Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms

Frank Hutter[1], Youssef Hamadi[2], Holger H. Hoos[1], and Kevin Leyton-Brown[1]

[1] University of British Columbia, 2366 Main Mall, Vancouver BC, V6T1Z4, Canada
{hutter, kevinlb, hoos}@cs.ubc.ca
[2] Microsoft Research, 7 JJ Thomson Ave, Cambridge, UK
youssefh@microsoft.com

Abstract. Machine learning can be used to build models that predict the run-time of search algorithms for hard combinatorial problems. Such *empirical hardness models* have previously been studied for complete, deterministic search algorithms. In this work, we demonstrate that such models can also make surprisingly accurate predictions of the run-time distributions of incomplete and randomized search methods, such as stochastic local search algorithms. We also show for the first time how information about an algorithm's parameter settings can be incorporated into a model, and how such models can be used to automatically adjust the algorithm's parameters on a per-instance basis in order to optimize its performance. Empirical results for Novelty$^+$ and SAPS on structured and unstructured SAT instances show very good predictive performance and significant speedups of our automatically determined parameter settings when compared to the default and best fixed distribution-specific parameter settings.

1 Introduction

The last decade has seen a dramatic rise in our ability to solve combinatorial optimization problems in many practical applications. High-performance heuristic algorithms increasingly exploit problem instance structure. Thus, knowledge about the relationship between this structure and algorithm behavior forms an important basis for the development and successful application of such algorithms. This has inspired a large amount of research on methods for extracting and acting upon such information. These range from search space analysis to automated algorithm selection and tuning methods.

An increasing number of studies explore the use of machine learning techniques in this context [15,18,6,8]. One recent approach uses linear basis function regression to obtain models of the time an algorithm will require to solve a given problem instance [19,21]. These so-called *empirical hardness models* can be used to obtain insights into the factors responsible for an algorithm's performance, or to induce distributions of problem instances that are challenging for a given algorithm. They can also be leveraged to select among several different algorithms for solving a given problem instance.

In this paper, we extend on this work in three significant ways. First, past work on empirical hardness models has focused exclusively on complete, deterministic algorithms [19,21]. Our first goal is to show that the same methods can be used to predict sufficient statistics of the run-time distributions (RTDs) of incomplete, randomized algorithms, and in particular of stochastic local search (SLS) algorithms for SAT. This is

F. Benhamou (Ed.): CP 2006, LNCS 4204, pp. 213–228, 2006.

important because SLS algorithms are among the best existing techniques for solving a wide range of hard combinatorial problems, including hard subclasses of SAT [14].

The behavior of many randomized heuristic algorithms is controlled by parameters with continuous or large discrete domains. This holds in particular for most state-of-the-art SLS algorithms. For example, the performance of WalkSAT algorithms such as Novelty [20] or Novelty$^+$ [12] depends critically on the setting of a noise parameter whose optimal value is known to depend on the given SAT instance [13]. Understanding the relationship between parameter settings and the run-time behavior of an algorithm is of substantial interest for both scientific and pragmatic reasons, as it can expose weaknesses of a given search algorithm and help to avoid the detrimental impact of poor parameter settings. Thus, our second goal is to extend empirical hardness models to include algorithm parameters in addition to features of the given problem instance.

Finally, hardness models could also be used to automatically determine good parameter settings. Thus, an algorithm's performance could be optimized for each problem instance without any human intervention or significant overhead. Our final goal is to explore the potential of such an approach for automatic per-instance parameter tuning.

In what follows, we show that we have achieved all three of our goals by reporting the results of experiments with SLS algorithms for SAT. (We note however, that our approach is by no means limited to SLS algorithms or SAT, though the features we use were created with some domain knowledge. In experimental work it is obviously necessary to choose *some* specific domain. We have chosen to study the SAT problem because it is the prototypical and best-studied $\mathcal{NP}$-complete problem and there exists a great variety of SAT benchmark instances and solvers.) Specifically, we considered two high-performance SLS algorithms for SAT, Novelty$^+$ [12] and SAPS [17], and several widely-studied structured and unstructured instance distributions. In Section 2, we show how to build models that predict the sufficient statistics of RTDs for randomized algorithms. Empirical results demonstrate that we can predict the median run-time for our test distributions with surprising accuracy (we achieve correlation coefficients between predicted and actual run-time of up to 0.995), and that based on this statistic we can also predict the complete exponential RTDs Novelty$^+$ and SAPS exhibit. Section 3 describes how empirical hardness models can be extended to incorporate algorithm parameters; empirical results still demonstrate good performance for this harder task (correlation coefficients reach up to 0.98). Section 4 shows that these models can be leveraged to perform automatic per-instance parameter tuning that results in significant reductions of the algorithm's run-time compared to using default settings (speedups of up to two orders of magnitude) or even the best fixed parameter values for the given instance distribution (speedups of up to an order of magnitude). Section 5 describes how Bayesian techniques can be leveraged when predicting run-time for test distributions that differ from the one used for training of the empirical hardness model. Finally, Section 6 concludes the paper and points out future work.

2 Run-Time Prediction: Randomized Algorithms

Previous work [19,21] has shown that it is possible to predict the run-time of deterministic tree-search algorithms for combinatorial problems using supervised machine learning techniques. In this section, we demonstrate that similar techniques are able

to predict the run-time of algorithms which are both randomized and incomplete. We support our arguments by presenting the results of experiments involving two powerful local search algorithms for SAT.

2.1 Prediction of Sufficient Statistics for Run-Time Distributions

It has been shown in the literature that high-performance randomized local search algorithms tend to exhibit exponential run-time distributions [14], meaning that the run-times of two runs that differ only in their random seeds can easily vary by an order of magnitude. Even more extreme variability in run-time has been observed for randomized complete search algorithms [11]. Due to this inherent algorithm randomness, we have to predict a probability distribution over the amount of time an algorithm will take to solve the problem. For many randomized algorithms such *run-time distributions* closely resemble standard parametric distributions such as exponential or Weibull (see, e.g., [14]). These parametric distributions are completely specified by certain sufficient statistics. For example, an exponential distribution can be specified by its median. It follows that by predicting such sufficient statistics, a prediction for the entire run-time distribution for an unseen instance is obtained.

Note that for randomized algorithms, the error in a model's predictions can be divided into two components: the extent to which the model fails to describe the data, and the inherent noise in the employed summary statistics due to randomness of the algorithm. This latter component may be reduced by measuring the statistics over a larger number of runs per instance. As we will see in Figures 1(a) and 1(b), while empirical hardness models of SLS algorithms are able to predict the run-times of single runs reasonably well, their predictions of median run-times over a larger set of runs are much more accurate.

Our approach for run-time prediction of randomized incomplete algorithms largely follows the basis function regression approach of [19,21].[1] While an extension of our work to randomized tree search algorithms should be straight-forward, experiments in this paper are restricted to incomplete local search algorithms.

In order to predict the run-time of an algorithm $\mathcal{A}$ on a distribution $\mathcal{D}$ of instances, we draw an i.i.d. sample of N instances from $\mathcal{D}$. For each instance s_n in this training set, $\mathcal{A}$ is run some constant number of times and an empirical fit r_n of the sufficient statistics of interest is recorded. Note that r_n is a $1 \times M$ vector if there are M sufficient statistics of interest. We also compute a set of $k = 43$ instance features $\boldsymbol{x}_n = [x_{n,1}, \ldots, x_{n,k}]$ for each instance. This set is a subset of the features used in [21], including basic statistics, graph-based features, local search probes, and DPLL-based measures.[2] We restricted

[1] In previous preliminary and unpublished experiments for the winner determination problem, we examined other techniques such as support vector machine regression, multivariate adaptive regression splines and lasso regression; none improved predictive performance significantly. More recent experiments (see Section 5) suggest that Gaussian process regression *can* increase performance, especially when the amount of training data is small. However, this method has complexity cubic in the number of *data points*, complicating its practical use.

[2] Information on precisely which features we used, as well as the rest of our experimental data and Matlab code, is available online at `http://www.cs.ubc.ca/labs/beta/Projects/Empirical-Hardness-Models/`

the subset of features because some features from [21] timed out for large instances—the computation of all our 43 features took only about 2 seconds per instance.

Given this data for all the training instances, a function $f(\boldsymbol{x})$ is fitted that, from the features $\boldsymbol{x}_n$ of an instance s_n, approximates the sufficient statistics r_n of $\mathcal{A}$'s run-time distribution on this instance. Since linear functions of these raw features may not be expressive enough, we construct a richer set of basis functions which can include arbitrarily complex functions of *all* features $\boldsymbol{x}_n$ of an instance s_n, or simply the raw features themselves. These basis functions typically contain a number of elements which are either unpredictive or highly correlated. Predictive performance can thus be improved (especially in terms of robustness) by applying some form of feature selection that identifies a small subset of D important features; as explained later, here we use forward selection with a designated validation set to select up to $D = 40$ features. We denote the reduced set of D basis functions for instance s_n as $\boldsymbol{\phi}_n = \boldsymbol{\phi}(\boldsymbol{x}_n) = [\phi_1(\boldsymbol{x}_n), \ldots, \phi_D(\boldsymbol{x}_n)]$. We then use ridge regression to fit the $D \times M$ matrix of free parameters $\boldsymbol{w}$ of a linear function $f_{\boldsymbol{w}}(\boldsymbol{x}_n) = \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{w}$, that is, we compute $\boldsymbol{w} = (\delta I + \boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{r}$, where δ is a small regularization constant (set to 10^{-2} in our experiments), $\boldsymbol{\Phi}$ is the $N \times D$ design matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1^\top, \ldots, \boldsymbol{\phi}_N^\top]^\top$, and $\boldsymbol{r} = [{r_1}^\top, \ldots, {r_N}^\top]^\top$. Given a new, unseen instance s_{N+1}, a prediction of the M sufficient statistics can be obtained by computing the instance features $\boldsymbol{x}_{N+1}$ and evaluating $f_{\boldsymbol{w}}(\boldsymbol{x}_{N+1}) = \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^\top \boldsymbol{w}$. One advantage of the simplicity of ridge regression is a low computational complexity of $\Theta(\max\{D^3, D^2 N, DNM\})$ for training and of $\Theta(DM)$ for prediction for an unseen test instance.

2.2 Experimental Setup and Empirical Results for Predicting Median Run-Time

We performed experiments for the prediction of run-time distributions for two SLS algorithms, SAPS and Novelty$^+$. Because previous studies [12,17,14] have shown that these algorithms tend to have approximately exponential run-time distributions, the sufficient statistics r_n for each instance s_n reduce to the empirical median run-time of a fixed number of runs. In this section we fix SAPS parameters to their default values $\langle \alpha, \rho, P_{smooth} \rangle = \langle 1.3, 0.8, 0.05 \rangle$. For Novelty$^+$, we use its default parameter setting $\langle noise, wp \rangle = \langle 0.5, 0.01 \rangle$ for unstructured instances. On structured instances Novelty$^+$ is known to perform better with lower noise settings, and indeed with noise=0.5 the majority of runs did not finish within an hour of CPU time. Thus, we chose $\langle noise, wp \rangle = \langle 0.1, 0.01 \rangle$ which solved all structured instances in 15 minutes of CPU time. We consider models that incorporate multiple parameter settings in the next section.

In our experiments, we used six widely-studied SAT benchmark distributions, half consisting of unstructured instances and half of structured instances. The first two distributions we studied each consisted of 20,000 uniform-random 3-SAT instances with 400 variables; the first (**CV-var**) varied the clauses-to-variables ratio between 3.26 and 5.26, while the second (**CV-fixed**) fixed $c/v = 4.26$. These distributions were previously studied in [21], facilitating a comparison of our results with past work. Our third unstructured distribution (**SAT04**) consisted of 3,000 random unstructured instances generated with the two generators used for the 2004 SAT solver competition (with identical parameters) and was employed to evaluate our automated parameter tuning procedure on a competition benchmark.

Table 1. Evaluation of learned models on test data. N is the number of instances for which the algorithm's median runtime is ≤ 900 CPU seconds (only those instances are used and split 50:25:25 into training, validation, and test sets). Columns for correlation coefficient and RMSE indicate values using only raw features as basis functions, and then using raw features and their pairwise products. SAPS was always run with its default parameter settings $\langle \alpha, \rho \rangle = \langle 1.3, 0.8 \rangle$. For Novelty$^+$, we used noise=0.5 for unstructured and noise=0.1 for structured instances.

Unstructured instances					
Dataset	N	Algorithm	Runs	Corrcoeff	RMSE
CV-var	9952	SAPS	1	0.903/0.911	0.37/0.35
CV-var	9952	SAPS	10	0.960/0.968	0.23/0.20
CV-var	9952	SAPS	100	0.967/0.977	0.21/0.17
CV-var	9952	SAPS	1000	0.968/0.978	0.20/0.17
CV-var	9952	Novelty$^+$	10	0.947/0.952	0.25/0.23
CV-fixed	10125	SAPS	10	0.765/0.781	0.46/0.44
CV-fixed	10125	Novelty$^+$	10	0.586/0.603	0.61/0.60
SAT04	1457	SAPS	10	0.933/0.938	0.52/0.50
SAT04	1426	Novelty$^+$	10	0.934/0.938	0.58/0.56

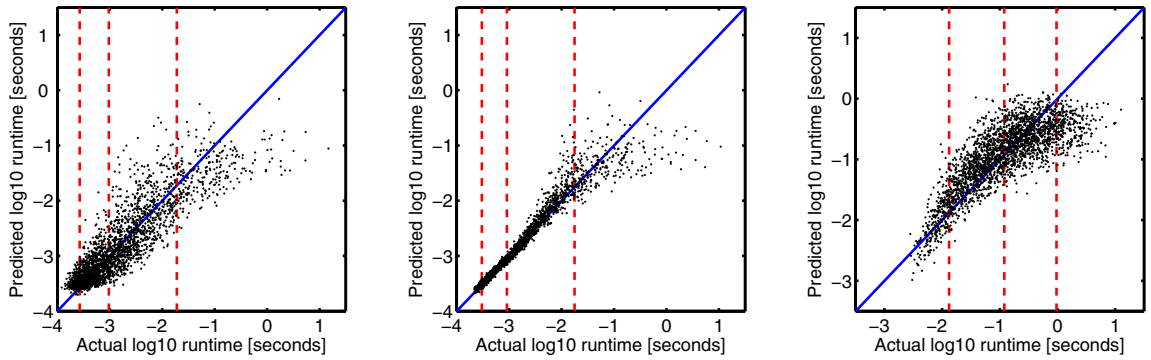
Structured instances					
Dataset	N	Algorithm	Runs	Corrcoeff	RMSE
QWH	7793	SAPS	10	0.988/0.995	0.33/0.21
QWH	8049	Novelty$^+$	10	0.988/0.992	0.22/0.18
QCP	14716	SAPS	10	0.995/0.997	0.17/0.15
QCP	15263	Novelty$^+$	10	0.993/0.994	0.12/0.11
SW-GCP	4287	SAPS	10	0.890/0.892	0.45/0.45
SW-GCP	5573	Novelty$^+$	10	0.690/0.691	0.23/0.23

Our first two structured distributions are different variants of quasigroup completion problems. The first one (**QCP**) consisted of 30,626 quasigroup completion instances, while the second one (**QWH**) contained 9,601 instances of the quasigroup completion problem for quasigroups with randomly punched holes) [10]. Both distributions were created with the generator `lsencode` by Carla Gomes. The ratio of unassigned cells varied from 25% to 75%. We chose quasigroup completion problems as a representative of structured problems because this domain allows the systematic study of a large instance set with a wide spread in hardness, and because the structure of the underlying Latin squares is similar to the one found in applications such as scheduling, time-tabling, experimental design, and error correcting codes [10]. Our last structured distribution (**SW-GCP**) contained 20,000 instances of graph coloring based on small world graphs that were created with the generator `sw.lsp` by Toby Walsh [9].

As is standard in the study of SLS algorithms, all distributions were filtered to contain only satisfiable instances, leading to 9,952, 10,125, 1,470, 17,989, 9,601, and 11,182 instances for CV-var, CV-fixed, SAT04, QCP, QWH, and SW-GCP, respectively. To limit computational time we only used instances that were solved in a single SAPS run of one hour. This further reduced the sets to 9,952, 10,125, 1,469, 15,263, 8,049, and 5,573 instances for CV-var, CV-fixed, SAT04, QCP, QWH, and SW-GCP, respectively.

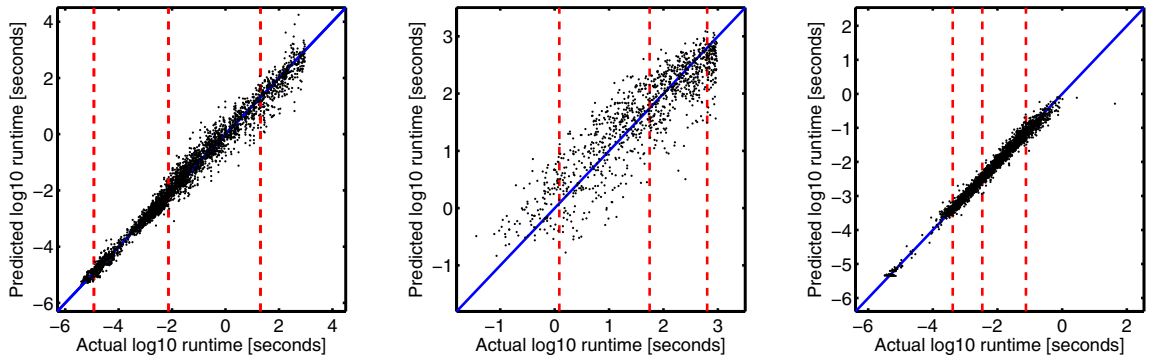
We then randomly split each instance set 50:25:25 into training, validation, and test sets; all experimental results are based on the test set and were stable with respect to reshuffling. We chose the 43 raw features and the constant 1 as our basis functions, and also included pairwise multiplicative combinations of all raw features. We then performed forward selection to select up to 40 features, stopping when the error on the validation set first began to grow. Experiments were run on a cluster of 50 dual 3.2GHz Intel Xeon PCs with 2MB cache and 2GB RAM, running SuSE Linux 9.1.

Overall, our experiments show that we can consistently predict median run-time with surprising accuracy. Results for all our benchmark distributions are summarized in Table 1. Note that a correlation coefficient (CC) of 1 indicates perfect prediction while 0 indicates random noise; a root mean squared error (RMSE) of 0 means perfect prediction



(a) 1 Novelty$^+$ run on CV-var. CC=0.878, RMSE=0.37 (b) 100 Novelty$^+$ runs on CV-var. CC=0.962, RMSE=0.21 (c) 100 SAPS runs on CV-fixed. CC=0.800, RMSE=0.42

Fig. 1. Correlation between observed and predicted run-times/medians of run-times of SAPS and Novelty$^+$ on unstructured instances. The basis functions were raw features and their pairwise products. The three red vertical dashed lines in these and all other scatter plots in this paper denote the 10%, 50%, and 90% quantiles of the data. For example, this means that 40% of the data points lie between the left and the middle vertical lines.



(a) 10 SAPS runs on QWH (b) 10 SAPS runs on SW-GCP (c) 10 Novelty$^+$ runs on QCP

Fig. 2. Correlation between observed and predicted run-times/medians of run-times of SAPS and Novelty$^+$ on SAT04 and QWH. The basis functions were raw features and their pairwise products. For RMSEs and correlations coefficients, see Table 1.

while 1 roughly means average misprediction by one order of magnitude. Also note that the predictive qualities for Novelty$^+$ and SAPS are qualitatively similar.

Figure 1(a) shows a scatterplot of predicted vs. actual run-time for Novelty$^+$ on CV-var, where the model is trained and evaluated on a single run per instance. Most of the data points are located in the very left of this plot, which we visualize by plotting the 10%, 50% and 90% quantiles of the data (the three red dashed lines). While a strong trend is evident in Figure 1(a), there is significant error in the predictions. Figure 1(b) shows the same algorithm on the same dataset, but now predicting the median of an empirical run-time distribution based on 100 runs. The error for the leftmost 90% of the data points is substantially reduced, leading to an almost halved RMSE when compared to predictions for a single run. It is also noteworthy that these run-time predictions are

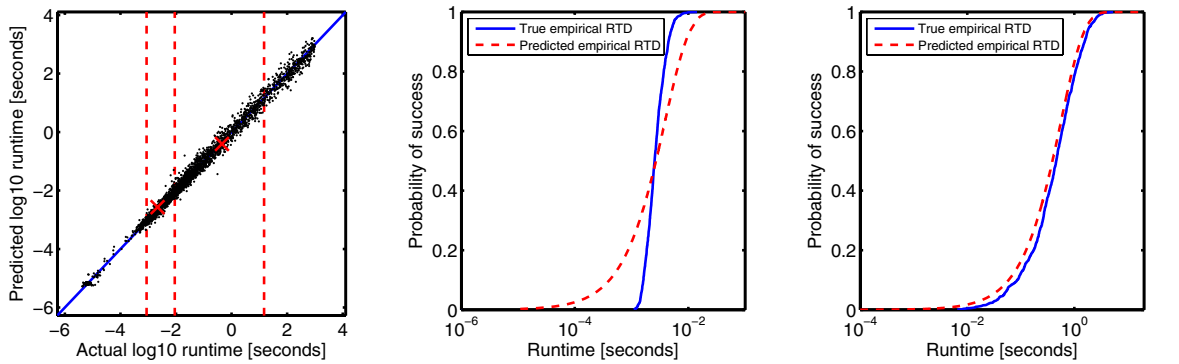
more accurate than the predictions for the deterministic algorithms kcnfs, satz, and oksolver (compare against Figure 5(left) in [21]). While this is already true for predictions based on single runs it is much more pronounced when predicting median run-time. This same effect holds true for predicting median run-time of SAPS, and for different distributions. Figure 1(c) also shows much better predictions than we observed for deterministic tree search algorithms on CV-fix (compare this plot against Figure 7(left) in [21]). We believe that two factors contribute to this effect. First, we see deterministic algorithms as comparable to randomized algorithms with a fixed seed. Obviously, the single run-time of such an algorithm on a particular instance is less informative about its underlying run-time distribution (were it randomized) than the sufficient statistics of multiple runs. Second, one of the main reasons to introduce randomness in search is to achieve diversification. This allows the heuristic to recover from making a bad decision by exploring a new part of the search space, and hence reduces the variance of run-times across very similar instances. Because deterministic solvers do not include such diversification mechanisms, they can exhibit strikingly different run-times on very similar instances. (This observation is the basis of the literature on heavy-tailed runtime distributions in complete search, see e.g. [11].) For example, consider modifying a SAT instance by randomly shuffling the names of its variables. One would expect a properly randomized algorithm to have essentially the same run-time distributions for both instances; however, a deterministic solver could exhibit very different runtimes on the two instances [5]. Because empirical hardness models must give similar predictions for instances with similar feature values, the model for the deterministic solver could be expected to exhibit higher error in this case.

Figure 2 visualizes our predictive quality for structured data sets. Performance for both QWH and QCP, as shown in Figures 2(a) and 2(c), was very good with correlation coefficients between predicted and actual median run-time of up to 0.995. Note, however, that the hardest instance in Figure 2(c) was predicted to be much easier than it actually is. This is because the instance was exceptionally hard: over an order of magnitude harder than the hardest instance in the training set. The last structured data set, SW-GCP, is the hardest distribution for prediction we have encountered thus far (unpublished data shows RMSEs of around 1.0 when predicting the run-time of deterministic algorithms on SW-GCP). As shown in Figure 2(b), the predictions for SAPS are surprisingly good; predictive quality for Novelty$^+$ (see Table 1) is also much higher than what we have seen for deterministic algorithms.

We now look at which features are most important to our models; this is not straightforward since the features are highly correlated. Following [19,21], we build subset models of increasing size until the RMSE and correlation coefficient are comparable to the ones for the full model with 40 basis functions. Table 2 reports the results for SAPS on CV-fix and Novelty$^+$ on QCP and for each of these also gives the performance of the best model with a single basis function. Overall, we observe that the most important features for predicting run-time distributions of our SLS algorithms are the same ones that were observed to be important for predicting run-times of deterministic algorithms in [21]. Also similar to observations from [21], we found that very few features are needed to build run-time models of instances that are all satisfiable. While [21] studied only uniform-random data, we found in our experiments that this is true for both unstructured

Table 2. Feature importance in small subset models for predicting median run-time of 10 runs. The cost of omission for a feature specifies how much worse validation set predictions are without it, normalized to 100 for the top feature. The RMSE and Corrcoeff columns compare predictive quality on the test set to that of full 40-feature models.

#	Basis function	Cost of omission	Corrcoeff	RMSE
SAPS on CV-fix				
1.	saps_BestSolution_CoeffVariance × saps_BestStep_CoeffVariance	100	0.744/0.785	0.47/0.44
1.	saps_BestSolution_CoeffVariance × saps_AvgImproveToBest_Mean	100		
2.	saps_BestStep_CoeffVariance × saps_FirstLMRatio_Mean	45		
3.	gsat_BestSolution_CoeffVariance × lobjois_mean_depth_over_vars	37	0.758/0.785	0.46/0.44
4.	saps_AvgImproveToBest_CoeffVariance	15		
5.	saps_BestCV_Mean × gsat_BestStep_Mean	11		
Novelty$^+$ on QCP				
1.	VG_mean × gsat_BestStep_Mean	100	0.966/0.994	0.29/0.11
1.	saps_AvgImproveToBest_CoeffVariance × gsat_BestSolution_Mean	100		
2.	vars_clauses_ratio × lobjois_mean_depth_over_vars	68		
3.	VG_mean × gsat_BestStep_Mean	12	0.991/0.994	0.13/0.11
4.	TRINARY_PLUS × lobjois_log_num_nodes_over_vars	7		



(a) Predictions of median run-time, instances $q_{0.25}$ and $q_{0.75}$ (b) Easy QCP instance ($q_{0.25}$) (c) Hard QCP instance ($q_{0.75}$)

Fig. 3. Predicted versus actual empirical RTDs for SAPS on two QCP instances. 10 runs were used for learning median run-time and in (a), 1000 runs for the empirical RTDs in (b) and (c).

and structured instances and for both algorithms we studied. Small models for CV-var (both for SAPS and Novelty$^+$) almost exclusively use local search features (almost all of them based on short SAPS trajectories). The structured domain QCP employs a mix of local search probes (based on both SAPS and GSAT), constraint-graph-based features (e.g., VG_mean) and in the case of Novelty$^+$ also some DPLL-based features, such as the estimate of the search tree size (lobjois_mean_depth_over_vars). In some cases (e.g., models of SAPS on CV-var), and when we record relatively few runs per instance, a single feature can be sufficient for predicting single run-times with virtually the same accuracy as the full model.

To illustrate that based on the median we can fairly accurately predict entire run-time distributions for the SLS algorithms studied here, we show the predicted and empirically measured RTDs for SAPS on two QCP instances in Figure 3. The two instances correspond to the 0.25 and 0.75 quantiles of the distribution of actual median hardness for SAPS on the entire QCP instance set; they correspond to the red crosses in Figure 3(a),

Table 3. Parameter configurations employed in our experiments

Algorithm	Fixed parameters	Default parameters	Used parameter configurations
Novelty$^+$	$wp = 0.01$	$noise = 0.5\%$	$noise \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$
SAPS	$P_{smooth} = 0.05$, $wp = 0.01$	$\langle \alpha, \rho \rangle = \langle 1.3, 0.8 \rangle$	All combinations of $\alpha \in \{1.2, 1.3, 1.4\}$ and $\rho \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$

which shows the tight correlation between actual and predicted run-times. Consistent with previous results by Hoos et al. (see, e.g., Chapters 4 and 6 of [14]), the RTD for the $q_{0.75}$ instance is closely approximated by an exponential distribution, which our approach almost perfectly fits (see Figure 3(c)). The RTDs for easier instances are known to typically exhibit smaller variance; therefore, an approximation with an exponential distribution is less accurate (see Figure 3(b)). We plan to predict sufficient statistics for the more general distributions needed to characterize such RTDs, such as Weibull and generalized exponential distributions, in the future.

3 Run-Time Prediction: Parametric Algorithms

The behavior of most high-performance SLS algorithms is controlled by one or more parameters. It is well known that these parameters often have a substantial effect on the algorithm's performance (see, e.g., [14]). In the previous section, we showed that quite accurate empirical hardness models can be constructed when these parameters are held constant. In practice, however, we also want to be able to model an algorithm's behavior when these parameter values are changed. In this section, we demonstrate that it is possible to incorporate parameters into empirical hardness models for randomized, incomplete algorithms. Our techniques should also carry over to both deterministic and complete parametric algorithms (in the case of deterministic algorithms using single run-times instead of sufficient statistics of RTDs).

Our approach is to learn a function $g(\boldsymbol{x}, c)$ that takes as inputs both the features $\boldsymbol{x}_n$ of an instance s_n and the parameter configuration c of an algorithm $\mathcal{A}$, and that approximates sufficient statistics of $\mathcal{A}$'s RTD when run on instance s_n with parameter configuration c. In the training phase, for each training instance s_n we run $\mathcal{A}$ some constant number of times with a set of parameter configurations $\boldsymbol{c}_n = \{c_{n,1}, \ldots, c_{n,k_n}\}$, and collect fits of the sufficient statistics $\boldsymbol{r}_n = [r_{n,1}^\top, \ldots, r_{n,k_n}^\top]^\top$ of the corresponding empirical run-time distributions. We also compute s_n's features $\boldsymbol{x}_n$. The key change from the approach in Section 2.1 is that now the parameters that were used to generate an $\langle$instance,run-time$\rangle$ pair are effectively treated as additional features of that training example. We define a new set of basis functions $\boldsymbol{\phi}(\boldsymbol{x}_n, c_{n,j}) = [\phi_1(\boldsymbol{x}_n, c_{n,j}), \ldots, \phi_D(\boldsymbol{x}_n, c_{n,j})]$ whose domain now consists of the cross product of features and parameter configurations. For each instance s_n and parameter configuration $c_{n,j}$, we will have a row in the design matrix $\boldsymbol{\Phi}$ that contains $\boldsymbol{\phi}(\boldsymbol{x}_n, c_{n,j})^\top$—that is, the design matrix now contains k_n rows for training instance s_n. The target vector $\boldsymbol{r} = [\boldsymbol{r}_1^\top, \ldots, \boldsymbol{r}_N^\top]^\top$ just stacks all the sufficient statistics on top of each other. We learn the function $g_{\boldsymbol{w}}(\boldsymbol{x}, c) = \boldsymbol{\phi}(\boldsymbol{x}, c)^\top \boldsymbol{w}$ by applying ridge regression as in Section 2.1.

Our experiments in this section concentrate on predicting median run-time of SAPS since that is the more challenging problem. SAPS has three interdependent, continuous

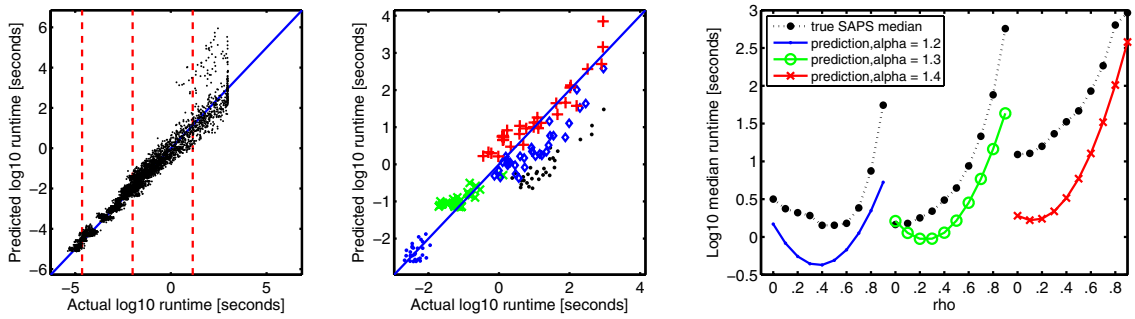


Fig. 4. Left: Predictions for SAPS on QWH with 30 parameter settings. Middle: Data points for 5 instances from SAPS on SAT04, different symbol for each instance. Right: Predicted run-time vs. median SAPS run-time over 1000 runs for 30 parameter settings on the median SAT04 instance, the one marked with blue diamonds in the middle figure.

parameters, as compared to Novelty$^+$ which has only one interesting parameter. (Both algorithms have an additional parameter, wp, which is typically set to a default value that results in uniformly good performance.) This difference notwithstanding, we observed qualitatively similar results with Novelty$^+$. Note that the approach outlined above allows one to use different parameter settings for each training instance. How to pick these settings for training in the most informative way is an interesting experimental design question which invites the use of active learning techniques; we plan to tackle it in future work. In this study, we used the parameter combinations defined in Table 3. We fixed $P_{smooth} = 0.05$ for SAPS since its effect is highly correlated with that of ρ.

As basis functions, we used multiplicative combinations of the raw instance features $\boldsymbol{x}_n$ and a 2nd-order expansion of all non-fixed (continuous) parameter settings. For K raw features ($K = 43$ in our experiments), this meant $3K$ basis functions for Novelty$^+$, and $6K$ for SAPS, respectively. As before we applied forward selection to select up to 40 features, stopping when the error on the validation set first began to grow. For each data set reported here, we randomly picked 1000 instances to be split 50:50 for training and validation. We ran one run per instance and parameter configuration yielding 30,000 data points for SAPS and 6,000 for Novelty$^+$. (Training on the median of more runs would likely have improved the results.) For the test set, we used an additional 100 distinct instances and computed the median of 10 runs for each parameter setting.

In Figure 4(left), we show predicted vs. actual SAPS run-time for the QWH dataset and the 30 $\langle \alpha, \rho \rangle$ combinations in Table 3. This may be compared to Figure 2(a), which shows the same algorithm on the same dataset for fixed parameter values. (Note, however, that Figure 2(a) was trained on more runs and using more powerful basis functions for the instance features.) We observe that our model still achieves good performance, yielding correlation coefficient/RMSE of 0.98/0.41, as compared to 0.988/0.33 for the fixed-parameter setting (using raw features as basis functions).

Figure 4(middle) shows predicted vs. actual SAPS median run-time for five instances from SAT04, namely the easiest and hardest instance, and the 0.25, 0.5, and 0.75 quantiles. Runs corresponding to the same instance are plotted using the same symbol. Note that run-time variation due to the instance is often greater than variation due to parame-

Table 4. Results for automated parameter tuning. For each instance set and algorithm, we give the correlation between actual and predicted run-time for all instances, RMSE, the correlation for all data points of an instance (mean $\pm$ stddev), and the best fixed a posteriori parameter setting on the test set. We also give the average speedup over the best possible parameter setting per instance (s_{bpi}, always ≤ 1), over the worst possible setting per instance (s_{wpi}, always ≥ 1), the default (s_{def}), and the best fixed setting. For example, on Mixed, Novelty$^+$ is on average 9.52 times faster than its best data-set specific fixed parameter setting (s_{fixed}). All experiments use second order expansions of the parameters (combined with the instance features). Bold face indicates speedups of the automated parameter setting over the default and best fixed parameter settings.

Set	Algo	Gross corr	RMSE	Corr per inst.	best fixed a posteriori	s_{bpi}	s_{wpi}	s_{def}	s_{fixed}
SAT04	Nov	0.90	0.76	0.84 ± 0.29	0.5	0.65	193.19	0.88	0.88
QWH	Nov	0.98	0.52	0.76 ± 0.43	0.1	0.85	683.04	**257.96**	0.94
Mixed	Nov	0.95	0.77	0.80 ± 0.35	0.2	0.71	350.12	**14.49**	**9.52**
SAT04	SAPS	0.91	0.60	0.63 ± 0.29	$\langle 1.3, 0 \rangle$	0.43	15.95	**2.66**	0.96
QWH	SAPS	0.98	0.41	0.43 ± 0.39	$\langle 1.2, 0 \rangle$	0.67	5.88	**2.39**	**1.02**
Mixed	SAPS	0.95	0.61	0.47 ± 0.38	$\langle 1.3, 0 \rangle$	0.48	8.53	**2.22**	0.97

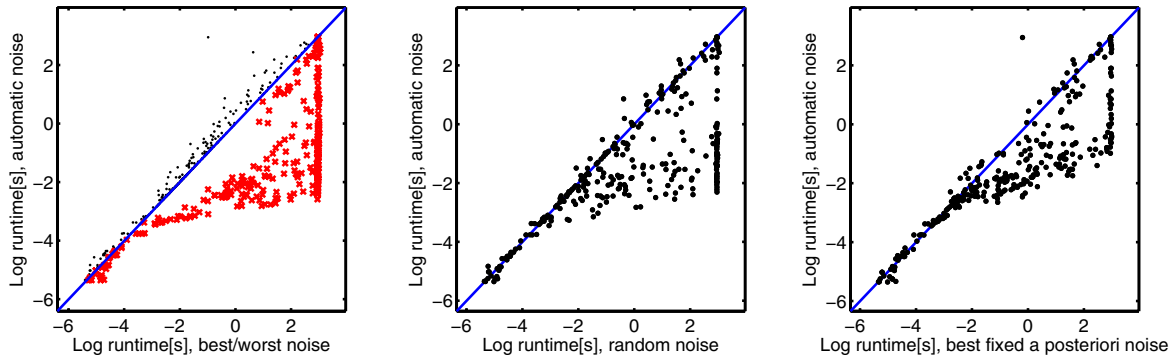
ter settings. However, harder instances tend to be more sensitive to variation in the algorithm's parameters than easier ones – this indicates the importance of parameter tuning, especially for hard instances. The average correlation coefficient for the 30 points per instance is 0.63; for the 6 points per instance in Novelty$^+$ it is 0.84, much higher.

Figure 4(right) shows SAPS run-time predictions for the median instance of our SAT04 test set at each of its 30 $\langle \alpha, \rho \rangle$ combinations; these are compared to the actual median SAPS run-times on this instance. We observe that the learned model predicts the actual run-times fairly well, despite the fact that the relationship between run-time and the two parameters is complex. In the experiment the figure is based on feature selection chose 40 features; thus, the model learned a 40-dimensional surface and the figure shows that its projection onto the 2-dimensional parameter space at the current instance features qualitatively captures the shape of the actual parameter-dependent run-time for this instance.

4 Automated Parameter Tuning

Our results, as suggested by Figure 4 indicate that our methods are able to predict per-instance and per-parameter run-times with reasonable accuracy. We can thus hope that they would also be able to predict which parameter settings result in the lowest run-time for a given instance. This would allow us to use a learned model to automatically tune the parameter values of an SLS algorithm on a per-instance basis by simply picking the parameter configuration out of the ones we consider (see Table 3) that is predicted to yield the lowest run-time. Note that our approach for parameter tuning is orthogonal to that of reactive search approaches such as Adaptive Novelty$^+$ [13] and RSAPS [17].

In this section we investigate this approach. We now focus on the Novelty$^+$ algorithm, because we observed SAPS's performance around $\langle \alpha, \rho \rangle = \langle 1.3, 0.1 \rangle$ to be very



(a) Automatic vs. best & worst (b) Automatic vs. random (c) Automatic vs. best fixed

Fig. 5. (a) Performance of automated parameter setting for Novelty$^+$ on data set Mixed, compared to the best (dots) and worst (crosses) per-instance parameter setting (out of the 6 parameter settings we employed). (b) Compared to independent random noise values for each instance. (c) Speedup of Novelty$^+$ over the best fixed a posteriori parameter setting for Mixed.

close to optimal across many different instance distributions.[3] SAPS thus offers little *possibility* for performance improvement through per-instance parameter tuning (Table 4 quantifies this). Novelty$^+$, on the other hand, exhibits substantial variation in the best parameter setting from one instance distribution to another, making it a good algorithm for the evaluation of our approach.[4] We used the same test and training data as in the previous section; thus, Table 4 summarizes the experiments both from the previous section and from this section. However, in this section we also created a new instance distribution "Mixed", which is the union of the QWH and SAT04 distributions. This mix enables a large gain for automated parameter tuning (when compared to the best fixed parameter setting) since Novelty$^+$ performs best with high noise settings on unstructured instances and low settings on structured instances.

Figure 5(a) shows the performance of our automatic parameter-tuning algorithm on test data from Mixed, as compared to upper and lower bounds on its possible performance. We observe that the run-time with automatic parameter setting is close to the optimal setting and far better than the worst one, with an increasing margin for harder instances. Figure 5(b) provides a comparison of our method against a uniform random picking of parameter combinations from the six considered Novelty$^+$ configurations (see Table 3). Figure 5(c) compares our automatic tuning against the best fixed parameter setting (this was determined in an a posteriori fashion as the setting with the best performance on the test set out of the ones we considered, see Table 3). This setting is often the best that can be hoped for in practice. (A common approach for tuning parameters is to perform a set of experiments, to identify the parameter setting which achieves the lowest overall run-time,

[3] This is true even though it has been demonstrated that for each SAPS parameter there exist instances for which a statistically significant improvement can be obtained over the default setting $\langle \alpha, \rho \rangle = \langle 1.3, 0.8 \rangle$ (defined in [17]) by tuning that parameter [25]. We note that the setting $\langle \alpha, \rho \rangle = \langle 1.3, 0.1 \rangle$ differs from the default studied by [25], raising the question of whether the cited result would also hold for this setting.

[4] Indeed, the large potential gains for tuning WalkSAT's noise parameter on a per-instance basis have been exploited before [22].

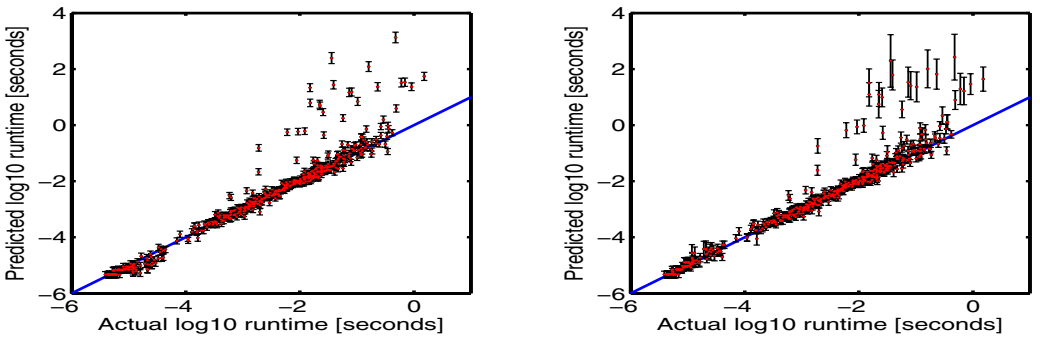
and then to fix the parameters to this setting.) Figure 5(c), in conjunction with Table 4, shows that our techniques can dramatically outperform this form of parameter tuning: Novelty$^+$ almost achieves an average speedup of an order of magnitude on Mixed as compared to the best fixed parameter setting on that set. SAPS improves upon its default setting by more than a factor of two for all three distributions. Considering that our method is fully automatic and very general, these are very promising results.

Related Work on Automated Parameter Tuning

The task of configuring an algorithm's parameters for high and robust performance has been widely recognized as a tedious and time-consuming task that requires well-developed engineering skills. Automating this task is a very promising and active area of research. There exists a large number of approaches to find the best configuration for a given problem distribution [3,24,1]. All these techniques aim to find a parameter setting that optimizes some scoring function which averages over all instances from the given input distribution. If the instances are sufficiently homogeneous, this approach can perform quite well. However, if the problem instances to be solved come from heterogeneous distributions or even from completely unrelated application areas, the best parameter configuration may differ vastly from instance to instance. In such cases it is advisable to apply an approach like ours that can choose the best parameter setting for each run contingent on the characteristics of the current instance to be solved. This per-instance parameter tuning is more powerful but less general than tuning on a per-distribution basis in that it requires the existence of a set of discriminative instance features. However, we believe it to be not too difficult to engineer a good set of instance features if one is familiar with the general problem domain.

The only other approach for parameter tuning on a per-instance basis we are aware of is the Auto-WalkSAT framework [22]. This approach is based on empirical findings showing that the optimal parameter setting of WalkSAT algorithms tends to be about 0.1 above the one that minimizes the invariance ratio [20]. Auto-WalkSAT chooses remarkably good noise settings on a variety of instances, but for domains where the above relationship between invariance ratio and optimal noise setting does not hold (such as logistics problems), it performs poorly [22]. Furthermore, its approach is limited to SAT and in particular to tuning the (single) noise parameter of the WalkSAT framework. In contrast, our automated parameter tuning approach applies to arbitrary parametric algorithms and all domains for which good features can be engineered.

Finally, reactive search algorithms [2], such as Adaptive Novelty$^+$[13] or RSAPS [17] adaptively modify their search strategy *during* a search. (Complete reactive search algorithms include [18,6].) Many reactive approaches still have one or more parameters whose settings remain fixed throughout the search; in these cases the automated configuration techniques we presented here should be applicable to tune these parameters. While a reactive approach is in principle more powerful than ours (it can utilize different search strategies in different parts of the space), it is also less general since the implementation is typically tightly coupled to a specific algorithm. Ultimately, we aim to generalize our approach to allow for modifying parameters during the search—this requires that the features evaluated during search are very cheap to compute. We also see reinforcement learning as very promising in this context [18].



(a) Bayesian linear regression (b) GP with squared exponential kernel

Fig. 6. Predictions and their uncertainty of Novelty$^+$ median run-time of 10 runs: Bayesian linear regression and Gaussian Process with squared exponential kernel, trained on QWH and tested on QCP. The run-time predictions of these approaches are Gaussian probability distributions for every instance. The red dots specify the predictive mean and the black bars the standard deviation.

5 Uncertainty Estimates Through Bayesian Regression

So far, research in empirical hardness models has focused on the case where the targeted application domain is known *a priori* and training instances from this domain are available. In practice, however, an algorithm may have to solve problem instances that are significantly different from the ones encountered during training. Empirical hardness models may perform poorly in such cases. This is because the statistical foundations upon which their machine learning approach is built rely upon the test set being drawn from the same distribution as the training set. Bayesian approaches may be more appropriate in such scenarios since they explicitly model the *uncertainty* associated with their predictions. Roughly, they provide an automatic measure of how similar the basis functions for a particular test instance are to those for the training instances, and associate higher uncertainty with relatively dissimilar instances. We implemented two Bayesian methods: (a) sequential Bayesian linear regression (BLR) [4], a technique which yields mean predictions equivalent to ridge regression but also offers estimates of uncertainty; and (b) Gaussian Process Regression (GPR) [23] with a squared exponential kernel. We detail BLR and the potential applications of a Bayesian approach to run-time prediction in an accompanying technical report [16]. Since GPR scales cubically in the number of data points, we trained it on a subset of 1000 data points (but used all 9601 data points for BLR). Even so, GPR took roughly 1000 times longer to train.

We evaluated both our methods on two different problems. The first problem is to train and validate on our QWH data-set and test on our QCP data-set. While these distributions are not identical, our intuition was that they share enough structure to allow models trained on one to make good predictions on the other. The second problem was much harder: we trained on data-set SAT04 and tested on a very diverse test set containing instances from ten qualitatively different distributions from SATLIB. Figure 6 shows predictions and their uncertainty ($\pm$ one stddev) for both methods on the first problem. The two distributions are similar enough to yield very good predictions for both approaches. While BLR was overconfident on this data set, the uncertainty estimates of GPR make more sense: they are very small for accurately predicted data

points and large for mispredicted ones. Both models achieved similar predictive accuracy (CC/RMSE 0.953/0.50 for BLR; 0.953/0.51 for GPR). For the second problem (space restrictions prevent a figure), BLR showed massive mispredictions (several tens of orders of magnitude) but associated very high uncertainty with the mispredicted instances, reflecting their dissimilarity with the training set. GPR showed more reasonable predictions, and also did a good job in indicating high uncertainty about instances for which predictive quality was low. Based on these preliminary results, we view Gaussian process regression as particularly promising and plan to study its application to run-time prediction in more detail. However, we note that its scaling behavior somewhat limits its usefulness in practice.

6 Conclusion and Future Work

In this work, we have demonstrated that empirical hardness models obtained from linear basis function regression can be extended to make surprisingly accurate predictions of the run-time of randomized, incomplete algorithms such as Novelty$^+$ and SAPS. Based on a prediction of sufficient statistics for run-time distributions (RTDs), we showed very good predictions of the entire empirical RTDs for unseen test instances. We have also demonstrated for the first time that empirical hardness models can model the effect of algorithm parameter settings on run-time, and that these models can be used as a basis for automated per-instance parameter tuning. In our experiments, this tuning never hurt and sometimes resulted in substantial and completely automatic performance improvements, as compared to default or optimized fixed parameter settings.

There are several natural ways in which this work can be extended. First, we are currently studying Bayesian methods for run-time prediction in more detail. Further, it should be straight-forward to apply our approach to randomized systematic search methods and we plan to do this in future work. We also plan to study the extent to which our results generalize to problems other than SAT and in particular to optimization problems. Finally, we would like to apply active learning approaches [7] in order to probe the parameter space in the most informative way in order to reduce training time.

References

1. B. Adenso-Daz and M. Laguna. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1), 2006. To appear.
2. R. Battiti and M. Brunato. Reactive search: machine learning for memory-based heuristics. Technical Report DIT-05-058, Università Degli Studi Di Trento, Dept. of information and communication technology, Trento, Italy, September 2005.
3. M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proc. of GECCO-02*, pages 11–18, 2002.
4. C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
5. F. Brglez, X. Y. Li, and M. F. Stallmann. On SAT instance classes and a method for reliable performance experiments with SAT solvers. *Annals of Mathematics and Artificial Intelligence*, 0:1–34, 2004.
6. T. Carchrae and J. C. Beck. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence*, 21(4):372–387, 2005.

7. D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *JAIR*, 4:129–145, 1996.
8. C. Gebruers, B. Hnich, D. Bridge, and E. Freuder. Using CBR to select solution strategies in constraint programming. In *Proc. of ICCBR-05*, pages 222–236, 2005.
9. I. P. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proc. of AAAI-99*, pages 654–660, Orlando, Florida, 1999.
10. C. P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proc. of AAAI-97*, 1997.
11. C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1), 2000.
12. H. H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. of AAAI-99*, pages 661–666, 1999.
13. H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proc. of AAAI-02*, pages 655–660, 2002.
14. H. H. Hoos and T. Stützle. *Stochastic Local Search - Foundations & Applications*. Morgan Kaufmann, SF, CA, USA, 2004.
15. E. Horvitz, Y. Ruan, C. P. Gomes, H. Kautz, B. Selman, and D. M. Chickering. A Bayesian approach to tackling hard computational problems. In *Proc. of UAI-01*, 2001.
16. F. Hutter and Y. Hamadi. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical Report MSR-TR-2005-125, Microsoft Research, Cambridge, UK, December 2005.
17. F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. of CP-02*, volume 2470, pages 233–248, 2002.
18. M. G. Lagoudakis and M. L. Littman. Learning to select branching rules in the DPLL procedure for satisfiability. In *Electronic Notes in Discrete Mathematics (ENDM)*, 2001.
19. K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proc. of CP-02*, 2002.
20. D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proc. of AAAI-97*, pages 321–326, 1997.
21. E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proc. of CP-04*, 2004.
22. D. J. Patterson and H. Kautz. Auto-WalkSAT: a self-tuning implementation of WalkSAT. In *Electronic Notes in Discrete Mathematics (ENDM), 9*, 2001.
23. C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
24. B. Srivastava and A. Mediratta. Domain-dependent parameter selection of search-based algorithms compatible with user performance criteria. In *Proc. of AAAI-05*, 2005.
25. J. R. Thornton. Clause weighting local search for SAT. *J. of Automated Reasoning*, 2005.