Satisfiability as a Classification Problem*

David Devlin and Barry O'Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{d.devlin|b.osullivan}@4c.ucc.ie

Abstract. Given a Boolean formula, the classic satisfiability problem is to decide whether there is a truth assignment to its variables such that the formula evaluates to true. The satisfiability problem was the first decision problem proven to be NP-Complete. Therefore, it is very unlikely that there exists an algorithm for solving the satisfiability problem that has good worst-case performance. However, the satisfiability problem is ubiquitous in artificial intelligence. In this paper, we view the satisfiability problem as a classification task. Based on easy to compute structural features of instances of large satisfiability problems we use a variety of standard classifier learners to classify previously unseen instances of the satisfiability problem as either satisfiable or unsatisfiable. We show that standard learning techniques can very reliably perform this task, with accuracy in excess of 99% for hard 3-SAT problems, and usually in excess of 90% for large industrial benchmarks. These results are surprising, and suggest that machine learning techniques can be very effective at revealing the significant structural characteristics that are important in satisfiability testing.

1 Introduction

The satisfiability problem (SAT) is, informally, defined as: given a Boolean formula, decide whether there is a truth assignment to each of its variables such that the formula evaluates to *true*. For example, consider the Boolean function AND over variables A and B:

$$\phi_{\text{AND}}(A, B) = A.B$$

This formula evaluates to true if both A and B are assigned *true*. We say that this formula is, therefore, satisfiable.

SAT is important from both theoretical and practical perspectives. SAT was the first problem proved to be NP-Complete. The proof, Cook's Theorem, was published in a 1971 paper by Stephen A. Cook [3]. That was a breakthrough result in computational complexity. Leonid Levin independently discovered the proof of SAT's NP-Completeness, although he did not formally publish it until 1973. For this reason the proof is usually referred to as the Cook-Levin Theorem.

* This work was supported by Science Foundation Ireland (Grant No. 05/IN/I886).

The following year Richard Karp proved that a further twenty-one intractable problems were NP-Complete [5]. These key results provided the basis for the theory of NP-Completeness that is so important in computer science. However, SAT is not just of theoretical interest. SAT problems occur in a variety of domains such as hardware verification, security protocol analysis, theorem proving, scheduling problems, routing, planning, digital circuit design and artificial intelligence.

Deciding whether a SAT problem is satisfiable or not is usually performed by either systematic search, based on backtracking or resolution, or local search. Because the general problem is NP-Complete, systematic search algorithms have exponential worst-case running times, which has the effect of limiting the scalability of systematic search methods. If a SAT problem is unsatisfiable, local search algorithms, while scalable, cannot prove that to be the case.

Recent advances in machine learning have provided the artificial intelligence community with powerful techniques for classification problems. SAT can be seen as a classification problem: given a Boolean formulae we are asked to classify it as either satisfiable (*true*) or unsatisfiable (*false*).

The *objective* of the work reported in this paper is to apply a suite of standard classification algorithms to the problem of deciding SAT. Our *methodology* involved considering large hard SAT instances from the International SAT Competition[1] and Satlib[2] and implementations of classification algorithms through the data-mining system WEKA[3]. Our *results* show that classification algorithms perform extremely well on deciding SAT. We argue that an approach such as ours can be useful for informing which techniques should be used to solve large complex SAT problems. Also, this work represents the beginning of a research agenda that will study how what structural features of SAT problems can be exploited during search.

2 Background

The objective of the work reported in this paper was to evaluate the performance of standard classification algorithms from the field of machine learning on the task of deciding whether a satisfiability problem has as a solution or not. We, therefore, provide an overview of relevant background concepts in the areas of satisfiability and machine learning, in particular, classification algorithms.

2.1 The Satisfiability Problem

The satisfiability (SAT) problem is defined as follows: given a propositional formula, $\phi = f(x_1, \ldots, x_n)$, over a set of variables $x_1, \ldots, x_n$, decide whether or not there exists a truth assignment to the variables such that ϕ evaluates to true.

[1] `http://www.satcompetition.org/`
[2] `http://www.satlib.org/`
[3] `http://www.cs.waikato.ac.nz/ml/weka/`

SAT problem instances are usually expressed in a standard form, called conjunctive normal form (CNF). A SAT problem in this way is expressed as a conjunction of *clauses*, where each clause is a disjunction of *literals*; a literal is either a variable or its negation. The following SAT formula is in CNF:

$$\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_4) \wedge (x_2 \vee \neg x_3).$$

This formulae comprises three clauses: the first a disjunction of literals x_1, x_3 and $\neg x_4$; the second involves a single literal x_4; the third is a disjunction of literals x_2 and $\neg x_3$. The SAT problem ϕ is satisfiable because we can set x_1, x_2 and x_4 to *true*, satisfying the first, third and second clauses, respectively.

Sometimes one refers to instances of k-SAT, which simply means we restrict ourselves to formulae in which each clauses involves at most k literals. All SAT instances with $k \geq 3$ can be transformed to 3-SAT in time polynomial in the size of the k-SAT instance. SAT is NP-Complete for $k = 3$ and higher. SAT is polynomial for $k < 3$. Of course, other polynomial classes of SAT are known, for example Horn-SAT, in which each clause corresponds to an implication, i.e. they are Horn clauses. Horn-SAT is an extremely important practical tractable class of SAT since a large proportion of clauses in real-world SAT instances are Horn.

An important phenomenon that is important to understand in SAT is that while one might believe that as the number of clauses increases, the probability of a formula being satisfiable smoothly reduces from 1 to 0, the transition is actually abrupt. For example, for randomly generated 3-SAT problems the transition from under-constrained problems with very high probability of satisfiability to over-constrained problems with very low probability of satisfiability is extremely abrupt, and occurs when the ratio of the number of clauses to the number of variables is approximately 4.26. This abrupt change is called the *phase transition* [2]. The hardest SAT instances are those that occur close to the phase transition. This phenomenon is illustrated in Figure 1.

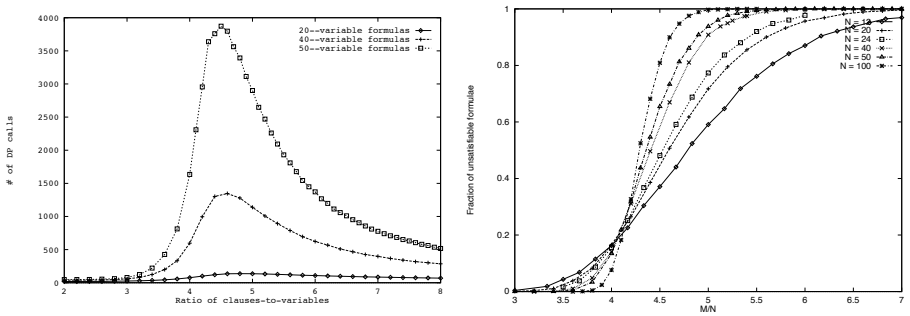

Fig. 1. The phase transition phenomenon in random 3-SAT. Left: Computational hardness peaks at a clause-to-variable ratio of approximately 4.26. Right: Problems change from being mostly satisfiable to mostly unsatisfiable. The transitions sharpen as the number of variables grows.

2.2 Machine Learning and Classification

Machine learning "is concerned with the question of how to construct computer programs that automatically improve with experience". It is a broad field that uses concepts from computer science, mathematics, statistics, information theory, complexity theory, biology and cognitive science [6]. Machine learning can be applied to well-defined problems, where there is both a source of training examples and one or more metrics for measuring performance.

In this paper we are particularly interested in classification tasks. A classifier is a function that maps an *instance* with one or more discrete or continuous features to one of a finite number of classes [6]. A classifier is trained on a set of instances whose class is already known, with the intention that the classifier can transfer its training experiences to the task of classifying new instances. We consider a *supervised learning* setting, where the instances in the training set have been manually labeled with the correct class. In this paper we use the following classifiers: random forests of decision trees, decision trees, multilayer perceptron, nearest neighbour, and naive Bayes. A summary of each of these algorithms is beyond the scope of this paper, largely due to space limitations, but a detailed survey can be found in any standard machine learning textbooks, such as [6].

3 Useful Features of SAT Instances

In this paper we employed the same set of SAT instance features as those used in `SATzilla`[4]. `SATzilla` is a successful algorithm portfolio for SAT, i.e. a system that uses machine learning techniques to select the faster SAT solver for a given problem instance. That system uses a total of 48 features, summarised in Figure 2 [8].

These features can be summarised under nine different categories: problem size features; variable-clause graph features; variable graph features; balance features; proximity to Horn formula; DPLL probing features; and local search probing features. The first category of features are self explanatory, and simply relate to the number of variables and clauses in the SAT instance. The next two categories relate to two different graph representations of a SAT instance. The variable-clause graph 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 has a node for each variable and an edge between variables that occur together in at least one clause. The balance features are self explanatory and relate, primarily, to the distribution of positive and negative literals within the SAT instance. The category related to the proximity to a Horn formula captures how close the SAT instance is to an important polynomial class of SAT that can be solved using the standard inference method used in all systematic SAT solvers (i.e. unit propagation). The DPLL probing features are related to statistics that a standard systematic search algorithm gathers while

[4] `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

Problem Size Features:
1. **Number of clauses**: denoted c
2. **Number of variables:** denoted v
3. **Ratio:** c/v

Variable-Clause Graph Features:
4-8. **Variable nodes degree statistics:** mean, variation coefficient, min, max and entropy.
9-13. **Clause nodes degree statistics:** mean, variation coefficient, min, max and entropy.

Variable Graph Features:
14-17. **Nodes degree statistics:** mean, variation coefficient, min and max.

Balance Features:
18-20. **Ratio of positive and negative literals in each clause:** mean, variation coefficient and entropy.
21-25. **Ratio of positive and negative occurrences of each variable:** mean, variation coefficient, min, max and entropy.
26-27. **Fraction of binary and ternary clauses**

Proximity to Horn Formula:
28. **Fraction of Horn clauses**
29-33. **Number of occurrences in a Horn clause for each variable:** mean, variation coefficient, min, max and entropy.

DPLL Probing Features:
34-38. **Number of unit propagations:** computed at depths 1, 4, 16, 64 and 256.
39-40. **Search space size estimate:** mean depth to contradiction, estimate of the log of number of nodes.

Local Search Probing Features:
41-44. **Number of steps to the best local minimum in a run:** mean, median, 10th and 90th percentiles for SAPS.
45. **Average improvement to best in a run:** mean improvement per step to best solution for SAPS.
46-47. **Fraction of improvement due to first local minimum:** mean for SAPS and GSAT.
48. **Coefficient of variation of the number of unsatisfied clauses in each local minimum:** mean over all runs for SAPS.

Fig. 2. A summary of the features of the SAT instances in our data-set that were used to build our classifiers. These are the same features used by `SATzilla` to build its runtime prediction system (Figure taken from [8]).

testing the difficulty of the instance. The local search features are the non-systematic analogue of the latter category.

4 Building Data-sets for SAT

We composed four different data-sets of SAT instances. These were Crafted, Industrial, Random 3-SAT and Random. This is based on the categories used in the International SAT Competition, but with an additional category that contains only random 3-SAT instances, a subset of the Random category.

All instances were gathered from one of three sources: the International SAT Competition problem-sets, Miroslav Velev's SAT benchmark suite [7] and Satisfiability Library (Satlib)[5]. The instances we gathered are considered hard by the satisfiability world, demonstrated by the fact that they were used in competitions or as benchmarks. Many of these instances were large, with their specifications being circulated in text files several hundred megabytes in size. All instances were in the standard DIMACS CNF format, a standardised file format for SAT instances. An advantage of using standard competition instances and benchmarks is that they are pre-classified as either satisfiable or unsatisfiable, thus saving us the effort of determining this by search.

[5] `http://www.satlib.org/`

4.1 Descriptions of the Data-sets

We summarise the details of our data-sets below, and in Table 1.

Table 1. Number of SAT instances in each category of our data-set.

Category	SAT	UNSAT	Total	Source
Crafted	169	246	415	SAT Competitions 2004, 2005, 2007
Industrial	117	84	201	SAT Competition, Miroslav Velev's Benchmarks
Random 3-SAT	2073	2699	4772	SATLib
Random k-SAT	3148	2699	5587	SATLib

Crafted. The crafted instances were sourced from the published SAT Competition benchmarks, specifically from the 2004, 2005 and 2007 competitions. In the SAT Competition this category contains all hard instances that are neither random nor industrial. Some examples of instances that were included in this category are [1]: `LinvRinv` benchmarks (by Armin Biere); 3-Regular Graphs (by Matti Järvisalo); counting, ordering and pebbling problems (by Ashish Sabharwal); social golfer problems (by Inês Lynce); algebraic benchmarks (by Volker Sorge); Eulerian graphs (by Klas Markström). Many of these instances have been specifically designed to be difficult to solve. There is usually a prize at the International SAT Competition for the smallest crafted instance that cannot be solved by any competing solver.

Industrial. The industrial instances were sourced from both the SAT Competition 2007 benchmarks, and from Miroslav Velev's SAT Benchmarks. These instances consist of formal hardware verification problems formulated as SAT. Some of these instances are exceptionally large – over 400MB file size specification for a single instance in one case.

Random 3-SAT and Random k-SAT. These instances were sourced from Satlib. They can be divided into Uniform Random 3-SAT, and other random problems such as [4]: graph colouring, planning, quasigroup problems, and bounded model checking problems. The uniform random 3-SAT instances that make up our 3-SAT data-set are taken from the phase transition region (Figure 1), and are generated using an unforced filtered method. In forced generation, the satisfiability of the instance generated is guaranteed by the way it constructed. It has been found that such instances are solved more easily than unforced instances, whose satisfiability was not guaranteed. Following unforced generation it is necessary to use a solver to determine whether the generated instance is satisfiable or unsatisfiable.

As Satlib contained only satisfiable instances of other random problems, it was necessary to mix these instances with the Uniform Random 3-SAT instances to build the final training set. A training set should contain a relatively balanced

distribution of satisfiable and unsatisfiable instances, and cannot be made from instances that are all in the same class. It would be preferable to have unsatisfiable instances of graph colouring, planning, quasigroup and bounded model checking problems in this data-set.

4.2 Building the Feature-based Data-sets

Based on the SAT instances we gathered, a feature-based description of each instance must be built based on the features discussed earlier. We modified `SATzilla` to extract features and populate an SQL database with the feature description of each instance. As well as the features mentioned earlier, we recorded the time taken to extract each group of features. It is not possible to measure the time taken to extract individual features, as several can be extracted by performing a single operation. For example, reading the instance description provides both the number of clauses, number of variables and the clause to variable ratio. Also, building a variable-clause graph is useful for up to 30 distinct features, so the effort for this task is not easily aportionable to each of those features.

In Table 2 we present the times for extracting the full set of features from each instance. It is worth noting that even the task of computing the features we need from each instance can take a considerable amount of time, specifically for the large industrial category, and, to a lesser extent, the crafted category. It is also worth noting that the difficulty of extracting features from SAT versus UNSAT instances can also be asymmetric. For example, extracting features from UNSAT industrial instances takes much longer than from the SAT instances in that class.

Table 2. Building the feature descriptions – feature extraction times in seconds.

	Crafted		Industrial		Ran.3SAT		Random	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
Minimum	1.03	1.99	3.33	2.94	0.98	1.00	1.00	–
Average	4.43	3.92	87.04	214.35	1.23	2.31	1.84	–
Maximum	59.26	27.87	912.82	3660.64	3.04	3.06	3.06	–

5 Experiment – Classification of SAT Instances

We present the results of our main experiment. The objective of the experiment was to evaluate the accuracy with which standard classification algorithms could determine whether a SAT instance was satisfiable or not. Our results show that the performance of classification algorithms is very convincing, with classification accuracies typically in excess of 90%, and sometimes in excess of 99%.

The classifiers we considered here were: random forest of decision trees, decision tree, multilayer perceptron, nearest neighbour, and naive Bayes. We used

implementations of these algorithms from the Waikato Environment for Knowledge Analysis (WEKA) machine learning library, a machine learning tool released as free open-source software by the University of Waikato, New Zealand under the GNU General Public Licence. We used WEKA version 3.4.12.

We randomly shuffled the instances (in fact, the feature-based description of each instance) in our dataset to ensure a uniform distribution of instances. The specific settings of WEKA used are as follows (default setting were used if not stated otherwise):

- Random Forest – `weka.classifiers.trees.RandomForest`
  20 trees, 10 attributes/tree;
- Best-First Decision Tree – `weka.classifiers.trees.BFTree`
  All attributes, Post-Pruned;
- Multilayer Perceptron – `weka.classifiers.functions.MultilayerPerceptron`
  500 epochs, 0.3 learning rate;
- 1-Nearest Neighbour with Generalization – `weka.classifiers.rules.NNge`;
- Naive Bayes – `weka.classifiers.bayes.NaiveBayes`.

Each classifier was trained on three different subsets of our features to evaluate the improvement in classification accuracy of the resultant classifiers. The subsets we considered were:

1. The base set of features (Features 1–33 from Figure 2), i.e. excluding search related features (Base);
2. All `SATzilla` features referred to in Figure 2, but excluding feature extraction time (All);
3. All `SATzilla` features including those related to feature extraction time (+t).

To evaluate each classifier we used 10-fold cross validation. The k-fold cross validation procedure partitions the data-set into k equally sized disjoint subsets, $T_1, T_2, \ldots, T_k$. For i from 1 to k, T_i is used as the validation set, and the classifier is trained on all other subsets. The accuracy when validating with T_i is recorded for each i, and the final measure of accuracy is the mean of these values [6]. The accuracy of the classifier is defined as the number of correctly classified instances divided by the total number of instances, by class and overall. The results of these experiments are presented in Table 3.

The classification accuracies we obtain are extremely high. In every problem category, except *Crafted* we have at least one classifier that has in excess of 90% accuracy over all satisfiability classes. Random forests perform consistently well across all categories and classes, followed closely by decision trees. Multi-layer perceptrons do not tend to dominate, but do have competitive results in the *Industrial* category. 1-nearest neighbour has, essentially, the best performance for satisfiable instances in the *Industrial* category, while naive Bayes is best for unsatisfiable *Crafted* instances and satisfiable *Random* instances. However, in the latter category, because naive Bayes performs poorly on unsatisfiable instances, it is questionable whether this is a useful classifier in this context.

Table 3. Detailed accuracy results for all classifiers over all problem categories. In the case of the nearest neighbour classifier WEKA failed silently on the random instances (a bug reported has been submitted to the developers of WEKA). These are marked with a '–'. The best performing classifier for each problem category and each satisfiability class is marked in bold.

		Crafted			Industrial			Random 3SAT			Random		
Classifier	Class	Base	All	+t	Base	All	+t	Base	All	+t	Base	All	+t
Forest	SAT	78.9	82.5	81.1	93.3	94.1	**94.9**	**98.2**	**99.4**	**99.8**	93.2	96.2	**99.2**
	UNSAT	81.4	83.9	84.4	92.7	92.8	92.9	96.3	**97.2**	99.3	**90.7**	**94.7**	**97.9**
	ALL	**80.5**	83.4	83.1	93.0	**93.5**	94.0	97.1	**98.1**	99.5	92.0	**95.5**	**98.6**
DT	SAT	**82.2**	**84.5**	**83.4**	87.8	89.7	93.3	98.0	97.3	98.0	96.0	95.3	98.5
	UNSAT	78.0	83.5	85.3	**97.1**	**94.7**	93.8	**96.6**	96.8	**99.5**	88.6	93.4	97.5
	ALL	79.3	**83.9**	**84.6**	91.0	91.5	93.5	**97.2**	97.1	**99.6**	**92.3**	94.4	98.0
MLP	SAT	71.8	72.4	71.5	92.6	92.5	95.0	88.4	93.9	88.4	90.9	92.1	**99.2**
	UNSAT	79.4	81.2	79.6	94.9	92.6	**96.3**	90.8	92.4	99.4	82.7	86.7	97.8
	ALL	76.4	77.6	76.4	93.5	92.5	**95.6**	89.8	93.0	99.4	86.8	89.5	98.5
1-NN	SAT	74.7	79.1	78.1	**95.7**	**94.7**	94.6	–	–	–	–	–	–
	UNSAT	80.2	81.7	80.7	94.0	88.6	87.6	–	–	–	–	–	–
	ALL	78.1	80.7	79.8	**95.0**	92.0	91.5	–	–	–	–	–	–
Bayes	SAT	58.6	65.3	69.0	80.2	86.5	87.3	64.2	64.3	87.6	**99.9**	**99.1**	**99.2**
	UNSAT	**84.8**	**85.9**	**85.6**	75.0	76.7	76.9	94.1	93.8	96.9	57.6	56.5	91.7
	ALL	69.4	75.2	69.4	78.1	82.1	82.6	74.9	74.9	92.4	66.0	64.4	95.4

In Table 4 we summarise the best performing classifiers in each category/class combination. It is clear that random forest is very versatile, performing well in almost every region of the data-set.

These results are very convincing. They show that a simple classification approach to SAT can give very good results. While one needs to know with certainty whether a SAT instance is satisfiable or not, having an accurate classifier, while not perfect, is useful in practice. Firstly, performing classification during search can be very helpful in guiding heuristics decisions. For example, if one wishes to decide whether to assign a variable *true* or *false* one would prefer to make the assignment that we believe to lead to a satisfiable instance. Of course, this is only practical if the features associated with the current state of search can be quickly computed. Secondly, before search, one may wish to determine whether a SAT instance should be solved using local search or systematic search. If an instance is likely to be unsatisfiable, it might be futile to use a local search approach, since local search will never be able to prove unsatisfiability. Thirdly, for population-based algorithms, one can use the classifier to generate a set of assignments to variables that are likely to be solutions/non-solutions to a SAT instance. This can be very useful in approaches such as genetic algorithm or solution-guided approaches to satisfiability testing.

Table 4. The best classifier(s) for each category/class combination.

Class	Crafted	Industrial	Random 3SAT	Random
SAT	Decision Tree	Random Forest Nearest Neighbour	Random Forest	Naive Bayes
UNSAT	Naive Bayes	Decision Tree Multi-layer Per.	Random Forest Decision Tree	Random Forest
ALL	Random Forest Decision Trees	Random Forest Multi-layer Per. Nearest Neighbour	Random Forest Decision Tree	Random Forest Decision Tree

6 Conclusions

In this paper, we viewed the SAT problem as a classification task. Based on structural features of instances of large satisfiability problems we built and evaluated a suite of classifiers on challenging problems from a variety of problem categories, including hand-crafted, industrial and random settings. We showed that standard learning techniques can very reliably perform this task, with accuracy in excess of 99% for hard 3-SAT problems, and usually in excess of 90% for large industrial benchmarks. We plan to explore the utility of these results in improving the performance of SAT solvers.

References

1. Daniel Le Berre and Laurent Simon. Preface to the special volume on the sat 2005 competitions and evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, Volume 2, 2005.
2. Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.
3. Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.
4. Holger H. Hoos and Thomas Stützle. Satlib: An online resource for research on sat. pages 283–292.
5. Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. 1972.
6. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
7. Miroslav N. Velev. Exploiting signal unobservability for efficient translation to cnf in formal verification of microprocessors. In *Proc of DATE '04*, page 10266. IEEE Computer Society, 2004.
8. Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla-07: The design and analysis of an algorithm portfolio for sat. In *Principles and Practice of Constraint Programming (CP-07)*, Lecture Notes in Computer Science 4741, pages 712–727. Springer Berlin, 2007.