32 PDDL features from Roberts et al. (2008); (b) shows the improved performance using our full set of 311 features.

both for 10-fold and leave-one-domain-out cross-validation. We note that many points in those plots lie close to the main diagonal, indicating accurate predictions: For LAMA and 10-fold cross-validation, 90.5% of our predictions are within a factor of 2 of the observed runtimes with our full feature set, vs. 83.1% when using the PDDL features of Roberts et al. (2008) and 88.1% when using the PDDL features plus those of Cenamor et al. (2012; 2013). For the leave-one-domain-out scenario, the corresponding numbers are 69.8% vs. 54.0% and 62.5%. We observed qualitatively similar results for the other planners.

If we sort the instances in our set by hardness (approximated by the runtime required for LAMA to solve them), for all planners the RMSE for the easier half of the instances is lower than for the harder half. In the case of LAMA and 10-fold cross-validation, the resulting RMSE values are 0.198 and 0.420, respectively.

Experiments: Feature Importance

EPMs are valuable beyond their ability to make performance predictions: an effective model can be examined to

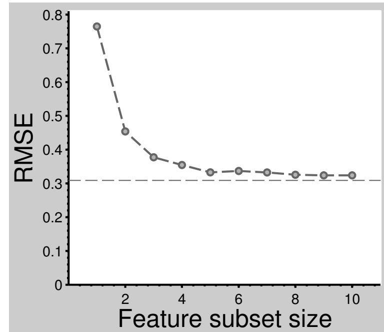


Figure 2: RMSE across number of features in greedy forward selection (LAMA, 10-fold cross-validation).

identify properties of an instance that strongly influence runtime. In practice, such an analysis is complicated by two hurdles. First, many models (such as random forests) are complex, making them difficult for humans to understand. Second, features are often strongly correlated, making it difficult to assess feature importance by looking at individual features' effects on model performance. (For example, a natural way of assessing importance is measuring the amount model performance suffers by omitting each feature; when features are correlated, this method fails badly.) We therefore investigated which features were most informative for runtime prediction by using forward selection (Leyton-Brown, Nudelman, and Shoham 2002; Hutter, Hoos, and Leyton-Brown 2013). This method starts with a 0-feature model and then repeatedly greedily adds the feature that most improves RMSE. This avoids highly correlated features; a small, easy-to-understand model may be obtained by stopping this process before performance stops increasing substantially at each step. We quantify a feature's importance as the increase in RMSE when omitting the feature from that small model. In our experiments, we ran forward selection for 10 steps, averaging feature importance measurements across ten forward selection runs for each planner.

Figure 2 shows the impact of the top 10 features on RMSE for runtime models for LAMA. Observe that the first few features were already sufficient to achieve nearly the same accuracy as the full model; the same pattern also held for our other planners. There is not a unique subset of important features shared by all planners. Features from each of our categories are important for all of the planners we study, but the exact features selected differ across planners: for FD-based planners FDR-based features work well, but FD-probing, LPG-preprocessing and SAT features were selected for many of the planners. Timing and Torchlight features were generally less important. The reasons for this deserve further investigation, but likely include Torchlight's relative lack of robustness.

Tables 3 and 4 assess the computational cost of feature extraction, considering increasingly costly feature sets. Overall, "cheap" features were usually sufficient for outperforming the previous state of the art, and "moderate" features often came close to the performance of our full model. In some cases—in particular Mp, for which the ("expensive") SAT representation features are of course important—adding expensive features significantly improved model performance.

Conclusions

Predicting performance is an important research direction in its own right, and also offers huge potential for practical improvements to solver performance via intelligent per-instance solver selection, as amply demonstrated by the success of SATzilla (Xu et al. 2008) in SAT. Improvements in prediction depend critically on stronger features, which we consider to be a gaping hole in the planning literature. Recent attempts by Cenamor et al. (2012; 2013) made valuable progress by considering causal graph and DTG structures; the use of Torchlight (Hoffmann 2011) permits deeper analysis of these same structures. But no previous work took the (in retrospect obvious) step of designing features based on search probing and SAT encodings. We close that part of the hole in this work, designing a comprehensive feature set, and conducting extensive experiments showing that significant improvements can be obtained in the accuracy of runtime prediction. Our next steps will be to further extend our feature collection, and to engineer a portfolio approach exploiting its predictions. We believe that a comprehensive feature set will be important for many other purposes as well, such as the recent methods for learning macro actions by Alhossaini and Beck (2012).

References

Alhossaini, M., and Beck, J. 2012. Macro learning in planning as parameter configuration. In *Proceedings of the 25th Canadian Conference on Artificial Intelligence*, 13–24.

Brewer, E. A. 1994. *Portable high-performance supercomputing: high-level platform-dependent optimization*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2012. Mining IPC-2011 results. In *Proceedings of the 3rd workshop on the International Planning Competition*.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2013. Learning predictive models to configure planning portfolios. In *Proceedings of the 4th workshop on Planning and Learning (ICAPS-PAL 2013)*, 14–22.

Fawcett, C.; Helmert, M.; Hoos, H. H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Domain-specific configuration using Fast Downward. In *Proceedings of the 3rd workshop on Planning and Learning (ICAPS-PAL 2011)*, 13–20.

Fink, E. 1998. How to solve it automatically: Selection among problem-solving methods. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 128–136.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239–290.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Hoffmann, J. 2011. Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal of Artificial Intelligence Research* 41:155–229.

Howe, A.; Dahlman, E.; Hansen, C.; Von Mayrhauser, A.; and Scheetz, M. 1999. Exploiting competitive planner performance. In *Proceedings of the 5th European Conference on Planning (ECP-99)*, 62–72.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206:79–111.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Identifying key algorithm parameters and instance features using forward selection. In *Proceedings of the 7th International Conference on Learning and Intelligent Optimization (LION 7)*, 364–381.

Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Principles and Practice of Constraint Programming - CP 2002*, 556–572.

Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2009. Empirical hardness models: methodology and a case study on combinatorial auctions. *Journal of the ACM* 56(4):1–52.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 154–161. AAAI press.

Nakhost, H.; Müller, M.; Valenzano, R.; and Xie, F. 2011. Arvand: the art of random walks. In *Booklet of the 7th International Planning Competition*.

Nudelman, E.; Leyton-Brown, K.; Devkar, A.; Shoham, Y.; and Hoos, H. 2004. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Principles and Practice of Constraint Programming - CP 2004*, 438–452.

Rintanen, J. 2012. Engineering efficient planners with SAT. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 684–689.

Roberts, M., and Howe, A. 2009. Learning from planner performance. *Artificial Intelligence* 173(5-6):536–561.

Roberts, M.; Howe, A. E.; Wilson, B.; and desJardins, M. 2008. What makes planners predictable? In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 288–295.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS 2012)*, 369–372.

Smith-Miles, K.; van Hemert, J.; and Lim, . X. Y. 2010. Understanding TSP difficulty by learning from evolved instances. In *Proceedings of the 4th International Conference on Learning and Intelligent Optimization (LION 4)*, 266–280.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1):38–69.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32:565–606.

Xu, L.; Hutter, F.; Shen, J.; Hoos, H.; and Leyton-Brown, K. 2012. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. Solver description, SAT Challenge 2012.