Understanding Random SAT: Beyond the Clauses-to-Variables Ratio

Eugene Nudelman[1], Kevin Leyton-Brown[2], Holger H. Hoos[2], Alex Devkar[1], and Yoav Shoham[1]

[1] Computer Science Department, Stanford University,Stanford, CA
{eugnud,avd,shoham}@cs.stanford.edu
[2] Computer Science Department, University of British Columbia, Vancouver, BC
{kevinlb,hoos}@cs.ubc.ca

Abstract. It is well known that the ratio of the number of clauses to the number of variables in a random k-SAT instance is highly correlated with the instance's empirical hardness. We consider the problem of identifying such features of random SAT instances automatically using machine learning. We describe and analyze models for three SAT solvers—`kcnfs`, `oksolver` and `satz`—and for two different distributions of instances: uniform random 3-SAT with varying ratio of clauses-to-variables, and uniform random 3-SAT with fixed ratio of clauses-to-variables. We show that surprisingly accurate models can be built in all cases. Furthermore, we analyze these models to determine which features are most useful in predicting whether an instance will be hard to solve. Finally we discuss the use of our models to build `SATzilla`, an algorithm portfolio for SAT.[3]

1 Introduction

SAT is among the most studied problems in computer science, representing a generic constraint satisfaction problem with binary variables and arbitrary constraints. It is also the prototypical $\mathcal{NP}$-hard problem, and its worst-case complexity has received much attention. Accordingly, it is not surprising that SAT has become a primary platform for the investigation of average-case and empirical complexity. Particular interest has been paid to randomly generated SAT instances. In this paper we concentrate on such instances as they offer both a range of very easy to very hard instances for any given input size and the opportunity to make connections to a wealth of existing work.

Early work [15, 2] considered the empirical performance of DPLL-type solvers running on uniform random k-SAT instances, finding a strong correlation between the instance's hardness and the ratio of the number of clauses to the number of variables in the instance. Further, it was demonstrated that the hardest region (*e.g.*, for random 3-SAT, a clauses-to-variables ratio of roughly 4.26) corresponds exactly to a phase transition in an algorithm-independent property of the instance: the probability that a randomly-generated formula having a given ratio will be satisfiable. This well-publicized finding led to increased enthusiasm for the idea of studying algorithm performance experimentally, using the same tools as are used to study natural phenomena. Over the past decade,

[3] We'd like to acknowledge very helpful assistance from Nando de Freitas, and our indebtedness to the authors of the algorithms in the `SATzilla` portfolio. We also thank the anonymous reviewers for helpful comments.

this approach has complemented worst-case analysis of algorithms, improving our understanding of algorithms' empirical behavior with interesting findings on (*e.g.*) islands of tractability [8], search space topologies for stochastic local search algorithms [5, 4], backbones [13], backdoors [18] and random restarts [3].

Inspired by the success of this work on SAT and related problems, in 2002 we proposed a new methodology for using machine learning to study empirical hardness [11]. We applied this methodology to the combinatorial auction winner determination problem (WDP), an $\mathcal{NP}$-hard combinatorial optimization problem equivalent to weighted set packing. In subsequent papers [10, 9] we extended our methodology, demonstrating techniques for improving empirical algorithm performance through the construction of algorithm portfolios and for automatically generating hard benchmark distributions. In this paper we come full-circle and apply our techniques to uniform random 3-SAT—the problem that originally inspired their development.

The work which is perhaps the most related to our own is [7, 14]. There classification techniques are used to categorize runs of CSP and SAT solvers according to length. In [11, 9] we discuss the relationship between this work and our approach in more detail. It is worth pointing out that, different from ours, their work focuses on understanding the behavior of solvers during the run, as opposed to studying the effect of problem structure on hardness. In addition, as argued in [11], standard classifications techniques can be sometimes inappropriate in this context, for example, because of boundary cases.

Our current work has three goals. First, we aim to show that inexpensively-computable features can be used to make accurate predictions about the empirical hardness of random SAT instances, and to analyze these models in order to identify important features. We consider three different SAT algorithms and two different instance distributions. The first distribution contains random 3-SAT instances with a varying ratio of clauses to variables, allowing us to see whether our techniques automatically select the clauses-to-variables ratio as an important feature, and also what other features are important in this setting. Our second distribution contains random 3-SAT instances with the ratio of clauses-to-variables held constant at the phase transition point. This distribution has received much attention in the past; it gives us the opportunity to explain the orders-of-magnitude runtime variation that persists in this so-called "hard region."

Second, we show that empirical hardness models have other useful applications for SAT. Most importantly, we describe a SAT solver, `SATzilla`, which uses hardness models to choose among existing SAT solvers on a per-instance basis. We explain some details of its construction and summarize its performance.

Our final goal is to offer a concrete example in support of our abstract claim that empirical hardness models are a useful tool for gaining understanding of the behavior of algorithms for solving $\mathcal{NP}$-hard problems. Thus, while we believe that our SAT results are interesting in their own right, and while studying random 3-SAT is useful because it allows connection to existing theoretical work, we want to emphasize that very few of our techniques are particular to SAT. Indeed, we have achieved equally strong results applying our methodologies to qualitatively different problems.[4]

[4] WDP, for example, is very different from SAT: while feasible solutions can be identified in constant time, the goal is to find an *optimal* feasible solution, and there is thus no opportunity to

2 Methodology

Although the work surveyed above has led to great advances in understanding the empirical hardness of SAT problems, most of these approaches scale poorly to more complicated domains. In particular, most of these methods involve exhaustive exploration of the search and/or distribution parameter spaces, and require considerable human intervention and decision-making. As the space of relevant features grows and instance distributions become more complex, it is increasingly difficult either to characterize the problem theoretically or to explore its degrees of freedom exhaustively. Moreover, most current work focuses on understanding algorithms' performance profiles, rather than trying to characterize the hardness of individual problem instances.

2.1 Empirical Hardness Models

In [11] we proposed a novel experimental approach for predicting the runtime of a given algorithm on individual problem instances:

1. **Select a problem instance distribution.**
   Observe that the choice of distribution is fundamental—different distributions can induce very different algorithm behavior.
2. **Select one or more algorithms.**
3. **Select a set of inexpensive, distribution-independent features.**
   It is important to remember that individual features need not be perfectly predictive of hardness; ultimately, our goal will be to combine features together. Thus, it is possible to take an inclusive approach, adding all features that seem reasonable and then removing those that turned out to be unhelpful (see step 5). Furthermore, many features that proved useful for one constraint satisfaction or optimization problem can carry over into another.
4. **Generate a set of instances and for each one, determine the running time of the selected algorithms and compute the features.**
5. **Eliminate redundant or uninformative features.**
   Much better models tend to be learned when all features are informative. A variety of statistical techniques are available for eliminating or de-emphasizing the effect of such features. The simplest approach is to manually examine pairwise correlations, eliminating features that are highly correlated with what remains. Shrinkage techniques (such as lasso [16] or ridge regression) are another alternative.
6. **Use machine learning to select a function of the features that predicts each algorithm's running time.**
   Since running time is a continuous variable, regression is the natural machine-learning approach to use for building runtime models. For more detail about why we prefer regression to other approaches such as classification, see [11].

---

terminate the algorithm the moment a solution is found. We also have promising unpublished results for TSP and the computation of Nash equilibria.

2.2 Building Models

There are a wide variety of different regression techniques; the most appropriate for our purposes perform supervised learning.[5] Such techniques choose a function from a given hypothesis space (*i.e.*, a space of candidate mappings from the given features to the running time) in order to minimize a given error metric (a function that scores the quality of a given mapping, based on the difference between predicted and actual running times on training data, and possibly also based on other properties of the mapping). Our task in applying regression to the construction of hardness models thus reduces to choosing a hypothesis space that is able to express the relationship between our features and our response variable (running time), and choosing an error metric that both leads us to select good mappings from this hypothesis space and can be tractably minimized.

The simplest regression technique is linear regression, which learns functions of the form $\sum_i w_i f_i$, where f_i is the i^{th} feature and the w's are free variables, and has as its error metric root mean squared error (RMSE). Linear regression is a computationally appealing procedure because it reduces to the (roughly) cubic-time problem of matrix inversion.

Choosing a Hypothesis Space Although linear regression seems quite limited, it can actually be extended to a wide range of hypothesis spaces. There are two key tricks. The first is to introduce new features that are functions of the original features. For example, in order to learn a model which is a quadratic function of the features, the feature set can be augmented to include all pairwise products of features. A hyperplane in the resulting much-higher-dimensional space corresponds to a quadratic manifold in the original feature space. The key problem with this approach is that the set of features grows quadratically, which may cause the regression problem to become intractable and can also lead to overfitting. Thus, it can make sense to add only a subset of the pairwise products of features; *e.g.*, only pairwise products of the k most important features in the linear regression model. Of course, we can use the same idea to reduce many other nonlinear hypothesis spaces to linear regression: all hypothesis spaces which can be expressed by $\sum_i w_i g_i(\mathbf{f})$, where the g_i's are arbitrary functions and $\mathbf{f} = \{f_i\}$.

Sometimes we want to consider hypothesis spaces of the form $h\left(\sum_i w_i g_i(\mathbf{f})\right)$. For example, we may want to fit a sigmoid or an exponential curve. When h is a one-to-one function, we can transform this problem to a linear regression problem by replacing our response variable y in our training data by $h^{-1}(y)$, where h^{-1} is the inverse of h, and then training a model of the form $\sum_i w_i g_i(\mathbf{f})$. On test data, we must evaluate the model $h\left(\sum_i w_i g_i(\mathbf{f})\right)$. One caveat about this trick is that it distorts the error metric: the error-minimizing model in the transformed space will not generally be the error-minimizing model in the true space. In many cases this distortion is acceptable, however, making this trick a tractable way of performing many different varieties of nonlinear regression. In this paper we use exponential models ($h(y) = 10^y$; $h^{-1}(y) = \log_{10}(y)$) and logistic models ($h(y) = 1/(1 + e^{-y})$; $h^{-1}(y) = \ln(y)\ln(1 - y)$ with values of y first mapped onto the interval $(0, 1)$). Because logistic functions have a finite range, we found them particularly useful for modeling capped runs.

[5] Because of our interests in being able to analyze our models and in keeping model sizes small, we avoid model-free approaches such as nearest neighbor.

2.3 Evaluating the Importance of Variables in a Hardness Model

Once we are able to construct an accurate empirical hardness model, it is natural to try to explain why it works. A key question is which features were most important to the success of the model. It is tempting to interpret a linear regression model by comparing the coefficients assigned to the different features, on the principle that larger coefficients indicate greater importance. This can be misleading for two reasons. First, features may have different ranges, though this problem can be mitigated by normalization. More fundamentally, when two or more features are highly correlated then models can include larger-than-necessary coefficients with different signs. A better approach is to force models to contain fewer variables, on the principle that the best low-dimensional model will involve only relatively uncorrelated features. Once such a model has been obtained, we can evaluate the importance of each feature to that model by looking at each feature's *cost of omission*. That is, we can train a model without the given feature and report the resulting increase in (cross-validated) prediction error. To make them easier to compare, we scale the cost of omission of the most important feature to 100 and scale the other costs of omission in proportion.

There are many different "subset selection" techniques for finding good, small models. Ideally, exhaustive enumeration would be used to find the best subset of features of desired size. Unfortunately, this process requires consideration of a binomial number of subsets, making it infeasible unless both the desired subset size and the number of base features are very small. When exhaustive search is impossible, heuristic search can still find good subsets. We considered four heuristic methods: *forward selection*, *backward elimination*, *sequential replacements* and *LAR*. Since none of these four techniques is guaranteed to find the optimal subset, we combine them together by running all four and keeping the model with the smallest cross-validated (or validation-set) error.

3 Hardness Models for SAT

3.1 Features

Figure 1 summarizes the 91 features used by our SAT models. Since not every feature is useful in every distribution, we discard uninformative or highly correlated features after fixing the distribution. For example, while ratio of clauses-to-variables was important for `SATzilla`, it is not at all useful for the fixed-ratio dataset. In order to keep values to sensible ranges, whenever it makes sense we normalize features by either the number of clauses or the number of variables in the formula.

We divide the features into nine groups. The first group captures problem size, measured by the number of clauses, variables, and the ratio of the two. Because we expect this ratio to be an important feature, we gave it additional expressive power by including squares and cubes of both the ratio and its reciprocal. Also, because we know that features are more powerful in simple regression models when they are directly correlated with the response variable, we include a "linearized" version of the ratio which is defined as the absolute value of the difference between the ratio and the phase transition point, 4.26. It turns out that for variable-ratio data this group of features alone suffices

Problem Size Features:

1. **Number of clauses:** denoted c
2. **Number of variables:** denoted v

3-5. **Ratio:** c/v, $(c/v)^2$, $(c/v)^3$

6-8. **Ratio reciprocal:** (v/c), $(v/c)^2$, $(v/c)^3$

9-11. **Linearized ratio:** $|4.26 - c/v|$, $|4.26 - c/v|^2$, $|4.26 - c/v|^3$

Variable-Clause Graph Features:

12-16. **Variable nodes degree statistics**: mean, variation coefficient, min, max and entropy.

17-21. **Clause nodes degree statistics**: mean, variation coefficient, min, max and entropy.

Variable Graph Features:

22-25. **Nodes degree statistics**: mean, variation coefficient, min, and max.

Clause Graph Features:

26-32. **Nodes degree statistics**: mean, variation coefficient, min, max, and entropy.

33-35. **Weighted clustering coefficient statistics**: mean, variation coefficient, min, max, and entropy.

Balance Features:

36-40. **Ratio of positive and negative literals in each clause**: mean, variation coefficient, min, max, and entropy.

41-45. **Ratio of positive and negative occurrences of each variable**: mean, variation coefficient, min, max, and entropy.

46-48. **Fraction of unary, binary, and ternary clauses**

Proximity to Horn Formula

49. **Fraction of Horn clauses**

50-54. **Number of occurrences in a Horn clause for each variable** : mean, variation coefficient, min, max, and entropy.

LP-Based Features:

55. **Objective value of linear programming relaxation**
56. **Fraction of variables set to 0 or 1**

57-60. **Variable integer slack statistics:** mean, variation coefficient, min, max.

DPLL Search Space:

61-65. **Number of unit propagations:** computed at depths 1, 4, 16, 64 and 256

66-67. **Search space size estimate:** mean depth to contradiction, estimate of the log of number of nodes.

Local Search Probes:

68-71. **Minimum fraction of unsat clauses in a run:** mean and variation coefficient for SAPS and GSAT (see [17]).

72-81. **Number of steps to the best local minimum in a run:** mean, median, variation coefficient, 10^{th} and 90^{th} percentiles for SAPS and GSAT.

82-85. **Average improvement to best:** For each run, we calculate the mean improvement per step to best solution. We then compute mean and variation coefficient over all runs for SAPS and GSAT.

86-89. **Fraction of improvement due to first local minimum:** mean and variation coefficient for SAPS and GSAT.

90-91. **Coefficient of variation of the number of unsatisfied clauses in each local minimum:** mean over all runs for SAPS and GSAT.

Fig. 1. SAT instance features used for constructing our predictive models.

to construct reasonably good models. However, including the rest of our features significantly improves these models. Moreover, in the presence of other features, including higher-order features 4, 5, 7, 8, 10 and 11 does not improve accuracy much and does not qualitatively change the results reported below. Due to space constraints, for the rest of this paper we focus on models that use all of the ratio features.

The next three groups correspond to three different graph representations of a SAT instance. The variable-clause graph (VCG) is a bipartite graph with a node for each variable, a node for each clause, and an edge between them whenever a variable occurs in a clause. The variable graph (VG) has a node for each variable and an edge between variables that occur together in at least one clause. The clause graph (CG) has nodes representing clauses and an edge between two clauses whenever they share a negated literal. Each of these graphs corresponds to a constraint graph for the associated CSP; thus, each encodes aspects of the problem's combinatorial structure. For each graph we compute various node degree statistics. For the CG we also compute statistics of weighted clustering coefficients, which measure the extent to which each node belongs to a clique. For each node the *weighted clustering coefficient* is the number of edges among its neighbors (including the node itself) divided by $k(k+1)/2$, where k is the number of neighbors. Including the node when counting edges has an effect of weighting the classical clustering coefficient by the node degree.

The fifth group measures the balance of a formula in several different senses, while the sixth group measures the proximity of the instance to a Horn formula, motivated by the fact that such formulas are an important SAT subclass. The seventh group of features is obtained by solving a linear programming relaxation of an integer program representing the current SAT instance. Denote the formula $C_1 \wedge \cdots \wedge C_n$ and let x_j denote both boolean and LP variables. Define $v(x_j) = x_j$ and $v(\neg x_j) = 1 - x_j$. Then the program is maximize $\sum_{i=1}^{n} \sum_{l \in C_i} v(l)$ subject to $\forall C_i : \sum_{l \in C_i} v(l) \geq 1, \forall x_j : 0 \leq x_j \leq 1$. The objective function prevents the trivial solution where all variables are set to 0.5. The eighth group involves running DPLL "probes." First, we run a DPLL procedure to an exponentially-increasing sequence of depths, measuring the number of unit propagations done at each depths. We also run depth-first random probes by repeatedly instantiating random variables and performing unit propagation until a contradiction is found. The average depth at which a contradiction occurs is an unbiased estimate of the log size of the search space [12]. Our final group of features probes the search space with two stochastic local search algorithms, GSAT and SAPS. We run both algorithms many times, each time continuing the search trajectory until a plateau cannot be escaped within a given number of steps. We then average statistics collected during each run.

3.2 Experimental Setup

Our first dataset contained 20 000 uniform random 3-SAT instances with 400 variables each. To determine the number of clauses in each instance, we determined the clauses-to-variables ratio by drawing a uniform sample from $[3.26, 5.26]$ (*i.e.*, the number of clauses varied between 1 304 and 2 104).[6] Our second dataset contained 20 000 uniform random 3-SAT instances with 400 variables and 1 704 clauses each, corresponding to a fixed clauses-to-variables ratio of 4.26. On each dataset we ran three solvers—`kcnfs`, `oksolver` and `satz`—which performed well on random instances in previous years' SAT competitions. Our experiments were executed on 2.4 GHz Xeon processors, under Linux 2.4.20. Our fixed-ratio experiments took about four CPU-months to complete. In contrast, our variable-ratio dataset took only about one CPU-month, since many instances were generated in the easy region away from the phase transition point. Every solver was allowed to run to completion on every instance.

Each dataset was split into 3 parts—training, test and validation sets—in the ratio $70 : 15 : 15$. All parameter tuning was performed with the validation set; the test set was used only to generate the graphs shown in this paper. We performed machine learning and statistical analysis with the `R` and `Matlab` software packages.

4 Variable-Ratio Random Instances

We had three goals with this distribution. First, we wanted to show that our empirical hardness model training and analysis techniques would be able to sift through all the features provided and "discover" that the clauses-to-variables ratio was important to

[6] This range was chosen symmetrically around the phase transition point, 4.26, to ensure that an approximately equal number of satisfiable and unsatisfiable instances would be obtained.

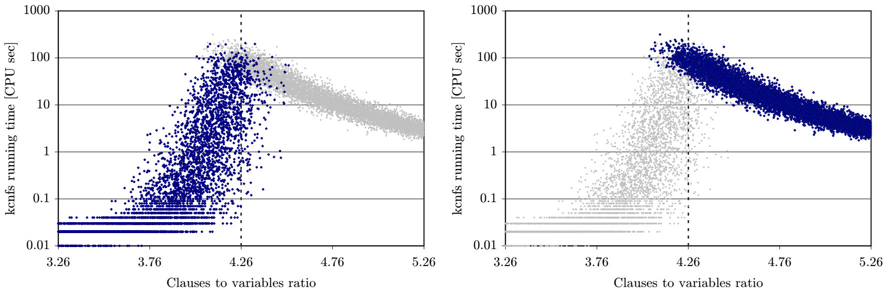

Fig. 2. Runtime of `kcnfs` on variable-ratio satisfiable (left) and unsatisfiable instances (right)

the empirical hardness of instances from this distribution. Second, having included nine features derived from this ratio among our 91 features we wanted to find out which particular function of these features would be most predictive of hardness. Third, we wanted to find out what *other* features, if any, were important in this setting.

We begin by examining the clauses-to-variables ratio, c/v, in more detail. Figure 2 shows `kcnfs` runtime (log scale) vs. c/v, for satisfiable and unsatisfiable instances. First observe that, as expected, there is a clear relationship between runtime and c/v. At the same time, c/v is not a very accurate predictor of hardness by itself: particularly near the phase transition point, there are several orders of magnitude of runtime variance across different instances. This is particularly the case for satisfiable instances around the phase transition; while the variation in runtime between unsatisfiable instances is consistently much smaller. (It may be noted that overall, our dataset is balanced in that it consists of 10 011 satisfiable and 9 989 unsatisfiable instances.)

To build models, we first considered linear, logistic and exponential models in our 91 features, evaluating the models on our validation set. Of these, linear were the worst, and logistic and exponential were similar, with logistic being slightly better. Next, we wanted to consider quadratic models under these same three transformations. However, a full quadratic model would have involved 4 277 features, and given that our training data involved 14 000 different problem instances, training the model would have entailed inverting a matrix of nearly sixty million values. In order to concentrate on the most important quadratic features, we first used our variable importance techniques to identify the best 30-feature subset of our 91 features. We computed the full quadratic expansion of these features, then performed forward selection—the only subset selection technique that worked with such a huge number of features—to keep only the most useful features. We ended up with 360 features, some of which were members of our original set of 91 features and the rest of which were products of these original features. Again, we evaluated linear, logistic and exponential models; all three model types were better with the expanded features, and again logistic models were best. Although the actual RMSE values obtained by three different kinds of models were very close to each other, linear models tended to have much higher prediction bias and many more outliers, especially among easy instances.

Figure 3 (left) shows the performance of our logistic models in this quadratic case for `kcnfs` (evaluated for the first time on our test set). Note that this is a very accurate model: perfect predictions would lie exactly on the line $y = x$, and the vast majority

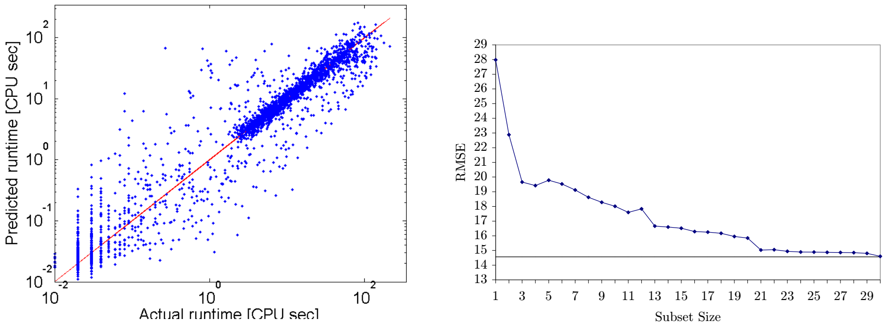

Fig. 3. Actual vs. predicted runtimes for `kcnfs` on variable-ratio instances (left) and RMSE as a function of model size (right).

Variable	Cost of Omission
$\lvert c/v - 4.26\rvert$ [9]	100
$\lvert c/v - 4.26\rvert^2$ [10]	69
$(v/c)^2 \times$ SAPS_BestCoeffVar_Mean [7 $\times$ 90]	53
$\lvert (c/v) - 4.26\rvert \times$ SAPS_BestCoeffVar_Mean [9 $\times$ 90]	33

Table 1. Variable importance in size 4 model for variable-ratio instances.

of points lie on or very close to this line, with no significant bias in the residuals.[7] The plots for `satz` and `oksolver` look very similar; the RMSE values for the `kcnfs`, `satz` and `oksolver` models are 13.16, 24.09, and 81.32 seconds, respectively.

We now turn to the question of which variables were most important to our models. For the remainder of this paper we focus only on our models for `kcnfs`; our results with the other two algorithms are comparable. First, we discuss what it means for our techniques to identify a variable as "important." If a set of variables X is identified as the best subset of a given size, and this subset has a RMSE that is close to the RMSE of the complete model, this indicates that the variables in X are *sufficient* to approximate the performance of the full model—useful information, since it means that we can explain an algorithm's empirical hardness in terms of a small number of features. It must be stressed, however, that this does not amount to an argument that choosing the subset X is *necessary* for good performance in a subset of size k. Because variables are very often correlated, there may be other sets that would achieve similar performance; furthermore, since our subset selection techniques are heuristic, we are not even guaranteed that X is the globally best subset of its size. Thus, we can draw conclusions about the variables that are present in small, well-performing subsets, but we must be very careful in drawing conclusions about the variables that are absent.

Figure 3 (right) shows the validation set RMSE of our best subset of each size. Note that our best four-variable model achieves a root-mean-squared error of 19.42 seconds, while our full 360-feature model had an error of about 14.57 seconds. Table 1 lists the four variables in this model along with their normalized costs of omission. Note that the most important feature (by far) is the linearized version of c/v, which also occurs (in different forms) in the other three features of this model. Hence, our techniques

[7] The banding on very small runtimes in this and other scatterplots is a discretization effect due to the low resolution of the operating system's process timer.

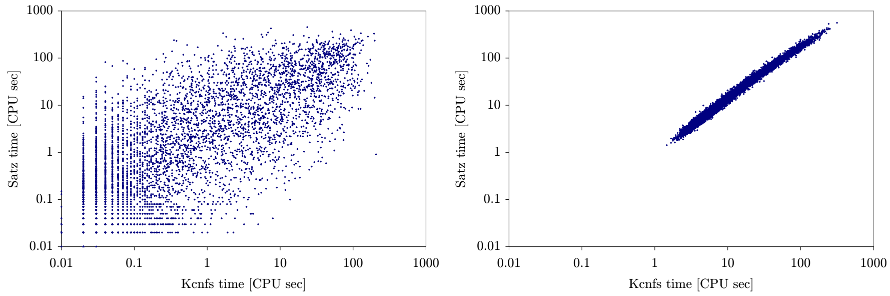

Fig. 4. Runtime correlation between `kcnfs` and `satz` for satisfiable (left) and unsatisfiable (right) variable-ratio instances.

correctly identified the importance of the clauses-to-variables ratio, which satisfies our first goal. In terms of the second goal, these results indicate that the simple absolute distance of the ratio c/v from the critical value 4.26 appears to be the most informative variant of the nine related features we considered.

The third and fourth features in this model satisfy our third goal: we see that c/v variants are not the only useful features in this model. Interestingly, both of these remaining variables are based on a local search probing feature, the coefficient of variation over the number of clauses unsatisfied in local minima found by SAPS, a high-performance local search algorithm for SAT. It may appear somewhat surprising that such a local search probing feature can convey meaningful information about the runtime behavior of a DPLL algorithm. However, notice that deep local minima in the space searched by a local search algorithm correspond to assignments that leave few clauses unsatisfied. Intuitively, such assignments can cause substantial difficulties for DPLL search, where the respective partial assignments may correspond to large subtrees that do not contain any solutions. However, our current understanding of the impact of the features captured by local search probes on DPLL solver performance is rather limited, and further work is needed to fully explain this phenomenon.

While analyzing our variable-ratio models, we discovered that the weighted clause graph clustering coefficient (33) was one of the most important features. In fact, it was the most important feature if we excluded higher-order c/v and v/c features from models. It turns out, that the WCGCC is almost perfectly correlated with v/c, as illustrated in Figure 6 (left). This is particularly interesting as both the clustering coefficient and the connectivity of the constraint graph have been shown to be important statistics in a wide range of combinatorial problems, such as graph coloring and WDP. This correlation provides very nice new structural insight into the clause-to-variables ratio: it shows explicitly how constraint structure changes as the ratio varies. This discovery demonstrates how our empirical hardness methodology can help to gain new understanding of the nature of $\mathcal{NP}$-Hard problems.

The previously mentioned similar performance of our predictive models for `kcnfs`, `satz` and `oksolver` raises the question of whether the underlying reason simply lies in a strong correlation between the respective runtimes. Figure 4 shows the correlation of `kcnfs` runtime vs. `satz` runtime on satisfiable and unsatisfiable instances. Note that there are two qualitatively different patterns in the performance correlation for the two

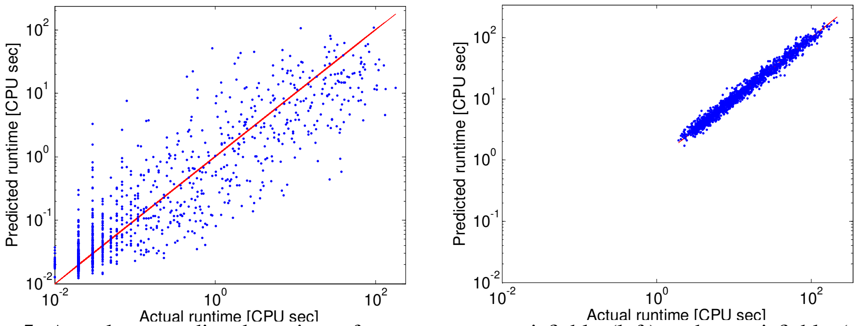

Fig. 5. Actual vs. predicted runtimes for `kcnfs` on satisfiable (left) and unsatisfiable (right) variable-ratio instances.

types of instances: runtimes on UNSAT instances are almost perfectly correlated, while runtimes on SAT instances are almost entirely uncorrelated. We conjecture that this is because proving unsatisfiability of an instance essentially requires exploring the entire search tree, which does not differ substantially between the algorithms, while finding a satisfiable assignment depends much more on each algorithm's different heuristics. We can conclude that the similar model accuracy between the algorithms is due jointly to the correlation between their runtimes on UNSAT instances and to the ability of our features to express each algorithm's runtime profile on both SAT and UNSAT instances.

Motivated by qualitative differences between satisfiable and unsatisfiable instances, we studied the subsets of all satisfiable and all unsatisfiable instances from our dataset separately. Analogously to what we did for the full dataset, we trained a separate predictive model for each of these two subsets. Interestingly, as seen in Figure 5, the predictions for unsatisfiable instances are much better than those for satisfiable instances (RMSE 5.3 vs. 13.4). Furthermore, the 'loss curves', which indicate the best RMSE achieved in dependence of model size (cf. Figure 3), are rather different between the two subsets: For the satisfiable instances, seven features are required to get within 10% of full model accuracy (in terms of RMSE), compared to only three for the unsatisfiable instances. While the seven features in the former model are all local search probe features (namely, in order of decreasing importance, features 68^2, 68×70, 90, 70, 70^2, 90×71 and 71), the three features in the latter are DPLL probe and constraint graph features (namely features 66^2, 66 and 26×27).

It must be noted that excluding all local search probe features (68-91 in Figure 1) in the process of model construction leads to models with only moderately worse performance (RMSE 16.6 instead of 13.4 for satisfiable, 5.5 instead of 5.3 for unsatisfiable, and 17.2 instead of 13.2 for all instances). Interestingly, in such models for satisfiable instances, features based on LP relaxation (features 55–60 in Figure 1) become quite important. Even when excluding all probing and LP features (features 55-91), reasonably accurate models can still be obtained (RMSE 14.7, 8.4, and 17.1 for satisfiable, unsatisfiable, and all instances, respectively); this indicates that combinations of the remaining purely structural features still provide a sufficient basis for accurate runtime predictions on the variable-ratio instance distribution.

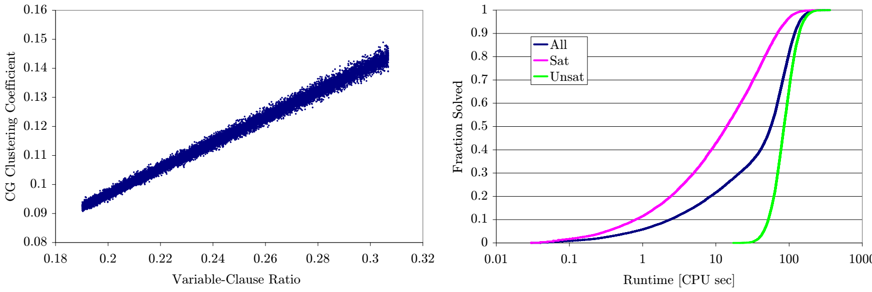

Fig. 6. Left: Correlation between CG weighted clustering coefficient and v/c. Right: Distribution of `kcnfs` runtimes across fixed-ratio instances.

Variable	Cost of Omission
SAPS_BestSolution_Mean^2 [68^2]	100
SAPS_BestSolution_Mean $\times$ Mean_DPLL_Depth [68 $\times$ 66]	74
GSAT_BestSolution_CoeffVar $\times$ Mean_DPLL_Depth [71 $\times$ 66]	21
VCG_CLAUSE_Mean $\times$ GSAT_FirstLMRatio_Mean [17 $\times$ 88]	9

Table 2. Variable importance in size 4 model for fixed-ratio instances.

5 Fixed-Ratio Random Instances

According to a widely held (yet somewhat simplistic) belief, uniform random 3-SAT is easy when far from the phase-transition point, and hard when close to it. In fact, while the first part of this statement is generally true, the second part is not. Figure 6 (right) shows cumulative distributions of the `kcnfs`'s runtime per instance across our second dataset, comprising 20 000 fixed-ratio uniform random-3-SAT instances with 400 variables at $c/v = 4.26$, indicating substantial variation in runtime between instances in the phase transition region. (Similar observations have been made previously for local search algorithms [6].) Random-3-SAT at the phase transition point is one of the most widely used classes of benchmark instances for SAT; in the context of our study of empirical hardness models this instance distribution is particularly interesting since the most important features for predicting instance hardness for the variable-ratio distribution, namely variants of c/v, are kept constant in this case. Hence, it presents the challenge of identifying other features underlying the observed variation in hardness.

We built models in the same way as described in Section 4, except that all variants of c/v are constant and were hence omitted. Again, we achieved the best (validation set) results with logistic models on a (partial) quadratic expansion of the features; Fig. 7 (left) shows the performance of our logistic model for `kcnfs` on test data (RMSE = 35.23); similar results were obtained for `oksolver` and `satz` (RMSE = 220.43 and 60.71, respectively; note that particularly for `oksolver` , the higher RMSE values are partly due to overall higher runtimes). The shape of the scatter plots can be visually misleading: although it appears to be not tight, there are many more points that lie along the diagonal than outliers (this becomes evident when plotting the data on a heat map).

Figure 7 (right) shows the validation set RMSE of the best model we found at each subset size. Here, a 4-variable model obtains RMSE 39.02 on the validation set, which is within 10% on the RMSE of the full model. The variables in the model, along with their costs of omission, are given in Table 2. Note that this model is dominated by local

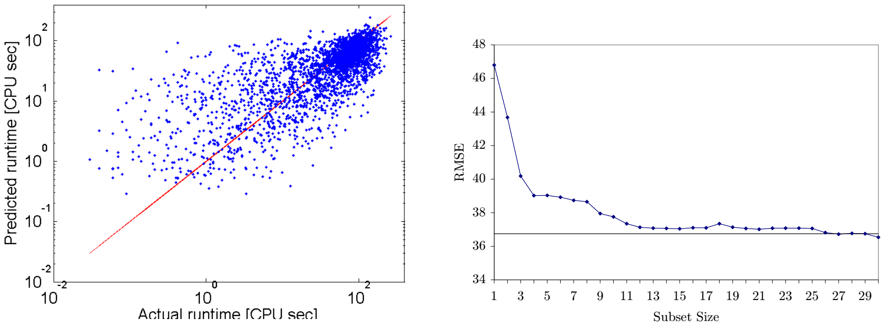

Fig. 7. Actual vs. predicted runtimes for `kcnfs` on fixed-ratio instances (left) and RMSE as a function of model size (right).

search and DPLL probing features, and the most important feature is the deepest local minimum reached on a SAPS trajectory (BestSolution), which intuitively captures the degree to which a given instance has "almost" satisfying assignments.

As for the variable-ratio set, we studied the subsets of all satisfiable and all unsatisfiable instances from our fixed-ratio data set separately and trained separate models for each of these subsets. Analogous to our results for the variable-ratio sets, we found that our model for the former subset gave significantly better predictions than that for the latter (RMSE 15.6 vs. 30.2). Surprisingly, in both cases, only a single feature is required to get within 10% of full model accuracy (in terms of RMSE on the training set): the product of the two SAPS probing features 69 and 82 in the case of satisfiable instances, and the square of DPLL probing feature 66 in the case of unsatisfiable instances.

We also constructed models that do not use local search features and/or probing features and obtained results that are qualitatively the same as those for the variable-ratio data set. Furthermore, we have observed results on the correlation of runtimes between solvers that are analogous to those reported in Section 4.

6 SATzilla and Other Applications of Hardness Models

While so far, we have argued that accurate empirical hardness models are useful because of the insight they give into problem structure, these models also have other applications [9]. For example, it is very easy to combine accurate hardness models with an existing instance generator to create a new generator that makes harder instances, through the use of rejection sampling techniques. Within the next few months, we intend to make available a new generator of harder random 3-SAT formulas. This generator will work by generating an instance from the phase transition region and then rejecting it in inverse proportion to the log time of the minimum of our three algorithms' predicted runtimes.

A second application of hardness models is the construction of algorithm portfolios. It is well known that for SAT different algorithms often perform very differently on the same instances (cf. left side of Figure 4). On distributions for which this sort of uncorrelation holds, selecting among algorithms on a per-instance basis offers the potential for substantial improvements over per-distribution algorithm selection. Empirical hardness models allow us to choose algorithms based on predicted runtimes. Interestingly, fairly inaccurate models often suffice to build good portfolios: if algorithms' performances are close to each other, picking the wrong one is not very costly, while if algorithms' behaviors differ significantly, the discrimination task is relatively easy.

We can offer concrete evidence for the utility of the latter application of hardness models: `SATzilla`, an algorithm portfolio that we built for the 2003 SAT competition. This portfolio consisted of `2clseq`, `eqSatz`, `HeerHugo`, `JeruSat`, `Limmat`, `oksolver`, `Relsat`, `Sato`, `Satz-rand` and `zChaff`. The 2004 version dropped `HeerHugo`, but added `Satzoo`, `kcnfs`, and `BerkMin`.

To construct `SATzilla` we gathered from various public websites a library of about 5 000 SAT instances, for which we computed runtimes and the features described in Section 3.1. We built models using ridge regression. To yield better models, we dropped from our dataset all instances that were solved by all or none of the algorithms, or as a side-effect of feature computation. Upon execution, `SATzilla` begins by running a UBCSAT [17] implementation of `WalkSAT` to filter out easy satisfiable instances. Next, it runs the `Hypre` preprocessor [1] to clean up instances, allowing features to better reflect the problem's "combinatorial core." Third, `SATzilla` computes its features, terminating if any feature (*e.g.*, probing or LP relaxation) solves the problem. Some features can take inordinate amounts of time, particularly with very large inputs. To prevent feature computation from consuming all of our allotted time, certain features run only until a timeout is reached, at which point `SATzilla` gives up on them. Fourth, `SATzilla` evaluates a hardness model for each algorithm. If some of the features have timed out, it uses a different model which does not involve the missing feature. Finally, `SATzilla` executes the algorithm with the best predicted runtime.

`SATzilla` performed very well both in 2003 and 2004. In 2003, it was the only complete solver that did well both on random and on structured instances. It finished second and third in different categories, loosing only to new-generation solvers. In 2004, it was leading among complete solvers in the first round, but didn't advance to the final round due to complicated new competition rules.

7 Conclusion and Future Work

We have shown that empirical hardness models are a valuable tool for the study of the empirical behavior of complex algorithms such as SAT solvers. We were able to build accurate models of runtime on test distributions of fixed- and variable-ratio uniform random-3-SAT instances. On the variable-ratio dataset, our techniques were able to automatically "discover" the importance of the c/v ratio. Analysis in this case provided insight into the structural variations in uniform random 3-SAT formulas at the phase transition point that correlate with the dramatic variation in empirical hardness. Finally, we argued that our empirical hardness models offer practical benefit in less well-controlled domains by presenting `SATzilla`, our algorithm portfolio for SAT.

The results presented suggest a number of avenues for future research. One issue that clearly deserves further investigation is the degree to which our methodology can be used to predict and explain the performance of stochastic local search algorithms for SAT, which have recently been shown to outperform the best systematic solvers on various classes of random SAT instances. Another obvious and very relevant direction is the extension of our work to more structured types of SAT instances. Also, our results for the satisfiable and unsatisfiable subsets suggest that hierarchical models could give even better results. Such models may use some features to predict the satisfiability (or more generally, the type) of a given instance, and a subsidiary model for predicting the runtime. And finally, we believe that by studying in more detail *how* some of the features identified through the use of predictive statistical models cause instances to be easy or hard for certain types of algorithms, our understanding of how to solve SAT most efficiently will be further advanced.

References

1. Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proc. SAT-2003*, pages 341–355, 2003.
2. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Proc. IJCAI-1991*, pages 331–337, 1991.
3. 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.
4. H. H. Hoos and T. Stützle. *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann, 2004 (to appear).
5. H.H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proc. IJCAI-99*, pages 296–302. Morgan Kaufmann, 1999.
6. H.H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for sat. *Artificial Intelligence*, 112:213–232, 1999.
7. E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and M. Chickering. A Bayesian approach to tackling hard computational problems. In *Proc. UAI-2001*, pages 235–244, 2001.
8. Phokion Kolaitis. Constraint satisfaction, databases and logic. In *Proc. IJCAI-2003*, pages 1587–1595, 2003.
9. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. Boosting as a metaphor for algorithm design. In *Proc. CP-2003*, pages 899–903, 2003.
10. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proc. IJCAI-2003*, pages 1542–1542, 2003.
11. K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proc. CP-2002*, pages 556–572, 2002.
12. L. Lobjois and M. Lemaître. Branch and bound algorithm selection by performance prediction. In *Proc. AAAI-1998*, pages 353–358, 1998.
13. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.
14. Y. Ruan, E. Horvitz, and H. Kautz. Restart policies with dependence among runs: A dynamic programming approach. In *Proc CP2-2002*, pages 573–586, 2002.
15. B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.
16. R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal Statist. Soc B*, 58(1):267–288, 1996.
17. D. Tompkins and H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Proc. SAT-2004*, pages 37–46, 2004.
18. R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proc. IJCAI-2003*, pages 1173–1178, 2003.