Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms: An Initial Investigation*

Frank Hutter
University of British Columbia
2366 Main Mall
Vancouver, Canada
hutter@cs.ubc.ca

Youssef Hamadi
Microsoft Research
7 JJ Thomson Ave
Cambridge, UK
youssefh@microsoft.com

Holger H. Hoos
University of British Columbia
2366 Main Mall
Vancouver, Canada
hoos@cs.ubc.ca

Kevin Leyton-Brown
University of British Columbia
2366 Main Mall
Vancouver, Canada
kevinlb@cs.ubc.ca

Abstract

Machine learning can be utilized 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 run-time predictions for incomplete, 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 random and structured SAT instances show good predictive performance and significant speedups using our automatically determined parameter settings when compared to the default and best fixed parameter settings.

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 properties of the problem instances to be solved. Thus, knowledge about the relationship between problem 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 (Horvitz *et al.* 2001; Lagoudakis & Littman 2001; Epstein *et al.* 2002; Carchrae & Beck 2005; Gebruers *et al.* 2005). One recent approach uses linear basis function regression to obtain models of the time an algorithm will require to solve a given problem instance (Leyton-Brown, Nudelman, & Shoham 2002; Nudelman *et al.* 2004). 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 present ongoing work based in part on a more comprehensive technical report (Hutter & Hamadi 2005) and utilizing methods from (Leyton-Brown, Nudelman, & Shoham 2002; Nudelman *et al.* 2004). Our current work extends empirical hardness models in three significant ways. First, past work on empirical hardness models has focused exclusively on complete, deterministic algorithms. Our first goal is to show that empirical hardness models can also be constructed for incomplete, randomized algorithms, and in particular for stochastic local search (SLS) algorithms for SAT. This is important because SLS algorithms are among the best existing techniques for solving a wide range of hard combinatorial problems, including hard subclasses of SAT (Hoos & Stützle 2004).

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 (McAllester, Selman, & Kautz 1997) or Novelty$^+$ (Hoos 1999) depends critically on the setting of a noise parameter whose optimal value is known to depend on the given SAT instance (Hoos 2002). 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, reasonably accurate 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. Specifically, we considered two high-

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

*An extended version of this paper is currently under review for CP 2006.

performance SLS algorithms for SAT, Novelty$^+$ (Hoos 1999) and SAPS (Hutter, Tompkins, & Hoos 2002), and several widely-studied random and structured instance distributions. We demonstrate that quite accurate empirical hardness models can be constructed for these SLS algorithms (we achieve correlation coefficients between predicted and actual log run-time of up to 0.99), and that these models can be extended to incorporate algorithm parameters (still yielding correlation coefficients of up to 0.98). We also show 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). For reproducibility, all experimental data and Matlab code this paper is based on will be made available online at `http://www.cs.ubc.ca/labs/beta/Projects/Empirical-Hardness-Models/`.

Run-time Prediction: Randomized Algorithms

Previous work (Leyton-Brown, Nudelman, & Shoham 2002; Nudelman *et al.* 2004) has shown that it is possible to predict the run-time of 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 state-of-the-art local search algorithms for SAT.

High-performance randomized local search algorithms tend to exhibit exponential run-time distributions (Hoos & Stützle 2004), meaning that the run-times of two runs that differ only in their random seeds can easily vary by more than one order of magnitude. Even more extreme variability in run-time has been observed for randomized complete search algorithms (Gomes *et al.* 2000). Due to this inherent randomness of the algorithm (and since we do not incorporate information on a particular run), we have to predict a run-time distribution. Fortunately, exponential distributions can be characterized completely by a single statistic, such as the median (which tends to be statistically more stable than the mean) (Hoos & Stützle 2004). Note that for randomized algorithms, the error in a model's predictions can be thought of as consisting of two components, one of which describes the extent to which the model fails to describe the data, and the second of which expresses the inherent noise in the employed summary statistic (in our case the median). This latter component reduces as the number of runs the median is based on increases. We demonstrate this in Figures 1(a) and 1(b): while empirical hardness models of SLS algorithms are able to predict the run-time of single runs reasonably well, their predictions of median run-time are much more accurate.

Our approach for run-time prediction of randomized incomplete algorithms largely follows the linear regression approach with linear and quadratic basis functions that was already used in (Leyton-Brown, Nudelman, & Shoham 2002). We have previously explored other techniques, namely support vector regression, multivariate adaptive regression splines (MARS), and Lasso Regression, but did not get significantly better run-time predictions when using these methods.[1] While we can handle both complete and incomplete algorithms, we restrict our experiments to incomplete local search algorithms. We note, however, that an extension of our work to randomized tree search algorithms would be straightforward.

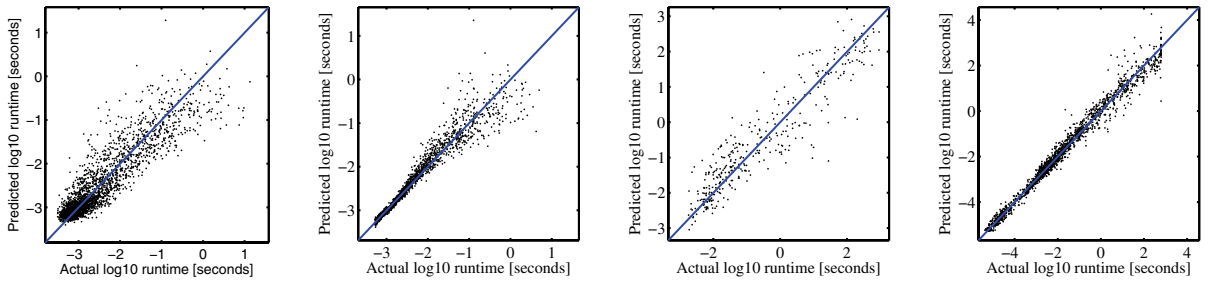
In order to predict the run-time of an algorithm $\mathcal{A}$ on a distribution $\mathcal{D}$ of instances, we draw an i.i.d. sample of N instances from $\mathcal{D}$. For each instance s_n in this training set, $\mathcal{A}$ is run some constant number of times and the median run-time r_n is recorded. 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 (Nudelman *et al.* 2004), including basic statistics, graph-based features, local search probes, and DPLL-based measures. The computation of all features took about 2 seconds for each instance.

Given this data for all the training instances, a function $f(\boldsymbol{x})$ is fitted that, given the features $\boldsymbol{x}_n$ of an instance s_n, approximates $\mathcal{A}$'s *median* run-time r_n. 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 standard ridge regression(see, e.g., (Bishop 1995)) to fit the D free parameters $\boldsymbol{w} = [w_1, \ldots, w_D]^T$ of a linear function $f_{\boldsymbol{w}}(\boldsymbol{x}_n) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)$, that is, we compute $\boldsymbol{w} = (\delta I + \boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{r}$, where δ is a small regularization constant (set to 10^{-2} in our experiments), $\boldsymbol{\Phi}$ is the $N \times D$ design matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1^T, \ldots, \boldsymbol{\phi}_N^T]^T$, and $\boldsymbol{r} = [r_1, \ldots, r_N]^T$. Given a new, unseen instance s_{N+1}, a run-time prediction can be obtained by computing its features $\boldsymbol{x}_{N+1}$ and evaluating $f_{\boldsymbol{w}}(\boldsymbol{x}_{N+1}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{x}_{N+1})$. One advantage of the simplicity of ridge regression is a low computational complexity of $\Theta(max\{D^3, D^2N\})$ for training and of $\Theta(D)$ for prediction for an unseen test instance.

We performed experiments using two SLS algorithms, SAPS and Novelty$^+$. In this section we fix their parameters to their defaults $\langle \alpha, \rho, P_{smooth} \rangle = \langle 1.3, 0.8, 0.05 \rangle$ and $\langle noise, wp \rangle = \langle 50, 0.01 \rangle$, respectively. We consider models that incorporate multiple parameter settings in the next section. We used four widely-studied SAT benchmark distributions. Our first two distributions each consisted of 20,000 uniform-random 3-SAT instances with 400 variables; the first (CV-var) varied the clauses-to-variables ratio be-

[1] Preliminary experiments suggest that Gaussian processes may increase performance, but their cubic scaling behaviour in the number of data points complicates their use in practice.



(a) 1 SAPS run on CV-var (b) 1000 SAPS runs on CV-var (c) 10 Novelty$^+$ runs on SAT04 (d) 10 SAPS runs on QWH

Figure 1: Correlation between observed and predicted run-times/medians of run-times of SAPS on various sets of SAT instances. Raw features and their pairwise products were used as basis functions.

tween 3.26 and 5.26, while the second (CV-fixed) fixed $c/v = 4.26$. Our third distribution (SAT04) consisted of 3,000 random instances generated with the generators used for the 2004 SAT solver competition (with identical parameters), while our fourth (QWH) contained 7,498 instances of the quasigroup completion problem with randomly punched holes (Gomes & Selman 1997). We chose QWH 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 many real-world applications, such as scheduling, time-tabling, experimental design, and error correcting codes (Gomes & Selman 1997). However, we also verified our techniques on other structured instances, such as planning and graph colouring. As is standard in the study of SLS algorithms, all distributions were filtered to contain only satisfiable instances, leading to 10,011, 10,129, 1,420, and 7,498 instances of CV-var, CV-fixed, SAT04, and QWH, respectively.[2]

All instance sets were randomly split 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, where indicated, also included pairwise multiplicative combinations of all raw features. We then performed forward selection [3] 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. All runs were cut off after 900 seconds.

Overall, our experiments showed that we can consistently build surprisingly accurate empirical hardness models. The results of experiments on each of our benchmark distributions are summarized in Table 1. Note that a correlation coefficient of 1 indicates perfect prediction while 0 indicates

Dataset	N	Algorithm	Runs	Corrcoeff	RMSE
CV-var	10011	SAPS	1	0.89/0.90	0.38/0.36
CV-var	10011	SAPS	10	0.95/0.96	0.26/0.22
CV-var	10011	SAPS	100	0.96/0.97	0.23/0.19
CV-var	10011	SAPS	1000	0.96/0.97	0.23/0.18
CV-fixed	10129	SAPS	10	0.73/0.74	0.48/0.47
CV-fixed	10129	Novelty$^+$	10	0.56/0.58	0.58/0.59
SAT04	1420	SAPS	10	0.91/0.94	0.56/0.49
SAT04	1420	Novelty$^+$	10	0.92/0.93	0.63/0.59
QWH	7498	SAPS	10	0.98/0.99	0.38/0.28
QWH	1000	Novelty$^+$	1	0.97/0.99	0.67/0.50

Table 1: Evaluation of learned models on test data. N is the number of instances, 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. For the last entry, we restricted experiments to a random subset of the data and 1 run since Novelty$^+$ was very slow on many instances of QWH.

random noise; an RMSE of 0 means perfect prediction while 1 roughly means misprediction of one order of magnitude on average. Figure 1(a) shows a scatterplot of predicted vs. actual run-time for SAPS on CV-var, where the model is trained to predict the time SAPS takes to execute a single run. While a strong trend is evident, 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 1000 runs of SAPS. The error is substantially reduced; as indicated in Table 1, the RMSE is halved for medians of 1000 runs but 10 runs are almost as good. Figure 1(c) shows predictions for the Novelty$^+$ algorithm on the SAT04 dataset; the higher error in this figure is partly due to the non-homogeneity of the data, partly to the smaller number of runs over which the median is taken, and partly to the smaller amount of training data. Note that our predictions for Novelty$^+$ and SAPS are qualitatively similar on all experiments we conducted (see Table 1). Finally, Figure 1(d) shows the performance of our models for predicting the run-time of SAPS on the QWH

[2] All QWH instances are satisfiable by construction.

[3] Forward selection is a standard method for feature selection that starts with an empty feature set and greedily includes one additional feature at a time, at each step minimizing an error metric (in our case RMSE on the validation set when running ridge regression with the chosen feature set).

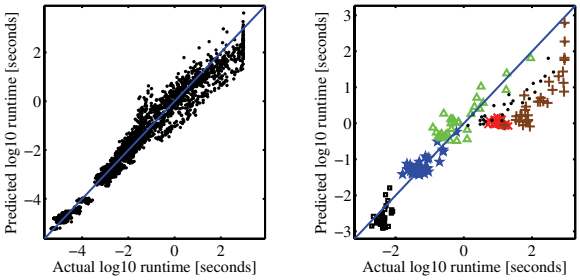
dataset; the nearly-linear plot shows very good performance on this structured dataset. Note, however, that even in this case, there exist instances whose hardness is mispredicted by up to two orders of magnitude. Also note that sometimes predictions tend to become less accurate with increasing hardness (see, e.g., Figures 1(b) and 1(d)). We attribute this latter effect to the sparseness of hard training instances; for example, in Figure 1(b), approximately 90% of the instances can be solved in less than 10^{-2} seconds.

Run-time Prediction: Parametric Algorithms

It is well known that the behaviour of most high-performance SLS algorithms is controlled by one or more parameters, and that these parameters often have a substantial effect on the algorithm's performance (Hoos & Stützle 2004). In the previous section, we showed that good empirical hardness models can be constructed when these parameters are held constant. In practice, however, we want to be able to change these parameter values and to understand what will happen when we do. In this section, we demonstrate that it is possible to incorporate parameters into empirical hardness models for randomized, incomplete algorithms. These results should carry over directly to deterministic and complete parametric algorithms.

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 the run-time of $\mathcal{A}$ with parameter configuration c when run on instance s_n. 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 the medians $\boldsymbol{r}_n = [r_{n,1}, \ldots, r_{n,k_n}]^T$ of the corresponding empirical run-time distributions. We also compute s_n's features $\boldsymbol{x}_n$. The key change from the approach in the last section is that now the parameters that were used to generate an ⟨instance,run-time⟩ 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 $\boldsymbol{\phi}(\boldsymbol{x}_n, c_{n,j})^T$ — that is, the design matrix now contains n_k rows for every training instance. The target vector $\boldsymbol{r} = [\boldsymbol{r}_1^T, \ldots, \boldsymbol{r}_N^T]^T$ just stacks all the median run-times on top of each other. We learn the function $g_{\boldsymbol{w}}(\boldsymbol{x}, c) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}, c)$ by applying ridge regression as in the last section.

Our experiments in this section concentrate on SAPS since it has three interdependent, continuous parameters, as compared to Novelty$^+$ which has only two parameters, one of which (wp) is typically set to a default value that results in uniformly good behaviour. 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 parameter settings for training in the most informative way is an interesting experimental design question with relations to active learning that we plan to tackle in the future. In this study, for SAPS we used all combinations of $\alpha \in \{1.2, 1.3, 1.4\}$ and $\rho \in \{0, .1, .2, .3, .4, .5, .6, .7, .8, .9\}$, keeping $P_{smooth} = 0.05$ constant since its effect is highly correlated with that of ρ. For Novelty$^+$, we used $noise \in \{10, 20, 30, 40, 50, 60\}$, fixing $wp = 0.01$.



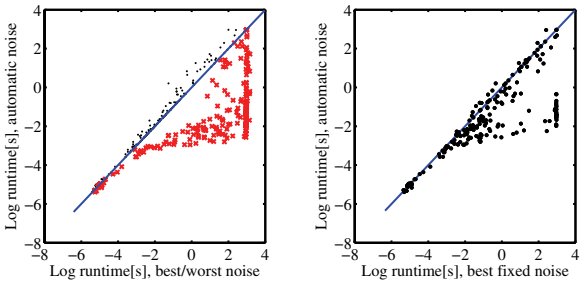
(a) SAPS on QWH, 30 settings (b) 6 symbol-coded instances

Figure 2: (a) Predictions for SAPS on QWH with 30 parameter settings. (b) Data points for 6 instances from SAPS on SAT04, different symbol for each instance.

For the 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 improve 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 2(a), we show predicted vs. actual SAPS runtime for the QWH dataset, where the SAPS parameters are varied as described above. This may be compared to Figure 1(d), which shows the same algorithm on the same dataset for fixed parameter values. (Note, however, that Figure 1(d) was trained on more runs and using more powerful basis functions for the instance features.) We observe that our model still achieves excellent performance, yielding a correlation coefficient of .98 and an RMSE of .40, as compared to .98 and .38 respectively for the fixed-parameter setting (using raw features as basis functions); for Novelty$^+$, the numbers are .98 and .58, respectively.

Figure 2(b) shows predicted vs. actual SAPS median runtime for six instances from SAT04 that cover the whole range of hardness within this dataset. Runs corresponding to the same instance are plotted using the same symbol. Observe that run-time variation due to the instance is often greater than variation due to parameter settings. However, harder instances tend to be more sensitive to variation in the algorithm's parameters than easier ones. The average cor-



(a) Automatic vs. best and worst (b) Automatic vs. best fixed

Figure 3: (a) Performance of automated parameter setting for Novelty$^+$ on mixed data set QWH/SAT04, compared to the best (dots) and worst (crosses) per-instance parameter setting (out of the 6 parameter settings we employed). (b) Speedup of Novelty$^+$ over the best data-set specific fixed parameter setting.

relation coefficient for the 30 points per instance is .52; for the 6 points per instance in Novelty$^+$ it is .86, much higher.

Automated Parameter Tuning

Our results, as suggested by Figures 2(a) and 2(b), indicate that our methods are able to predict per-instance and per-parameter run-times with high accuracy. We can thus hope that they would also be able to accurately 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 that is predicted to yield the lowest run-time.

In this section we investigate this question, focusing on the Novelty$^+$ algorithm. We made this choice because we observed SAPS's performance around $\langle \alpha, \rho \rangle = \langle 1.3, 0.1 \rangle$ to be very close to optimal across many different instance distributions. SAPS thus offers little *possibility* for performance improvement through per-instance parameter tuning (Table 2 quantifies this), and so serves as a poor proving ground for our techniques. Novelty$^+$, on the other hand, exhibits substantial variation from one instance to another and 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 2 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 random instances and low settings on structured instances.

[4]Indeed, the large potential gains for tuning WalkSAT's noise parameter on a per-instance basis have been exploited before (Patterson & Kautz 2001).

Figure 3(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 very close to the optimal configuration per instance and far better than the worst one, with an increasing margin for harder instances. The average performance across all parameter configurations (not shown in the figure) is closer to the worst than to the best setting. Note that this is the performance one could expect on average when sampling parameter settings at random. Figure 3(b) compares our automatic tuning against the best fixed parameter setting for the test set. 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, and then to fix the parameters to this setting.) Figure 3(b), in conjunction with Table 2 shows that our techniques dramatically outperform this form of parameter tuning. While Novelty$^+$ achieves an average speedup of over an order magnitude on Mixed as compared to the best fixed parameter setting on that set, SAPS improves upon its default setting by a factor of 2. Considering that our method is fully automatic and very general, these are very promising results.

Discussion and Related Work

Empirical hardness models assume the test set to be drawn from the same distribution as the training set. For many practical applications, this may pose a severe limitation. We are currently studying a Bayesian approach to empirical hardness models that also quantifies the uncertainty of its predictions. Preliminary experiments promise that this approach can automatically detect how similar a test case is to the training set and determine a confidence measure based on this similarity.

The task of configuring an algorithm's parameters for high and robust performance is 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 (Kohavi & John 1995; Minton 1996; Birattari *et al.* 2002; Srivastava & Mediratta 2005; Adenso-Daz & Laguna 2006). 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 very homogeneous, this approach can perform very 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 relatively

Set	Algo	Gross corr	RMSE	Corr per instance	best fixed params	$s_{bestperinst}$	$s_{worstperinst}$	s_{def}	$s_{bestfixed}$
SAT04	Nov	0.90	0.78	0.86	50	0.62	275.42	0.89	0.89
QWH	Nov	0.98	0.58	0.69	10	0.81	457.09	177.83	0.91
Mixed	Nov	0.95	0.8	0.81	40	0.74	363.08	13.18	10.72
SAT04	SAPS	0.95	0.67	0.52	$\langle 1.3, 0 \rangle$	0.56	10.72	2.00	1.07
QWH	SAPS	0.98	0.40	0.39	$\langle 1.2, .1 \rangle$	0.65	6.03	2.00	0.93
Mixed	SAPS	0.91	0.60	0.65	$\langle 1.2, 0.2 \rangle$	0.46	17.78	1.91	0.93

Table 2: 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 average correlation for all the data points of an instance, and the best fixed parameter setting on the test set. We also give the average speedup over the best possible parameter setting per instance (always < 1), over the worst possible setting per instance (> 1), the default, and the best fixed setting on the test set. For example, on Mixed, Novelty$^+$ is on average 10.72 times faster than its best fixed parameter setting. All experiments use second order expansions of the parameters (combined with the instance features).

straightforward to engineer a good set of instance features if one is familiar with the application domain.

The only other approach for parameter tuning on a per-instance basis we are aware of is the Auto-WalkSAT framework (Patterson & Kautz 2001). This approach is based on empirical findings showing that the optimal parameter setting of WalkSAT algorithms tends to be about 10% above the one that minimizes the invariance ratio (McAllester, Selman, & Kautz 1997). 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 (Patterson & Kautz 2001). 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 (Battiti & Brunato 2005), such as Adaptive Novelty$^+$(Hoos 2002) or RSAPS (Hutter, Tompkins, & Hoos 2002) adaptively modify their search strategy *during* a search. (Complete reactive search algorithms include (Borrett, Tsang, & Walsh 1995; Horvitz *et al.* 2001; Lagoudakis & Littman 2001; Epstein *et al.* 2002; Carchrae & Beck 2005).) 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 reactive approaches have the potential to be strictly more powerful than ours (they can utilize different search strategies in different parts of the space), they are also typically less general since their implementations tend to be tightly coupled to a specific algorithm.

Conclusion and Future Work

In this work, we have demonstrated that empirical hardness models obtained from linear basis function regression can be used to make surprisingly accurate predictions of the run-time of randomized, incomplete algorithms, such as Novelty$^+$ and SAPS. We have also shown for the first time that the same approach 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 working on Bayesian methods that integrate measures of predictive uncertainty in empirical hardness models. We also pursue even more accurate predictions by investigating more powerful classes of machine learning approaches, such as Gaussian processes (Rasmussen & Williams 2006). We further 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 (Cohn, Ghahramani, & Jordan 1996) in order to probe the parameter space in the most informative fashion.

References

Adenso-Daz, B., and Laguna, M. 2006. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research* 54(1). To appear.

Battiti, R., and Brunato, M. 2005. 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.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A racing algorithm for configuring metaheuristics. In *Proc. of GECCO-02*, 11–18.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.

Borrett, J. E.; Tsang, E. P. K.; and Walsh, N. R. 1995. Adaptive constraint satisfaction: The quickest first principle. Technical Report CSM-256, Dept. of Computer Science, University of Essex, Colchester, UK.

Carchrae, T., and Beck, J. C. 2005. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence* 21(4):372–387.

Cohn, D. A.; Ghahramani, Z.; and Jordan, M. I. 1996. Active learning with statistical models. *JAIR* 4:129–145.

Epstein, S. L.; Freuder, E. C.; Wallace, R. J.; Morozov, A.; and

Samuels, B. 2002. The adaptive constraint engine. In *Proc. of CP-02*, 525 – 540.

Gebruers, C.; Hnich, B.; Bridge, D.; and Freuder, E. 2005. Using CBR to select solution strategies in constraint programming. In *Proc. of ICCBR-05)*, 222–236.

Gomes, C. P., and Selman, B. 1997. Problem structure in the presence of perturbations. In *Proc. of AAAI-97*.

Gomes, C.; Selman, B.; Crato, N.; and Kautz, H. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning* 24(1).

Hoos, H. H., and Stützle, T. 2004. *Stochastic Local Search - Foundations & Applications*. Morgan Kaufmann, SF, CA, USA.

Hoos, H. H. 1999. On the run-time behaviour of stochastic local search algorithms for SAT. 661–666.

Hoos, H. H. 2002. An adaptive noise mechanism for WalkSAT. In *Proc. of AAAI-02*, 655–660.

Horvitz, E.; Ruan, Y.; Gomes, C. P.; Kautz, H.; Selman, B.; and Chickering, D. M. 2001. A Bayesian approach to tackling hard computational problems. In *Proc. of UAI-01*.

Hutter, F., and Hamadi, Y. 2005. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical Report MSR-TR-2005-125, Microsoft Research, Cambridge, UK.

Hutter, F.; Tompkins, D. A. D.; and Hoos, H. H. 2002. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. of CP-02*, volume 2470, 233–248.

Kohavi, R., and John, G. H. 1995. Automatic parameter selection by minimizing estimated error. In *Proc. of ICML-95*.

Lagoudakis, M. G., and Littman, M. L. 2001. Learning to select branching rules in the DPLL procedure for satisfiability. In *Electronic Notes in Discrete Mathematics (ENDM)*.

Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proc. of CP-02*.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for invariants in local search. In *Proc. of AAAI-97*, 321–326.

Minton, S. 1996. Automatically configuring constraint satisfaction programs: A case study. *Constraints* 1(1):1–40.

Nudelman, E.; Leyton-Brown, K.; Hoos, H. H.; Devkar, A.; and Shoham, Y. 2004. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proc. of CP-04*.

Patterson, D. J., and Kautz, H. 2001. Auto-WalkSAT: a self-tuning implementation of walksat. In *Electronic Notes in Discrete Mathematics (ENDM), 9*.

Rasmussen, C. E., and Williams, C. K. I. 2006. *Gaussian Processes for Machine Learning*. The MIT Press.

Srivastava, B., and Mediratta, A. 2005. Domain-dependent parameter selection of search-based algorithms compatible with user performance criteria. In *Proc. of AAAI-05*.