Hierarchical Hardness Models for SAT

Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown

University of British Columbia, 2366 Main Mall, Vancouver BC, V6T 1Z4, Canada
{xulin730,hoos,kevinlb}@cs.ubc.ca

Abstract. Empirical hardness models predict a solver's runtime for a given instance of an $\mathcal{NP}$-hard problem based on efficiently computable features. Previous research in the SAT domain has shown that better prediction accuracy and simpler models can be obtained when models are trained separately on satisfiable and unsatisfiable instances. We extend this work by training separate hardness models for each class, predicting the probability that a novel instance belongs to each class, and using these predictions to build a hierarchical hardness model using a mixture-of-experts approach. We describe and analyze classifiers and hardness models for four well-known distributions of SAT instances and nine high-performance solvers. We show that surprisingly accurate classifications can be achieved very efficiently. Our experiments show that hierarchical hardness models achieve higher prediction accuracy than the previous state of the art. Furthermore, the classifier's confidence correlates strongly with prediction error, giving a useful per-instance estimate of prediction error.

1 Introduction

For $\mathcal{NP}$-hard problems such as SAT, even the best known algorithms have worst-case running times that increase exponentially with instance size. In practice, however, many large instances of $\mathcal{NP}$-hard problems can still be solved within a reasonable amount of time. In order to understand this phenomenon, much effort has been invested in understanding the "empirical hardness" of such problems [15,18]. One recent approach uses linear basis-function regression to obtain models that can predict the time required for an algorithm to solve a given SAT instance [18]. These empirical hardness models can be used to gain insight 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 [13,14,20] and can be applied in automated algorithm configuration and tuning [9]. Empirical hardness models have proven very useful for combinatorial auction winner determination [15], a prominent $\mathcal{NP}$-hard optimization problem. In Section 2, we introduce some background knowledge about empirical hardness models as well as our experimental setup.

Considering the SAT problem in particular, previous work has shown that if instances are restricted to be either only satisfiable or only unsatisfiable, very different models are needed to make accurate runtime predictions for uniform random SAT instances. Furthermore, models for each type of instance are simpler and more accurate than models that must handle both types, which means that better empirical hardness models can be built if we know the satisfiability of instances. In this work we further investigate this idea by considering a variety of both structured and unstructured SAT instances and several state-of-the-art SAT solvers. The detailed experimental results are described in Section 3.

In Section 4, we study the feasibility of predicting the satisfiability of a novel SAT instance from a known distribution, using Sparse Multinomial Logistic Regression (SMLR) [11] as our classifier. Our experimental results are very promising. Even for uniform random 3-SAT instances generated at the phase transition, the prediction accuracy was greater than 86%. For the trickiest problems we encountered (SAT-encoded graph coloring problems on small-world graphs), the prediction accuracy was still greater than 73%.

Armed with a reasonably accurate (but imperfect) classifier, in Section 5 we consider the construction of hierarchical hardness models in order to make runtime predictions. We do so by using a mixture-of-experts approach with fixed ("clamped") experts—in other words, with conditional models trained on satisfiable instances and unsatisfiable instances separately. We evaluate both conditional models and then return a weighted sum of the two predictions, where these weights are given by a learned function that depends on both the instance features and the classifier's prediction. We found that using such hierarchical models improved overall prediction accuracy. Furthermore, the classifier's confidence correlated with prediction accuracy, giving useful per-instance evidence about the quality of the runtime prediction.

2 Background

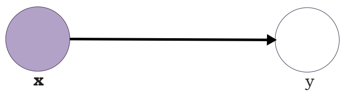
For a given problem instance, empirical hardness models predict the runtime of an algorithm based on polytime-computable instance features. We have investigated a wide variety of different regression techniques in past work [15]. Here, we use the same linear basis-function ridge regression method that has previously proven to be very successful in predicting runtime on uniform random SAT and combinational auctions [18,15] and focus on combining models specialized to different types of instances.

2.1 Empirical Hardness Models

In order to predict the runtime of an algorithm $\mathcal{A}$ on a distribution $\mathcal{I}$ of problem instances, we run algorithm $\mathcal{A}$ on n instances drawn from $\mathcal{I}$ and compute for each instance $i \in \mathcal{I}$ a feature vector $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,k})$. We then fit a function $f(\boldsymbol{x})$ that, given the features $\boldsymbol{x}_i$ of an instance i, approximates $\mathcal{A}$'s runtime on i, y_i. To improve numerical stability by eliminating highly correlated features, we reduce the set of features through forward selection. Then we perform a basis function expansion of our feature set. Our basis functions can include arbitrarily complex functions of sets of features, or can simply be the raw features themselves; in this work, quadratic basis functions are used. Finally, we perform another pass of forward feature selection and select a subset of extended features $\boldsymbol{\phi}_i = \boldsymbol{\phi}(\boldsymbol{x}_i) = [\phi_1(\boldsymbol{x}_i), \ldots, \phi_d(\boldsymbol{x}_i)]$ for which our models achieve the best performance on a given validation data set.

We then use ridge regression to fit the free parameters $\boldsymbol{w}$ of the linear function $f_{\boldsymbol{w}}(\boldsymbol{x}_i) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_i)$. We compute $\boldsymbol{w} = (\delta I + \boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \tilde{\boldsymbol{y}}$, where $\tilde{y} = (\tilde{y_1}, \ldots, \tilde{y_n})$, and $\tilde{y_i}$ is a transformation of the runtime y_i. In this work, we use $\tilde{y_i} = \log y_i$. The $n \times d$ matrix $\boldsymbol{\Phi}$ contains the feature values for all training instances, and δ is a small regularization constant that prevents arbitrarily large parameter values in $\boldsymbol{w}$ and increases numerical stability. Given a new, unseen instance j, a runtime prediction is made by computing its features $\boldsymbol{x}_j$ and evaluating $f_{\boldsymbol{w}}(\boldsymbol{x}_j) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_j)$.

Empirical hardness models have a probabilistic interpretation. The features $\boldsymbol{x}$ and the empirical algorithm runtime y, when seen as random variables, are related as in the following graphical model:



In this model, we observe the feature vector $\boldsymbol{x}$, and the probability distribution over runtime y is conditionally dependent on $\boldsymbol{x}$. Since we train a linear model using least squares fitting, we have implicitly chosen to represent $P(y|\boldsymbol{x})$ as a Gaussian with mean $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$ and some fixed variance β. What we call a prediction of an empirical hardness model is really $\mathbb{E}(y|\boldsymbol{x})$, the mean of this distribution conditioned on the observed feature vector.

2.2 Experimental Setup

For the experiments conducted throughout this study, we selected two distributions of unstructured SAT instances and two of structured instances:

– `rand3-fix`: uniform-random 3-SAT with 400 variables from the solubility phase transition (clauses-to-variables ratio 4.26) [2,19]; we generated 20,000 instances with a satisfiable/unsatisfiable ratio of 50.7/49.3.
– `rand3-var`: uniform-random 3-SAT with 400 variables and clauses-to-variables ratios randomly selected from 3.26 to 5.26; we generated 20,000 instances with a satisfiable/unsatisfiable ratio of 50/50.
– `QCP`: random quasi-group completion (the task of determining whether the missing entries of a partial Latin square can be filled in to obtain a complete Latin square [7]); using a range of parameter settings, we generated 30,620 SAT-encoded instances with a satisfiable/unsatisfiable ratio of 58.7/41.3.
– `SW-GCP`: graph-coloring on small-world graphs [6]; using a range of parameter settings, we generated 20,000 SAT-encoded instances with a satisfiable/unsatisfiable ratio of 55.9/44.1.

The latter two types of SAT distributions have been widely used as a model of hard SAT instances with interesting structure; we used the same instance generators and SAT encodings as the respective original studies. Each instance set was randomly split into training, validation and test sets, at a ratio of 70:15:15. All parameter tuning was performed with a validation set; test sets were used only to generate the final results reported in this paper. Note that since the test sets we used for our experiments are very big (at least 3000 instances each), we can expect similar performance for the whole distribution. For each instance, we computed the 84 features described by Nudelman et al. [18]; these features can be classified into nine categories: problem size, variable-clause graph, variable graph, clause graph, balance features, proximity to Horn formulae, LP-based, DPLL search space, and local search space. We used only raw features as basis functions for classification, because even a simple quadratic basis function expansion exceeded the 2GB of RAM available to us. For regression, we used raw features as well as quadratic basis functions for better runtime prediction accuracy. We evaluated the accuracy of logarithm runtime prediction using root mean squared error (RMSE). In order to reduce the number of redundant features, we used forward selection and kept the model with the smallest cross-validation error (this was done independently for each of the learned hardness models).

For uniform random 3-SAT instances, we ran four solvers that are known to perform well on these distributions: `kcnfs` [3], `oksolver` [12], `march_dl` [8], and `satz` [16]. For structured SAT instances, we ran six solvers that are known to perform well on these distributions: `oksolver`, `zchaff` [22], `sato` [21], `satelite` [4], `minisat` [5], and `satzoo` [5]. Note that in the 2005 SAT competition, `satelite` won gold medals for the Industrial and Handmade SAT+UNSAT categories; `minisat` and `zchaff` won silver and bronze, respectively, for Industrial SAT+UN SAT; and `kcnfs` and `march_dl` won gold and silver, respectively, in the Random SAT+UNSAT category.

All of our experiments were performed using a cluster consisting of 50 computers equipped with dual Intel Xeon 3.2GHz CPUs with 2MB cache and 2GB RAM, running Suse Linux 9.1. All runs of any solver that exceeded 1 CPU hour were terminated and recorded in our database of experimental results with a runtime of 1 CPU hour; this timeout occurred in fewer than 3% of all runs.

3 Conditional and Oracular Empirical Hardness Models

From previous research [18], we know that for uniform-random 3-SAT instances, much simpler and more accurate empirical hardness models can be learned when all instances are either satisfiable or unsatisfiable. In the following, we refer to these as *conditional* models, and to models trained on satisfiable and unsatisfiable instances as *unconditional* models. Let M_{sat} (M_{unsat}) denote a model trained only on satisfiable (unsatisfiable) instances. If we had an oracle that knew which conditional model performed better for a given instance, models equipped with such an oracle could achieve more accurate runtime predictions. We call such a (hypothetical) scheme an *oracular* model. (Note that our oracle chooses the *best* model for a particular instance, not the model trained on data with the same *satisfiability status* as the instance. This may seem counterintuitive; it will be discussed in detail below. For now, note simply that in most cases the two sorts of oracles would select the same model.) We can infer from the results of Nudelman et al. [18] that on uniform-random 3-SAT, oracular models could achieve much higher prediction accuracies than unconditional models. In fact, the performance of an oracular model bounds the performance of a conditional model from above.

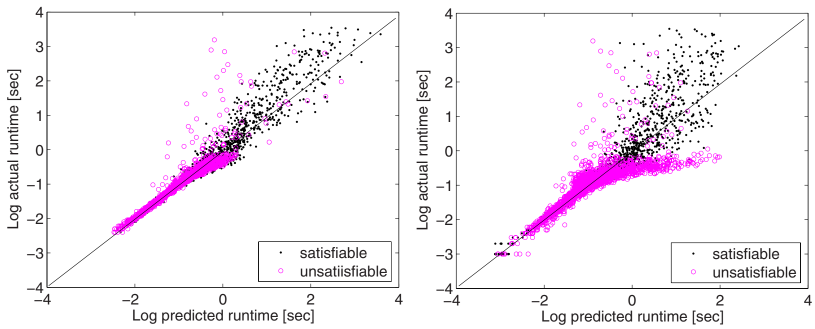


Fig. 1. Comparison of oracular model (left, RMSE=0.247) and unconditional model (right, RMSE=0.426). Distribution: `QCP`, solver: `satelite`.

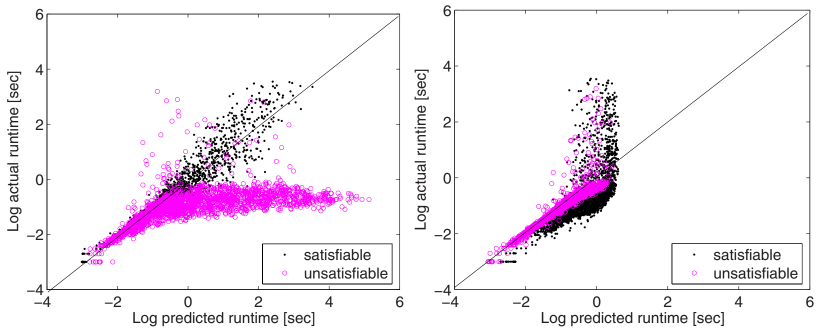


Fig. 2. Actual vs predicted runtime using only M_{sat} (left, RMSE=1.493) and only M_{unsat} (right, RMSE=0.683), respectively. Distribution: `QCP`, solver: `satelite`.

In the experiments conducted for this work, we found that the usefulness of oracular models extends to solvers and distributions not studied previously. Figure 1 shows the difference between using oracular models and unconditional models on structured SAT instances (distribution: `QCP`, solver: `satelite`). For oracular models, we observed almost perfect predictions of runtime for unsatisfiable instances and more noisy, but unbiased predictions for satisfiable instances (Figure 1, left). Figure 1 (right) shows that the runtime prediction for unsatisfiable instances made by unconditional models can exhibit both less accuracy and more bias.

Even though using the best conditional model can result in higher prediction accuracy, we found that there is a big penalty for using the wrong conditional model to predict the runtime of an instance. Figure 2 (left) shows that if we used M_{sat} for runtime prediction on an unsatisfiable instance, the prediction error was often very large. The large bias in the inaccurate predictions is due to the fact that models trained on different types of instances are very different. As shown in Figure 2 (right), similar phenomena occur when we use M_{unsat} to predict the runtime on a satisfiable instance.

Table 1. Accuracy of hardness models for different solvers and instance distributions

	RMSE for `rand3-var` models				**RMSE for `rand3-fix` models**			
Solvers	sat.	unsat.	unconditional	oracular	sat.	unsat.	unconditional	oracular
`satz`	5.481	3.703	0.385	0.329	0.459	0.835	0.420	0.343
`march_dl`	1.947	3.705	0.396	0.283	0.604	1.097	0.542	0.444
`kcnfs`	4.766	4.765	0.373	0.294	0.550	0.983	0.491	0.397
`oksolver`	8.169	4.141	0.443	0.356	0.689	1.161	0.596	0.497
	RMSE for `QCP` models				**RMSE for `SW-GCP` models**			
Solvers	sat.	unsat.	unconditional	oracular	sat.	unsat.	unconditional	oracular
`zchaff`	1.866	1.163	0.675	0.303	1.230	1.209	0.993	0.657
`minisat`	1.761	1.150	0.574	0.305	1.280	1.275	1.022	0.682
`satzoo`	1.293	0.876	0.397	0.240	0.709	0.796	0.581	0.384
`satelite`	1.493	0.683	0.426	0.247	1.232	1.226	0.970	0.618
`sato`	2.373	14.914	0.711	0.375	1.682	1.887	1.353	0.723
`oksolver`	1.213	1.062	0.548	0.427	1.807	2.064	1.227	0.601

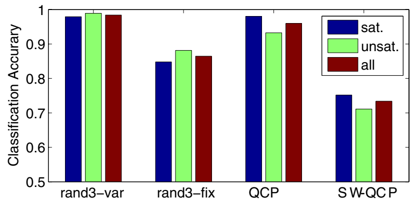
Our results are consistent across data sets and solvers; as shown in Table 1, oracular models always achieved higher accuracy than unconditional models. The very large prediction errors in Table 1 for M_{sat} and M_{unsat} indicate that these models are very different. In particular, the RMSE for using models trained on unsatisfiable instances to predict runtimes on a mixture of instances was as high as 14.914 (distribution: `QCP`, solver: `sato`).

Unfortunately, oracular models rely on information that is unavailable in practice: the respective accuracies of our two models on a given (test) instance. Still, the prediction accuracies achieved by oracular models suggest that it may be promising to find some practical way of combining conditional models. For the rest of this paper, we will investigate the question of how this can be done. We will be guided both by the potential benefit of relying on conditional models (oracular models outperform unconditional models) and by the danger in doing so (if we make the wrong choices, prediction error can be much higher than when using an unconditional model).

4 Predicting the Satisfiability of SAT Instances

In this section, we will consider the most obvious candidate for a practical approximation of the oracle from the previous section: a classifier that predicts whether or not a given (test) instance is satisfiable. Even if this classifier were perfect—which it surely could not be in general, given the $\mathcal{NP}$-completeness of SAT—it would not choose the *better* of M_{sat} and M_{unsat} on a per-instance basis, as our oracular model does. However, on our datasets it would do nearly as well, making the same choices as the oracular model 98% of the time for `rand3-var`, 86% for `rand3sat-fixed`, 92% for `QCP`, and 77% for `SW-GCP`.

To build such models, we used Sparse Multinomial Logistic Regression (SMLR) [11], a recently developed, state-of-the-art sparse classification algorithm. Like relevance vector machines and sparse probit regression, SMLR learns classifiers



Dataset	Classification Accuracy on sat.	on unsat.	overall
rand3sat-var	0.9791	0.9891	0.9840
rand3sat-fix	0.8480	0.8814	0.8647
QCP	0.9801	0.9324	0.9597
SW-GCP	0.7516	0.7110	0.7340

Fig. 3. Classification accuracy for different data sets

that use sparsity-promoting priors to control the expressivity of the learned classifier, thereby tending to result in better generalization. SMLR encourages parameter weights either to be significantly large or exactly zero. It also learns a sparse multi-class classifier that scales favorably in both the number of training samples and the input dimensionality, which is important for our problems since we have tens of thousands of samples per data set. We also evaluated other classifiers, such as support vector machines [10], but we found that SMLR achieved the best classification accuracy.

We applied SMLR to build a classifier that would distinguish between satisfiable and unsatisfiable SAT instances, using the same set of raw features that were available to the regression model, although in this case we did not find it necessary to use a basis-function expansion of these features. The difference was in the response variable: here we defined it as the probability that an instance is satisfiable, rather than an algorithm's runtime on that instance. Of course, although the output of the classifier is real-valued, all the training data was labelled as satisfiable with probability 1 or with probability 0.

Since SAT is $\mathcal{NP}$-hard and our feature computation and model evaluation are polynomial-time, we cannot expect perfect classification results in general. However, $\mathcal{NP}$-hardness is a worst-case notion; there is no theoretical result that rules out *often* correctly guessing whether instances from known distributions are satisfiable. Complexity theory simply tells us that our classifier must sometimes make mistakes (unless $\mathcal{P} = \mathcal{NP}$); as we show below, it does. Indeed, $\mathcal{NP}$-hardness does not imply that all—or even most—instances of a problem are indeed intractable. This is precisely why these problems can be successfully tackled with heuristic algorithms, such as those studied here.

Considering the difficulty of the classification task, our experimental results are very good. Overall accuracy on test data (measured as the fraction of the time the classifier assigned more than 50% of probability mass to the correct class) was as high as 98%, and never lower than 73%. Furthermore, the classifier was usually very confident about the satisfiability of an instance (i.e., returned probabilities very close to 0 or 1), and the more confident the classifier was, the more accurate it tended to be. These results are summarized in Figures 3–5.

For the `rand3-var` data set (Figure 4, left), the overall classification error was only 1.6%. Using only the clauses-to-variables ratio (greater or less than 4.26) as the basis for predicting the satisfiability of an instance yields an error of 3.7%; therefore, by using SMLR rather than this simple classifier, the classification error is halved. On the `QCP` data set (Figure 4, right), classification accuracy was 96%, and the classifier was extremely confident in its predictions.

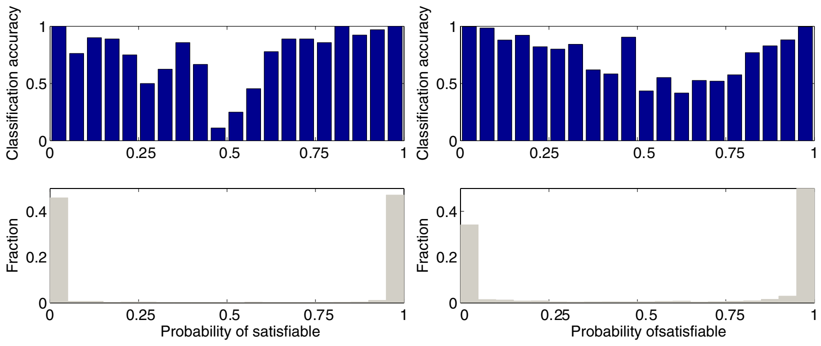


Fig. 4. Classification accuracy vs classifier output (top) and fraction of instances within the given set vs classifier output (bottom). Left: `rand3-var`, right: `QCP`.

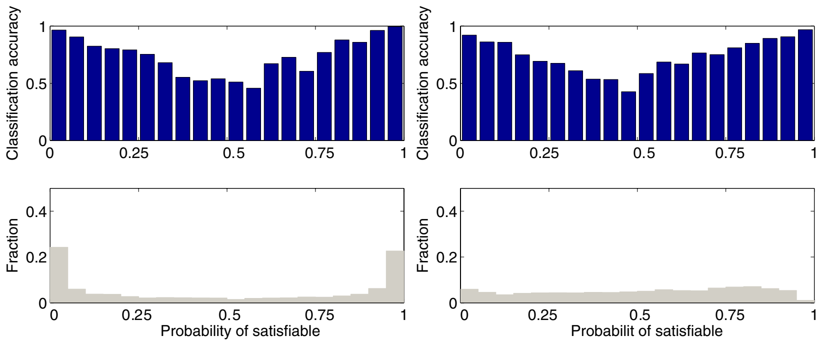


Fig. 5. Classification accuracy vs classifier output (top) and the fraction of instances within the given set vs classified output (bottom). Left: `rand3-fix`, right: `SW-GCP`.

Since all instances for `rand3sat-fix` (Figure 5, left) are generated at the phase transition, it is surprising to see a polynomial-time technique perform so well (accuracy of 86%). For `SW-GCP` (Figure 5, right) the classification accuracy is much lower (73%). We believe that this is primarily because our features are less predictive on this instance distribution, which is consistent with the results we obtained from unconditional hardness models for `SW-GCP`. Note that the fraction of instances for which the classifier was confident is smaller for the last two distributions than for `rand3-var` and `QCP`. However, even for `SW-GCP` we still see a strong correlation between the classifier's output and classification accuracy on test data.

One further interesting finding is that our classifiers can achieve very high accuracies even given very small sets of features. For example, on the `QCP` data, the SMLR classifier achieves an accuracy of 93% with only 5 features. The five most important features for classification on all four data sets are shown in Table 2. Interestingly, local-search based features turned out to be very important for classification in all four data sets.

Table 2. The five most important features (listed from most to least important) for classification as chosen by backward selection. (For details on the features, see [18].)

Data sets	`rand3-var`	`rand3-fix`
Five features	gsat_BestCV_Mean saps_BestStep_CoeffVariance lobjois_mean_depth_over_vars VCG_VAR_max saps_BestSolution_Mean	saps_BestSolution_CoeffVariance gsat_BestSolution_Mean saps_BestCV_Mean lobjois_mean_depth_over_vars gsat_BestCV_Mean
Accuracy (5 features)	98.4%	86.5%
Accuracy (all features)	98.4%	86.5%
Data sets	`QCP`	`SW-GCP`
Five features	lobjois_log_num_nodes_over_vars saps_BestSolution_Mean saps_BestCV_Mean vars_clauses_ratio saps_BestStep_CoeffVariance	vars_reduced_depth gsat_BestCV_Mean nvars VCG_VAR_min saps_BestStep_Mean
Accuracy (5 features)	93.0%	73.2%
Accuracy (all features)	96.0%	73.4%

To the best of our knowledge, our work represents the first attempt to predict the satisfiability of SAT instances using machine learning techniques. Overall, our experiments show that a classifier may be used to make surprisingly accurate polynomial-time predictions about the satisfiability of SAT instances. As discussed above, such a classifier cannot be completely reliable (unless $\mathcal{P} = \mathcal{NP}$). Nevertheless, our classifiers perform very well for the widely-studied instance distributions considered here. This finding may be useful in its own right. For example, researchers interested in evaluating incomplete SAT algorithms on large numbers of satisfiable instances drawn from a distribution that produces both satisfiable and unsatisfiable instances could use a complete search algorithm to label a relatively small training set, and then use the classifier to filter instances.

5 Hierarchical Hardness Models

Given our findings to far, it would be tempting to construct a hierarchical model that uses a classifier to pick the most likely conditional model and then simply returns that model's prediction. However, while this approach could sometimes be a good heuristic, it is not theoretically sound. Intuitively, the problem is that the classifier does not take into account the accuracies of the different conditional models. For example, recall Figure 2, which showed the prediction accuracy of M_{sat} and M_{unsat} for `satelite` on `QCP`. We saw that M_{sat} was much less accurate for unsatisfiable instances than M_{unsat} was for satisfiable instances. Thus if we encountered an instance that the classifier considered slightly more likely satisfiable than unsatisfiable, we would still expect to obtain a more accurate prediction from M_{unsat} than from M_{sat}.

A more principled way of combining conditional models can be derived based on the probabilistic interpretation of empirical hardness models introduced in

Section 2.1. As before (see Figure 6, left) we have a set of features that are always observed and a random variable representing runtime that is conditionally dependent on the features. Now we combine the features with our classifier's prediction s, yielding the feature vector $(\boldsymbol{x}, s)$. We also introduce a new random variable $z \in \{sat, unsat\}$, which represents the oracle's choice of which conditional model will perform best for a given instance. Instead of selecting one of the predictions from the two conditional models for runtime y, we use their weighted sum

$$P(y|(\boldsymbol{x}, s)) = \sum_{z \in \{sat, unsat\}} P(z|(\boldsymbol{x}, s)) \cdot P_{M_z}(y|(\boldsymbol{x}, s)), \tag{1}$$

where $P_{M_z}(y|(\boldsymbol{x}, s))$ is the probability of y evaluated according to model M_z. Since the models were fit using ridge regression, we can rewrite Eq. (1) as

$$P(y|(\boldsymbol{x}, s)) = \sum_{z \in \{sat, unsat\}} P(z|(\boldsymbol{x}, s)) \cdot \mathcal{N}(y|\boldsymbol{w}_z \boldsymbol{x}, \beta_z), \tag{2}$$

where $\boldsymbol{w}_z$ and β_z are the weights and standard deviation of model M_z. Thus, we will learn weighting functions $P(z|(\boldsymbol{x}, s))$ to maximize the likelihood of our training data according to $P(y|(\boldsymbol{x}, s))$. As a hypothesis space for these weighting functions we chose the commonly used softmax function

$$P(z = sat|(\boldsymbol{x}, s)) = \frac{e^{\boldsymbol{v}^\top (\boldsymbol{x}, s)}}{1 + e^{\boldsymbol{v}^\top (\boldsymbol{x}, s)}}, \tag{3}$$

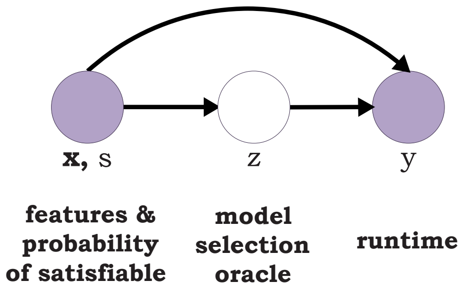
where $\boldsymbol{v}$ is a vector of free parameters that must be learned [1]. Then we have the following loss function to minimize, where $\mathbb{E}(y_{i,z}|\boldsymbol{x})$ is the prediction of M_z and $\bar{y_i}$ is the real runtime:

$$\mathcal{L} = \sum_{i=1}^{N} \left(\bar{y_i} - \left(\sum_{k \in \{sat, unsat\}} P(z = k|(\boldsymbol{x_i}, s_i)) \cdot \mathbb{E}(y_{i,z}|\boldsymbol{x_i}) \right) \right)^2. \tag{4}$$

This can be seen as a mixture-of-experts problem with the experts clamped to M_{sat} and M_{unsat} (see, e.g., [1]). For implementation convenience, we used an existing mixture of experts implementation, which is built around an EM algorithm and which performs iterative reweighted least squares in the M step [17]. We modified this code slightly to clamp the experts and to set the initial values of $P(z|(\boldsymbol{x}, s))$ to s (i.e., we initialized the choice of experts to the classifier's output). To evaluate the model and get a runtime prediction for test data, we simply compute the features $\boldsymbol{x}$ and the classifier's output s, and then evaluate

$$\mathbb{E}(y|(\boldsymbol{x}, s)) = \sum_{k \in \{sat, unsat\}} P(z|(\boldsymbol{x}, s)) \cdot \boldsymbol{w}_k^\top \phi(\boldsymbol{x}), \tag{5}$$

where $\boldsymbol{w}_k$ are the weights from M_k and $\phi(\boldsymbol{x})$ is the basis function expansion of $\boldsymbol{x}$. Thus, the classifier's output is not used directly to select a model, but rather as a feature upon which our weighting functions $P(z|(\boldsymbol{x}, s))$ depend, as well as for initializing the EM algorithm.



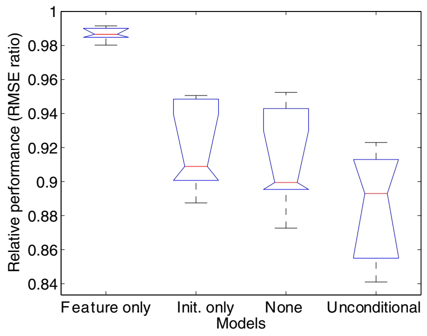


Fig. 6. Left: Graphical model for our mixture-of-experts approach. Right: Comparison of hierarchical hardness models' relative performance: RMSE of full hierarchical model ÷ RMSE of indicated model. Data set: QCP, Solvers: 6 solvers for QCP.

5.1 Experimental Results

Our first experiment used the QCP dataset to investigate the importance of the classifier's output to hierarchical models. Figure 6 (right) shows box plots comparing five scenarios. The first four are hierarchical models with classifier output used (1) both for EM initialization and as a feature to the weighting function; (2) only as a feature; (3) only for initialization; and (4) not at all. We also consider the case of an unconditional model. For each scenario except for the first we report the ratio between its RMSE and that of the first model. The best performance was achieved by full models (all ratios are less than 1). The ratio for keeping the feature only is nearly 1, indicating that the EM initialization is only slightly helpful. All of the hierarchical models outperform the unconditional model, indicating the power of leveraging conditional models; however, when we build a hierarchical model that disregards the classifier's output entirely we achieve only slightly better median RMSE than the unconditional model.

The broader performance of different unconditional, oracular and hierarchical models is shown in Table 3. For `rand3-var`, the accuracy of classification was very high (classification error was only 1.6%). Our experiments confirmed that hierarchical hardness models can achieve almost the same runtime prediction accuracy as oracular models for all four solvers considered in our study. Figure 7 shows that using the hierarchical hardness model to predict `satz`'s runtime is much better than using the unconditional model.

On the `rand3-fix` dataset, results for all four solvers were qualitatively similar: hierarchical hardness models gave slightly but consistently better runtime predictions than unconditional models. On this distribution the gap in prediction accuracy between unconditional and oracular models is already quite small, which makes further significant improvements more difficult to achieve. Detailed analysis of actual vs predicted runtimes for `satz` (see Figure 8) showed that particularly for unsatisfiable instances, the hierarchical model tends to produce slightly more accurate predictions. Further investigation confirmed that those instances in Figure 8 (right) that are far away from the ideal prediction

Table 3. Comparison of oracular, unconditional and hierarchical hardness models. The second number of each entry is the ratio of the model's RMSE to the oracular model's RMSE. (*For `SW-GCP`, even the oracular model has large runtime prediction error.)

Solvers	RMSE (`rand3-var` models) oracular	uncond.	hier.	RMSE (`rand3-fix` models) oracular	uncond.	hier.
`satz`	0.329	0.385(85%)	0.344(96%)	0.343	0.420(82%)	0.413(83%)
`march_dl`	0.283	0.396(71%)	0.306(92%)	0.444	0.542(82%)	0.533(83%)
`kcnfs`	0.294	0.373(79%)	0.312(94%)	0.397	0.491(81%)	0.486(82%)
`oksolver`	0.356	0.443(80%)	0.378(94%)	0.497	0.596(83%)	0.587(85%)
Solvers	**RMSE (`QCP` models) oracular**	**uncond.**	**hier.**	**RMSE (`SW-GCP` models)* oracular**	**uncond.**	**hier.**
`zchaff`	0.303	0.675(45%)	0.577(53%)	0.657	0.993(66%)	0.983(67%)
`minisat`	0.305	0.574(53%)	0.500(61%)	0.682	1.022(67%)	1.024(67%)
`satzoo`	0.240	0.397(60%)	0.334(72%)	0.384	0.581(66%)	0.581(66%)
`satelite`	0.247	0.426(58%)	0.372(66%)	0.618	0.970(64%)	0.978(63%)
`sato`	0.375	0.711(53%)	0.635(59%)	0.723	1.352(53%)	1.345(54%)
`oksolver`	0.427	0.548(78%)	0.506(84%)	0.601	1.337(45%)	1.331(45%)

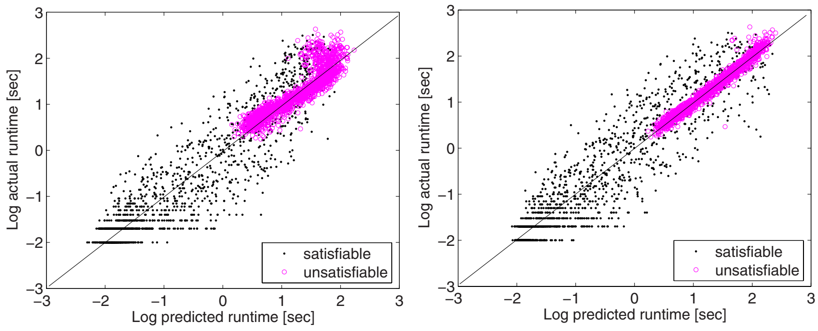


Fig. 7. Actual vs predicted runtime for `satz` on `rand3-var`. Left: unconditional model (RMSE=0.387); right: hierarchical model (RMSE=0.344).

line($y = x$) have low classification confidence. (We further discuss the relationship between classification confidence and runtime prediction accuracy at the end of this section.)

For the structured `QCP` instances, we observed similar runtime prediction accuracy improvements by using hierarchical models. Since the classification accuracy for `QCP` was higher than the classification accuracy for `rand3-fix`, we expected bigger improvements when using the hierarchical hardness model compared to the `rand3-fix` case. Our experimental results confirmed this hypothesis (Figure 9). For example, a hierarchical model for the `satelite` solver achieved a RMSE of 0.372, compared to 0.462 obtained from an unconditional model (whereas the oracular model yielded RMSE 0.247).

However, the runtime prediction accuracy obtained by hierarchical hardness models depends on the quality of the underlying conditional models (experts). In the case of data set `SW-GCP` (see Figure 10), we found that both unconditional and oracular models had fairly large prediction error (RMSE about 0.6; since

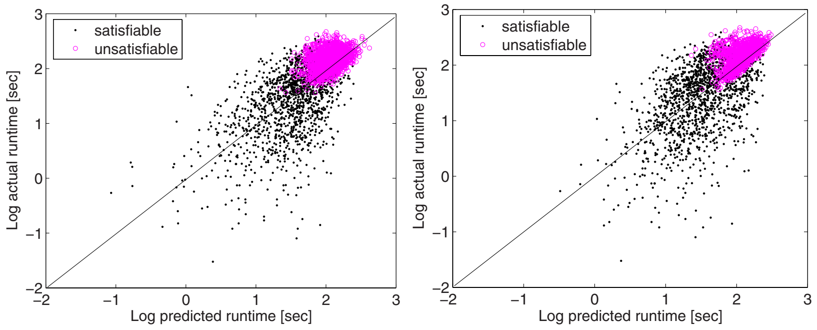


Fig. 8. Actual vs predicted runtime for `satz` on `rand3-fix`. Left: unconditional model (RMSE=0.420); right: hierarchical model (RMSE=0.413).

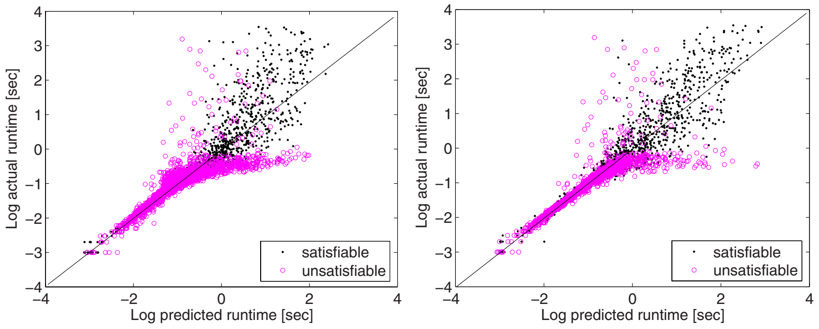


Fig. 9. Actual vs predicted runtime for `satelite` on `QCP`. Left: unconditional model (RMSE=0.426); right: hierarchical model (RMSE=0.372).

we used log runtime, this means that runtime predictions were off by about half an order of magnitude on average). As mentioned in Section 4, we believe that this is because our features are not as informative when applied to this data set as for the other three instance distributions. This is also consistent with the fact that the classification error on `SW-GCP` is much higher (26.6%, compared to 4.0% on `QCP` and 13.5% on `rand3sat-fix`).

When investigating the relationship between the classifier's confidence and regression runtime prediction accuracy, we found that higher classification confidence tends to be indicative of more accurate runtime predictions. This relationship is illustrated in Figure 11 for the `satelite` solver on the `QCP` data set: when the classifier is more confident about the satisfiability of an instance, both prediction error (Figure 11, left) and RMSE (Figure 11, right) are smaller.[1]

Though space constraints preclude a detailed discussion, we also observed that the features important for classification were similarly important for regression. For instance, only using the three features that were most important for

[1] Closer inspection of the raw data shown in Figure 11, left, revealed that a large number of the data points appear at $(0, 0)$ and $(1, 0)$. This is also reflected in the shape of the curve in the right pane of Figure 11.

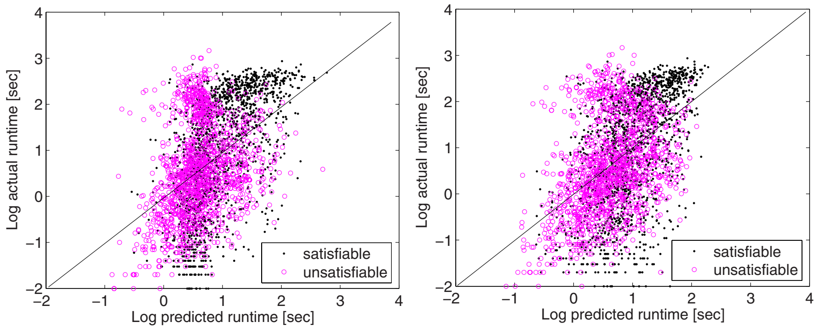


Fig. 10. Actual vs predicted runtime for `zchaff` on `SW-GCP`. Left: unconditional model (RMSE=0.993); right: hierarchical hardness model (RMSE=0.983).

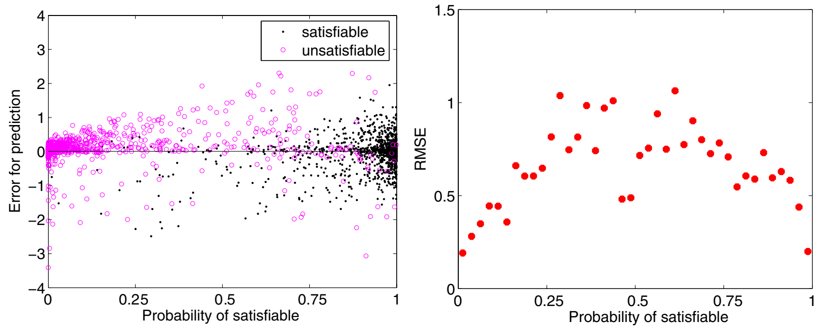


Fig. 11. Classifier output vs runtime prediction error (left); relationship between classifier output and RMSE (right). Data set: `QCP`, solver: `satelite`.

classification on `QCP` data, we achieved runtime prediction RMSE within 10% of the full model's accuracy for `satelite`.

6 Conclusions and Future Work

We have shown that there are big differences between models trained only on satisfiable and unsatisfiable instances, not only for uniform random 3-SAT (as was previously reported in [18]), but also for distributions of structured SAT instances, such as `QCP` and `SW-GCP`. Furthermore, these models have higher prediction accuracy than the respective unconditional models. A classifier can be used to distinguish between satisfiable and unsatisfiable instances with surprisingly high (though not perfect) accuracy. We have demonstrated how such a classifier can be combined with conditional hardness models into a hierarchical hardness model using a mixture-of-experts approach. In cases where we achieved high classification accuracy, the hierarchical models thus obtained always offered substantial improvements over an unconditional model. When the classifier was less accurate, our hierarchical models did not offer a substantial improvement over the unconditional model; however, hierarchical models were never significantly worse. It should be noted that our hierarchical models come at virtually no additional computational cost, as they depend on the same features as used for the individual regression models. The practical usefulness of our approach was recently demonstrated by our algorithm portfolio solver for SAT, SATzilla-07, which, utilizing hierarchical hardness models, placed 1st in three categories of the 2007 SAT competition [20].

In future work, we intend to investigate new instance features to improve both classification and regression accuracy on `SW-GCP`. Furthermore, we will test our approach on more real-world problem distributions, such as software and hardware verification problems. We also plan to study hierarchical models based on partitioning SAT instances into classes other than satisfiable and unsatisfiable—for example, such classes could be used to build different models for different underlying data distributions in heterogeneous data sets.

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Heidelberg (2006)
2. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: IJCAI-91, pp. 331–337 (1991)
3. Dubois, O., Dequen, G.: A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In: IJCAI-01, pp. 248–253 (2001)
4. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
5. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 502–518. Springer, Heidelberg (2005)
6. Gent, I.P., Hoos, H.H., Prosser, P., Walsh, T.: Morphing: Combining structure and randomness. In: AAAI-99, pp. 654–660 (1999)
7. Gomes, C., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. of Automated Reasoning 24(1), 67–100 (2000)
8. Heule, M., Maaren, H.V.: march_dl: Adding adaptive heuristics and a new branching strategy. J. on Satisfiability, Boolean Modeling and Computation 2, 47–59 (2006)
9. Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K.: Performance prediction and automated tuning of randomized and parametric algorithms. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 213–228. Springer, Heidelberg (2006)
10. Joachims, T.: Making large-scale support vector machine learning practical. In: Advances in Kernel Methods: Support Vector Machines, pp. 169–184 (1998)
11. Krishnapuram, B., Carin, L., Figueiredo, M., Hartemink, A.: Sparse multinomial logistic regression: Fast algorithms and generalization bounds. IEEE Trans. on Pattern Analysis and Machine Intelligence, 957–968 (2005)
12. Kullmann, O.: Heuristics for SAT algorithms: Searching for some foundations. Technical report (September 1998)
13. Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y.: Boosting as a metaphor for algorithm design. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 899–903. Springer, Heidelberg (2003)
14. Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y.: A portfolio approach to algorithm selection. In: IJCAI-03, pp. 1542–1543 (2003)

15. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 556–572. Springer, Heidelberg (2002)
16. Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability problems. In: Smolka, G. (ed.) Principles and Practice of Constraint Programming - CP97. LNCS, vol. 1330, pp. 341–355. Springer, Heidelberg (1997)
17. Murphy, K.: The Bayes Net Toolbox for Matlab. In: Computing Science and Statistics: Proc. of the Interface, vol. 33 (2001), `http://bnt.sourceforge.net/`
18. Nudelman, E., Leyton-Brown, K., Hoos, H.H., Devkar, A., Shoham, Y.: Understanding random SAT: Beyond the clauses-to-variables ratio. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 438–452. Springer, Heidelberg (2004)
19. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. Artificial Intelligence 81, 17–29 (1996)
20. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla-07: The design and analysis of an algorithm portfolio for SAT. In: CP-07 (2007)
21. Zhang, H.: SATO: an efficient propositional prover. In: Proc. of the Int'l. Conf. on Automated Deduction, pp. 272–275 (1997)
22. Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in Boolean satisfiability solver. In: Proc. of the Int'l. Conf. on Computer Aided Design, pp. 279–285 (2001)