Hierarchical Hardness Models for SAT

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

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

Abstract. Empirical hardness models are a recent approach for studying $\mathcal{NP}$-hard problems. They predict the runtime of an instance using 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 method by first training separate hardness models for each class. The probability that a novel instance belongs to each class is then computed by a classifier. Finally, a hierarchical hardness model is built using a linear combination of each class's model. To our best knowledge, this research is the first approach that uses a hierarchical model to study a problem's empirical hardness. 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. Compared to unconditional models, our experiments show that hierarchical hardness models tend to have higher runtime prediction accuracy. Furthermore, the classifier's confidence correlated 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 as the problem size increases. In practice, however, many large instances of $\mathcal{NP}$-hard problems still can 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[14, 16]. One recent approach uses linear basis function regression to obtain models of the time an algorithm will require to solve a given SAT instance [16]. These empirical hardness models can be used to obtain 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 [12, 13]. Empirical hardness models have also proven very useful for combinatorial auctions [14], an prominent $\mathcal{NP}$-hard optimization problem. In Section 2, we introduce some background knowledge about empirical hardness models as well as our experiment setup.

Considering the SAT problem in particular, previous work has suggested that very different models are needed to make accurate runtime predictions for satisfiable and unsatisfiable problem instances [16]. Furthermore, models for each type of instance are simpler and more accurate than models that must handle both types. This suggests that it might be possible to build better empirical hardness models by first using a classifier to predict whether an instance is satisfiable. In this work, we investigate this idea. We consider a variety of both structured and unstructured SAT instances, and several state-of-the-art SAT solvers. This experimental setup is described in Section 2.2.

In Section 3 we study the feasibility of predicting the satisfiability of a novel SAT instance from a known distribution, using Sparse Multinomial Logistic Regression (SMLR) [10] as our classifier. Our experimental results are very promising: For the distribution we found to be easiest, the prediction accuracy was greater than 98% (uniform random 3-SAT instances with variable constrainedness); for the trickiest problems we encountered (SAT-encoded graph coloring problems on small-world graphs), accuracy was still greater than 73%.

Armed with a reasonably accurate (but imperfect) classifier, in Section 4 we consider the construction of hierarchical hardness models in order to make runtime predictions. Specifically, we trained empirical hardness models using quadratic basis-function regression for both satisfiable and unsatisfiable training instances. On test data we evaluated both models, and weighted each model's prediction by the classifier's predicted probability that the given model is the right one to use. 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

One goal of using empirical hardness models is to predict the runtime of an algorithm based on some polytime-computable features. A wide variety of different regression techniques can be used for this purpose. In this research, we use the same ridge linear regression method that has previously proven to be very successful for runtime prediction on uniform random SAT and combinational auctions [16, 14].

2.1 Empirical Hardness Models

In order to predict the runtime of an algorithm $\mathcal{A}$ on a distribution $\mathcal{D}$ of problem instances, we run algorithm $\mathcal{A}$ for a number of instances drawn from $\mathcal{D}$ and compute for each instance i a set of features $\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,k}]$. 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, r_i. Unfortunately, our set of features typically includes members which are either unpredictive or highly correlated. Therefore, we reduce the set of features by performing feature selection (e.g., forward selection, backward selection). Finally, we perform a basis function expansion of our feature set: $\boldsymbol{\phi}_i = \boldsymbol{\phi}(\boldsymbol{x}_i) = [\phi_1(\boldsymbol{x}_i), \ldots, \phi_D(\boldsymbol{x}_i)]$. Our basis functions can include arbitrarily complex functions of *all* features $\boldsymbol{x}_i$ of an instance, or can simply return the raw features themselves.

We then use ridge regression to fit the free parameters $\boldsymbol{w}$ of the linear function $f_{\boldsymbol{w}}(\boldsymbol{x}_n) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)$. 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 that prevents arbitrary big free parameters in $\boldsymbol{w}$ and increases numerical stability. Given a new, unseen instance j, a runtime prediction can be obtained by computing its features $\boldsymbol{x}_j$ and evaluating $f_{\boldsymbol{w}}(\boldsymbol{x}_j) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{x}_j)$.

2.2 Experimental Setup

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

- `rand3-var`: uniform-random 3-SAT with 400 variables and clauses-to-variables-ratio randomly selected from [3.26, 5.26]. We generated 20,000 instances with a satisfiable/unsatisfiable ratio of 50/50.
- `rand3-fix`: uniform-random 3-SAT with 400 variables from the solubility phase transition (clauses-to-variables-ratio 4.26) [1, 17]. We generated 20,000 instances with a satisfiable/unsatisfiable ratio of 50.7/49.3.
- `QCP`: random quasi-group completion (the task of determining whether the remaining 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 SAT-encoded 20,000 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. We randomly split each data set used in this paper into training, validation and testing 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.

For each instance, we computed the 84 features described by Nudelman et al. [16]. 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 simple basis function expansions exceeded the 2GB of memory 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 runtime prediction by 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 which perform well on these distributions: `kcnfs`[2], `oksolver`[11], `march_dl`[8], and `satz`[15]. For structured SAT instances, we ran six solvers which perform well on these distributions: `oksolver`, `zchaff`[20], `sato`[19], `satelite`[3], `minisat`[4], and `satzoo`[4].

Note that in the 2005 SAT Solver Competition, `satelite` won gold medals for the Industrial and Handmade SAT+UNSAT categories; `minisat` and `zchaff` won silver and bronze, respectively, for Industrial SAT+UNSAT; and `kcnfs` and `march_dl` won gold and silver, respectively, in the Random SAT+UNSAT complete solvers category. All our experiments were performed using a compute cluster consisting of 50 nodes with dual Intel Xeon 3.2GHz CPU with 2MB cache and 2GB RAM each, running Suse Linux 9.1. All runs of any solver that exceeded 1 CPU hour were terminated prematurely and included into our database of experimental results with a runtime of 1 CPU hour; this timeout occurred in less than 3% of all runs.

3 Using Classification to Estimate Algorithm Output

From previous research [16], 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. If we had an oracle that could determine the satisfiability of an unsolved test instance, we could use such models to predict its runtime; in the following, we will refer to such an oracle-based scheme as a "magic" model. We can infer from the results of Nudelman et al. [16] that on uniform-random 3-SAT, magic models could achieve much higher prediction accuracies than unconditional models, i.e., models trained on the combined distribution of satisfiable and unsatisfiable instances that do not require any oracle-based knowledge.

From our experiments, we found that this phenomenon extends to solvers and distributions not studied previously. Figure 1 shows the difference between using magic models and unconditional models on structured SAT instances (distribution: `QCP`, solver: `satelite`). The magic models were trained on satisfiable instances and unsatisfiable instances separately. In this case we observed almost perfect predictions of runtime for unsatisfiable instances (Figure 1, right). Unconditional models, however, were trained on a mixture of two types of instances. Figure 1, left, shows that the runtime prediction for unsatisfiable instances made by unconditional models can exhibit both less accuracy and more bias. (Throughout the figures in this paper, we use '○' to represent unsatisfiable instances and '●' to represent satisfiable instances.)

Even though using the right type of model can result in a higher prediction accuracy, we found that there is a big penalty for using the wrong type of model to predict the runtime of an instance. We define models trained on satisfiable (unsatisfiable) instances only as M_{sat} (M_{unsat}). Figure 2 left shows that if we used M_{sat} to predict the runtime of an unsatisfiable instance, the prediction error could be very large. The large bias in the inaccurate predictions indicates that models trained on different types of instances are completely different.

We observed similar phenomena for all other data sets and all other solvers. Detailed information is shown in Table 1. These results suggest that magic mod-

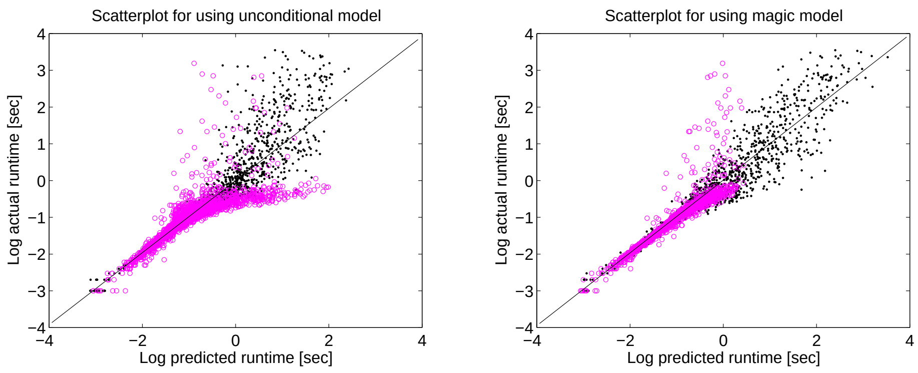


Fig. 1. *Comparison of unconditional model (left, RMSE=0.436) and magic model (right, RMSE=0.295). Distribution:* `QCP`*, solver:* `satelite`*.*

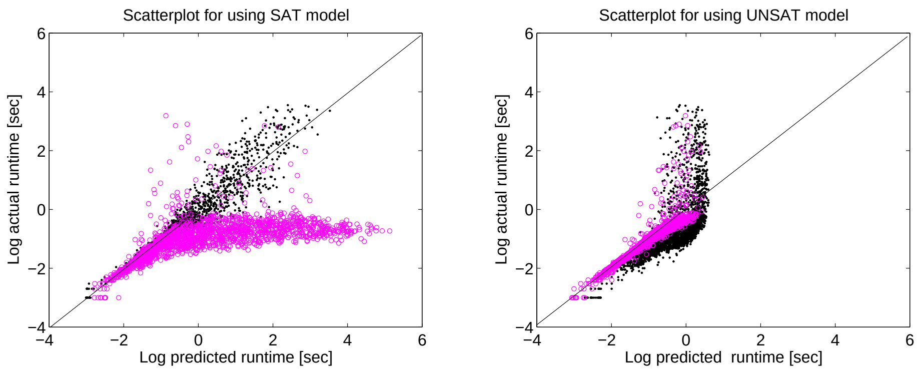


Fig. 2. *Actual vs predicted runtime using M_{sat} to predict sat+unsat instances (left, RMSE=1.185) and M_{unsat} to predict sat+unsat instances (right, RMSE=0.682). Distribution:* `QCP`*, solver:* `satelite`*.*

els always perform better (in terms of a smaller RMSE) than unconditional models. The very large predication 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 runtime of mixture instances was as high as 29.790(distribution: `QCP`, solver: `oksolver`). This indicates that the models trained on unsatisfiable instances fail to make accurate predictions for satisfiable instances.

It is thus clear that approximating the performance of magic models offers the possibility of performance gains for empirical hardness models. However, we can also see that using the wrong model (if a satisfiable instance is incorrectly labeled as unsatisfiable or vice versa) could result in a large *reduction* in performance. The key problem is therefore to find a reasonably accurate, yet computationally efficient way of distinguishing between satisfiable and unsatisfiable instances. In the following, we investigate the use of classification techniques from machine learning for solving this problem.

	RMSE for `rand3-var` models				**RMSE for `rand3-fix` models**			
Solvers	sat.	unsat.	unconditional	magic	sat.	unsat.	unconditional	magic
`satz`	4.532	3.361	0.389	0.343	0.461	0.831	0.422	0.388
`march_dl`	8.727	3.275	0.466	0.309	0.592	1.083	0.545	0.502
`kcnfs`	5.131	3.159	0.413	0.311	0.538	0.978	0.497	0.452
`oksolver`	7.284	3.952	0.558	0.382	0.663	1.166	0.597	0.547

	RMSE for `QCP` models				**RMSE for `SW-GCP` models**			
Solvers	sat.	unsat.	unconditional	magic	sat.	unsat.	unconditional	magic
`zchaff`	2.031	0.955	0.634	0.446	1.222	1.219	1.014	0.820
`minisat`	1.631	0.909	0.621	0.434	1.296	1.315	1.052	0.846
`satzoo`	1.318	7.224	0.450	0.347	0.692	0.784	0.581	0.441
`satelite`	1.185	0.682	0.436	0.295	1.190	1.219	0.943	0.752
`sato`	1.692	9.969	0.688	0.539	1.792	2.014	1.399	0.976
`oksolver`	1.315	29.790	0.701	0.600	1.467	1.941	1.132	0.739

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

3.1 Classifying SAT Instances with SMLR

The goal of a classification algorithm is to leverage a set of n training samples in order to design a classifier that is capable of distinguishing between m classes on the basis of an input vector of length d. As above, this input vector should be understood as comprising d basis functions computed from a set of observed features.

Sparse Multinomial Logistic Regression (SMLR) [10] is a recently developed sparse classification algorithm and can be counted among the state-of-the-art techniques in supervised learning. Like relevance vector machines (RVMs) [18] and sparse probit regression (SPR) [5], it learns classifiers that incorporate weighted sums of basis functions. Its use of sparsity-promoting priors encourages weight estimates to be either significantly large or exactly zero. This technique controls the capacity of the learned classifier by minimizing the number of basis functions used, and thereby tends to result in better generalization. SMLR learns a sparse multi-class classifier that scales favorably in both the number of training samples and the feature dimensionality, which is important for our problems since we have tens of thousands of samples per data set. We also evaluated different classifiers, such as support vector machines (SVMs) [9]; however, we found that SMLR achieved better classification accuracy.

We applied SMLR to build a classifier to distinguish satisfiable and unsatisfiable SAT instances, using as basis functions the same set of raw features as for the previously described hardness models. The main difference between the training data used for regression and classification lies in the fact that the former uses runtimes of a given solver, while the latter uses class labels (sat and unsat). Also, for classification we only used the raw features as basis functions. The outputs of the classifier are the probabilities for an input instance to belong to the two classes. Obviously, all the probabilities sum to 1, and we label the instance with the class that has the highest probability. We define the classification scores as $(P(C_{sat}|x) \times 2) - 1$. Hence, for instances that are predicated completely con-

Dataset	Classification Accuracy sat.	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

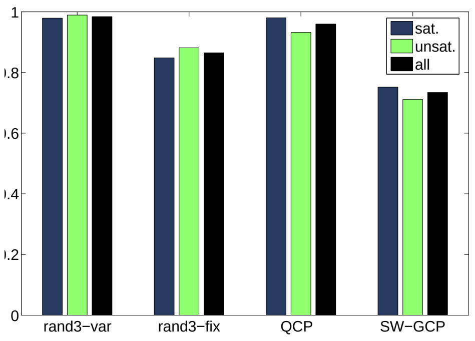


Fig. 3. *Classification accuracy for different data sets*

fidently as satisfiable or unsatisfiable, the classification scores would be equal to 1 or -1, respectively, while in cases where the classifier is very uncertain about the class of an instance, it gives a classification score close to 0.

Our experimental results are very encouraging. For `rand3-var` and `QCP`, classification accuracies are very high. The classifier was usually very confident about the satisfiability of an instance, even though its prediction depended only on computationally inexpensive features. The `rand3-fix` and `SW-GCP` distributions were harder for our classifier, but overall accuracy was never lower than 73%, which is substantially better than random guessing. We evaluated the quality of the classification by two factors: overall classification error and the fraction of instances with very high classification accuracy. Detailed information is shown in Figures 3, 4 and 5.

Note that for the `rand3-var` data set, the overall classification error is only 1.6%. Only using clauses-variables-ratio (bigger or less than 4.26) as a basis for predicting the satisfiability of an instance yields an error of 3.7%; hence, SMLR reduces the error to less than half. On the `QCP` data set, SMLR, achieved an overall classification accuracy of about 96.0%. Detailed classification results for these two data sets are shown in Figure 4.

In addition to the very high overall accuracy of classifications, there is a big fraction of instances with a very high classification score. Furthermore, we observed are strong relationships between classification scores and classification accuracies. We observed that the more confident the classifier is (absolute value of classification score close to 1), the more accurate the classification will be. This observation will later be exploited, as we construct hierarchical empirical hardness models. Note that there are some instances with classification scores closes to 0, but with a high classification accuracy. However, the number of such instances is very small. Indeed, the smallest bins in the histograms in the lower left and right parts of Figure 4 correspond to 0.13% and 0.43% of the total instances respectively.

On the `rand3sat-fix` and `SW-GCP` distributions, our SMLR classifiers reach accuracies of about 86% and 73%, respectively (see Figure 3). Based on our efforts to construct unconditional empirical hardness models for `SW-GCP`, we have

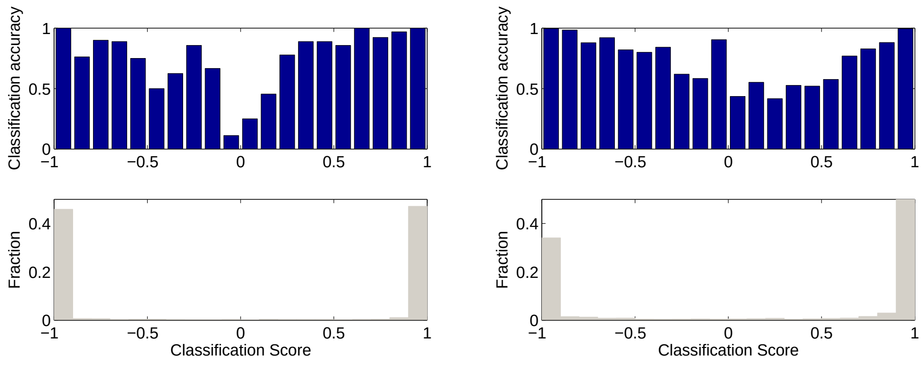


Fig. 4. *The classification accuracies vs classification scores (top) and the fractions of instances vs classifier scores (bottom). Left:* `rand3-var`*, right:* `QCP`*.*

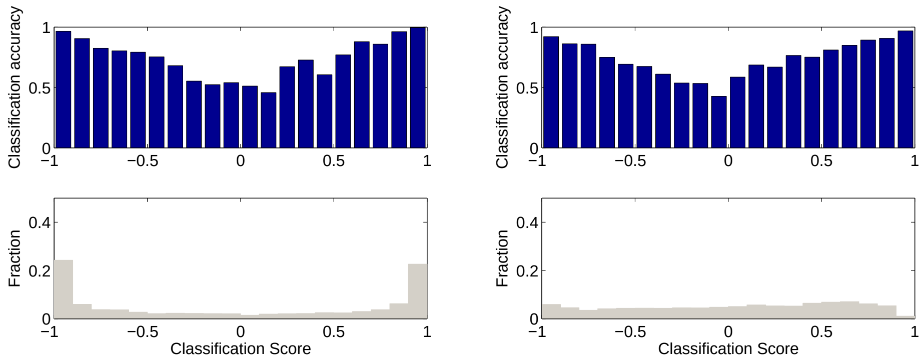


Fig. 5. *The classification accuracies vs classification scores (top) and the fractions of instances vs classifier scores (bottom). Left:* `rand3-fix`*, right:* `SW-GCP`*.*

found evidence that our features are less predictive on this distribution, which we believe to be the main reason for the lower accuracy of our classifier on the `SW-GCP` instance set. Figure 5 shows that the fraction of instances with high classification scores is smaller for these distributions than for `rand3-var` and `QCP`, indicating that the classifiers are uncertain about far more of the instances. However, even for `SW-GCP` we still see a strong relationship between classification score and classification accuracy on test data.

One interesting finding is that our classifier is also able to achieve very high accuracy with a very small number of features. For example on `QCP` data, SMLR 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. (For more detailed descriptions of the features, see [16].)

Overall, our experiments show that a classifier may be used to make surprisingly accurate polytime-predictions about the satisfiability of SAT instance.

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 using 5 features	98.4%	86.5%
Accuracy using 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 using 5 features	93.0%	73.2%
Accuracy using all features	96.0%	73.4%

Table 2. Five most important features (from most to least important) for classification selected by backward selection.

Of course, given the $\mathcal{NP}$-hardness of SAT, we cannot expect this to be true in general, but it certainly holds for the widely studied instance distributions used 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 which 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.

We are interested in combining classifiers with empirical hardness models. In this case, all of the features must be computed for the hardness model anyway, so the additional computational effort required to evaluate the classifier's prediction for a novel instance is very low. In the next section we evaluate whether combining these models can lead to an improvement in the accuracy of runtime predictions.

4 Hierarchical Hardness Models

Generally, based on a classifier that is used for partitioning instances into m sets, a hierarchical hardness models for a given instance distribution and solver can be constructed as follows.

Step 1: The classifier is trained on a set of training instances and subsequently tested on validation and ultimately test data. For every instance in the validation and test set, $P(C_i|x)$, the (predicted) probability of an instance belonging to class C_i ($i \in [1, ..., m]$) is recorded.

Step 2: m empirical hardness models M_i $(i \in [1, ..., m])$ are trained using the portion of the training data which consists of instances that belong to each class C_i. Each of the resulting models M_i is used to predict the runtime for every instance in the test set, and the responses r_i $(i \in [1, ..., m])$ are recorded. Note that the RMSE for using one model M_i to predict the runtime of all instances may be big, since M_i is only trained on instances of class C_i.

Step 3: For each instance, the probability distribution of the predicted runtime is $P(r|x) = \sum_{i=1}^{m} P(r|x, C_i) \cdot P(C_i|x)$. Here, $P(r|x, C_i)$ is the probability distribution of predicted runtime if we know the instance belonging to class C_i. For linear regression, $P(r|x, C_i) = \mathcal{N}(r|f(x, w_i), \beta_i^{-1})$. $P(C_i|x)$ is the probability of the instance belonging to class C_i given observed features x. Hence, $P(r|x)$ is a mixture of linear regression models. Therefore, the expected value of runtime prediction using the hierarchical hardness model is as follows:

$$\begin{aligned}
\mathbb{E}(r) &= \int r \cdot P(r|x)dr \\
&= \int r \cdot \sum_{i=1}^{m} P(r|x, C_i) \cdot P(C_i|x)dr \\
&= \sum_{i=1}^{m} r_i \cdot P(C_i|x)
\end{aligned}$$

Hence, we can compute the predicted response by using the linear combination of r_i $(i \in [1, ..., m])$ computed in Step 2 weighted by the class membership probabilities $P(C_i|x)$ obtained from the classifier from Step 1. The higher the probability of an instance belonging to class C_i, the more weight the prediction r_i will have. In particular, if the classifier is 100% confident that an instance belongs to a single class C_i, then the expected response using the hierarchical hardness model is the response of using the M_i model only.

4.1 Hierarchical Hardness Models for SAT

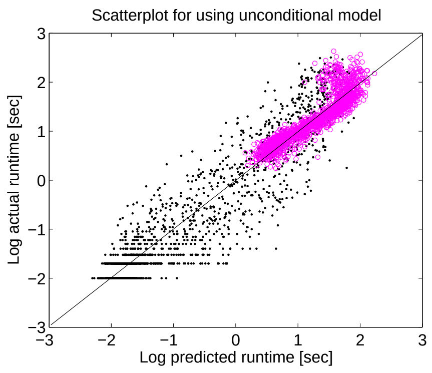
Applying this method for constructing hierarchical hardness models to our SAT domain, we have two candidate classes for each instance $(m = 2)$: satisfiable and unsatisfiable. The responses of using hierarchical hardness models consist of linear combinations of responses using M_{sat} and M_{unsat}, namely $r = r_{sat} \cdot P(C_{sat}|x) + r_{unsat} \cdot P(C_{unsat}|x)$. The performances of unconditional models, magic models, and our hierarchical models are shown in Table 3.

For `rand3-var`, the accuracy of classification was very high. We therefore know almost for certain whether an instance is satisfiable (classification error was only 1.6%). Our experiments confirmed that hierarchical hardness models can achieve almost the same runtime prediction accuracy as the magic model (RMSE 0.343 for solver `satz`). Figure 6 shows that using the hierarchical hardness model to predict runtime is much better than using the unconditional model. For example, note that on unsatisfiable instances, the prediction plot for the unconditional model bends, indicating bias in the predictions.

Solvers	**RMSE (`rand3-var` models)** magic	uncond.	hier.	**RMSE (`rand3-fix` models)** magic	uncond.	hier.
`satz`	0.343	0.389(88%)	0.343(100%)	0.388	0.422(92%)	0.416(93%)
`march_dl`	0.309	0.466(66%)	0.311(99%)	0.502	0.545(92%)	0.534(94%)
`kcnfs`	0.311	0.413(75%)	0.311(100%)	0.452	0.497(91%)	0.487(93%)
`oksolver`	0.382	0.558(68%)	0.385(99%)	0.547	0.597(92%)	0.591(93%)

Solvers	**RMSE (`QCP` models)** magic	uncond.	hier.	**RMSE (`SW-GCP` models)*** magic	uncond.	hier.
`zchaff`	0.446	0.634(70%)	0.559(80%)	0.820	1.014(81%)	1.008(81%)
`minisat`	0.434	0.621(70%)	0.547(79%)	0.846	1.052(80%)	1.072(80%)
`satzoo`	0.347	0.450(77%)	0.400(87%)	0.441	0.581(76%)	0.587(75%)
`satelite`	0.295	0.436(68%)	0.373(79%)	0.752	0.943(80%)	0.967(78%)
`sato`	0.539	0.688(78%)	0.614(88%)	0.976	1.399(70%)	1.409(69%)
`oksolver`	0.600	0.701(86%)	0.647(93%)	0.739	1.132(65%)	1.175(63%)

Table 3. Comparison of magic, unconditional and hierarchical hardness models. The second number of each entry is the ratio of the model's RMSE to the magic model's RMSE. (* The runtime prediction error is very big even using magic model.)



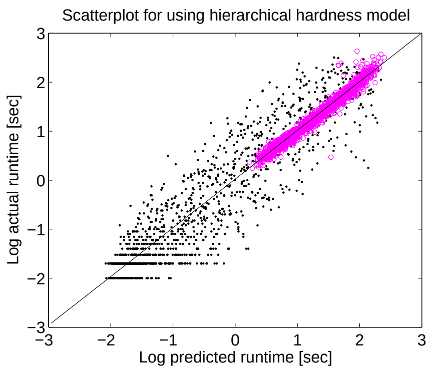


Fig. 6. *Actual vs predicted runtime plots for solver:* `satz`*, data set:* `rand3-var`*. Left: unconditional model, RMSE=0.389; right: hierarchical hardness model, RMSE=0.343.*

Now we turn to the `rand3-fix` dataset. Results for all four solvers were qualitatively similar on this distribution; here we discuss `satz`. Using the hierarchical hardness model to predict the runtime has RMSE 0.416. Compared to the unconditional model(RMSE=0.422), the performance of the hierarchical model was a little bit closer to the ideal magic model (RMSE=0.388). Since the unconditional model already achieved performance similar to the magical models for this data set, we did not find huge improvement in terms of RMSE moving from unconditional models to hierarchical models. A more intuitive view is shown in Figure 7. The big sparse cloud for the unsatisfiable instances in the scatter plot of the unconditional model becomes tighter and closer to the ideal prediction ($y = x$) in the scatter plot of the hierarchical model. Further investigation confirms that those instances in Figure 7 (Right) which are far away from the ideal prediction line ($y = x$) have a low classification confidence. This phenomenon

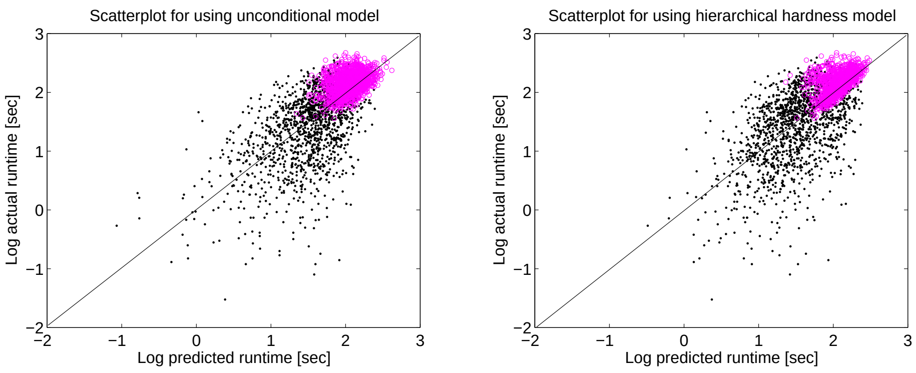


Fig. 7. *Actual vs predicted runtime plots for solver:* `satz`*, data set:* `rand3-fix`*. Left: unconditional model, RMSE=0.422; right: hierarchical hardness model, RMSE=0.416.*

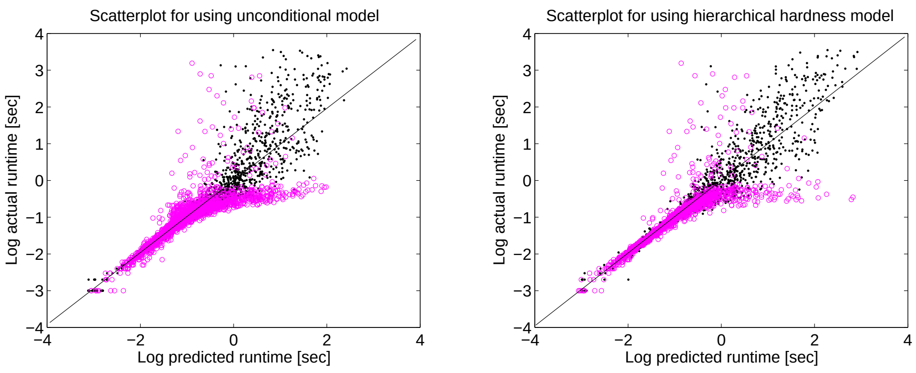


Fig. 8. *Actual vs predicted runtime plots for solver:* `satelite`*, data set:* `QCP`*. Left:unconditional model RMSE=0.436, Right:hierarchical hardness model RMSE=0.373*

suggests that some relationship exists between classification score and prediction accuracy; we investigate this relationship at the end of this section.

For structured instance set `QCP`, we observed similar prediction accuracy improvements by using a hierarchical model. Since the classification accuracy for `QCP` is higher than the classification accuracy for `rand3-fix`, we expected bigger improvements when using the hierarchical hardness model compared to the `rand3-fix` case. The experimental results confirmed our hypothesis (Figure 8). For the solver `satelite`, the RMSE for using the unconditional model is 0.436. However, the RMSE for using the hierarchical model is 0.373, which is much closer to that of the magic model (RMSE=0.295).

The data set `SW-GCP` generally caused some difficulties (see Figure 9). We found that both unconditional and hierarchical models have fairly large prediction error (RMSE about 1.0; since we used log runtime, this means that runtime predictions were off by about an order of magnitude on average). As discussed

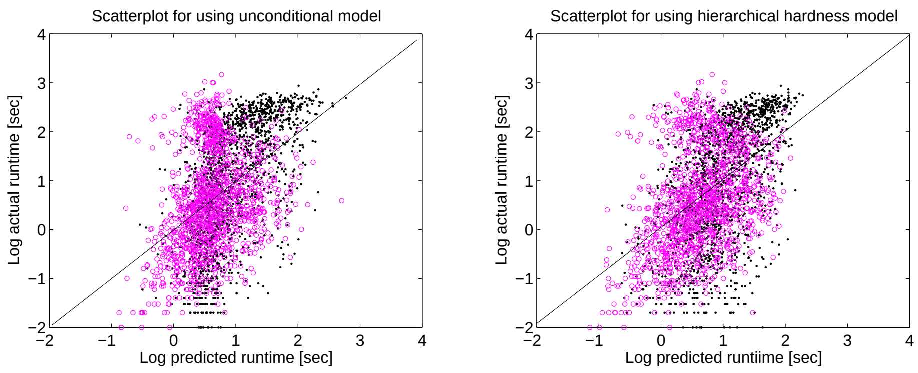


Fig. 9. *Actual vs predicted runtime plots for solver:* `zchaff`*, data set:* `SW-GCP`*. Left:unconditional model RMSE=1.014, Right:hierarchical hardness model RMSE=1.008*

earlier, our relative difficulty in making accurate predictions on this dataset indicates that our features are less informative here than on the other three datasets. It was not surprising that the weight combination of two bad models did not provide us with a good model, especially considering that the classifications themselves were also less accurate than for the other distributions (the overall classification error for `SW-GCP` was 26.6%).

There was a strong relationship between the classifier's confidence and regression runtime prediction accuracy. In particular, a more confident classification was indicative of a more accurate runtime prediction. This relationship is illustrated in Figure 10 for the `satelite` solver on `QCP` data. When the classifiers are very confident about the satisfiability of the instances, the prediction errors (Figure 10 Left) and RMSE (Figure 10, right) is smaller.[1]

Another interesting discovery from our experiments is that those features important to classification are also important for ridge regression. For instance, only using the three features which were most important for classification on `QCP` data, we achieved runtime prediction performance within 10% of full model accuracy for `satelite` (in terms of RMSE on the validation data).

5 Conclusion 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 problems (as was previously shown) but also for structured SAT problems such as

[1] The reader may feel that the classification error actually seems quite large at classification scores of -1 and 1 in Figure 10 (left side). However, this is a situation in which scatterplots are misleading. Remember that there are tens of thousands of points in this plot; as it turns out, most of the points are stacked on top of each other near (-1,0) and (1,0). We know this from examining the raw data; however, it can also be inferred from the shape of the curve in the right pane of Figure 10.

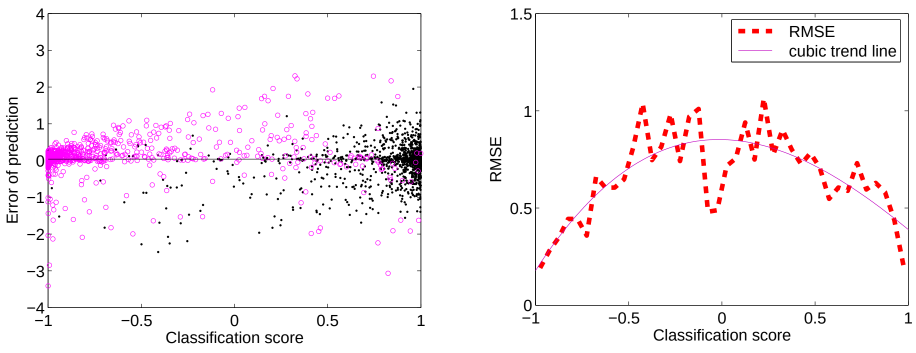


Fig. 10. *Classification score vs. runtime prediction error per instance (Left); relation between classification score and RMSE (Right). Data set:* `QCP`*, solver:* `satelite`

`QCP` and `SW-GCP`. Furthermore, these models have higher prediction accuracy than the unconditional model.

A classifier can be used to distinguish between satisfiable and unsatisfiable instances; in our experiments we achieved accuracies between 73% and 98%. Such a classifier can also be used to build a hierarchical hardness model. In cases where we achieved high classification accuracy, a hierarchical model 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 also used for the individual regression models.

In future work, we intend to investigate new features to improve both classification and regression accuracy on `SW-GCP`. We will also investigate other classification techniques besides SMLR and SVM.

References

1. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.
2. O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 248–253, 2001.
3. N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *Proceedings of the 8th Intl. Conf. on Theory and Applications of Satisfiability Testing, LNCS*, volume 3569, pages 61–75, 2005.
4. N. Eén and N. S Orensson. An extensible SAT-solver. In *Proceedings of the 6th Intl. Conf. on Theory and Applications of Satisfiability Testing, LNCS*, volume 2919, pages 502–518, 2004.

5. M. Figueiredo. Adaptive sparseness for supervised learning. *IEEE Trans. on Pattern Analysis and Mach. Intell.*, 25:1150–1159, 2003.
6. I. P. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 654–660, Orlando, Florida, 1999.
7. C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1):67–100, 2000.
8. M. Heule and H. V. Maaren. march_dl: Adding adaptive heuristics and a new branching strategy. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:47–59, 2006.
9. T. Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines*, pages 169–184. MIT Press, Cambridge, MA, 1998.
10. B. Krishnapuram, L. Carin, M. Figueiredo, and A. Hartemink. Sparse multinomial logistic regression: Fast algorithms and generalization bounds. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 957–968, 2005.
11. O. Kullmann. Heuristics for sat algorithms: Searching for some foundations. *Submitted to Discrete Applied Mathematics (Special Issue on the Satisfiability Problem), 23 pages*, 1998.
12. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. Boosting as a metaphor for algorithm design. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*, pages 899–903, 2003.
13. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 1542–1543, 2003.
14. K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 556–572, 2002.
15. C. M. Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the Principles and Practice of Constraint Programming, LNCS*, volume 1330, pages 342–356, 1997.
16. E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*, pages 438–452, 2004.
17. B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81:17–29, 1996.
18. M. E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.
19. H. Zhang. SATO: an efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction, LNAI*, volume 1249, pages 272–275, 1997.
20. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design*, pages 279–285, 2001.