Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2009-039 August 26, 2009

AvatarSAT: AN AUTO-TUNING BOOLEAN SAT SOLVER

RISHABH SINGH, JOSEPH P. NEAR, VIJAY GANESH, and MARTIN RINARD

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

AvatarSAT: An Auto-tuning Boolean SAT Solver

Rishabh Singh, Joseph P. Near, Vijay Ganesh, and Martin Rinard

Massachusetts Institute of Technology
{rishabh, jnear, vganesh, rinard}@csail.mit.edu

Abstract. We present AVATARSAT, a SAT solver that uses machine-learning classifiers to automatically tune the heuristics of an off-the-shelf SAT solver on a per-instance basis. The classifiers use features of both the input and conflict clauses to select parameter settings for the solver's tunable heuristics. On a randomly selected set of SAT problems chosen from the 2007 and 2008 SAT competitions, AVATARSAT is, on average, over two times faster than MINISAT based on the geometric mean speedup measure and 50% faster based on the arithmetic mean speedup measure. Moreover, AVATARSAT is hundreds to thousands of times faster than MINISAT on many hard SAT instances and is never more than twenty times slower than MINISAT on any SAT instance.

1 Introduction

We present AVATARSAT, an auto-tuning SAT solver that uses machine-learning methods to automatically tune an off-the-shelf SAT solver on a per-instance basis. AVATARSAT is designed to classify each SAT instance based on its syntactic features so as to find the corresponding optimal parameter setting for the heuristics in the underlying SAT solver (for example, dynamic variable selection [11] and restarts [5]).

AVATARSAT is built on top of MINISAT version 2.0, a state of the art SAT solver [5]. Based on its performance in recent SAT competitions [1], MINISAT can reasonably claim to be one of the fastest open-source SAT solvers.

We compared AVATARSAT and MINISAT on a randomly selected set of SAT problems chosen from the 2007 and 2008 SAT competitions [1]. AVATARSAT is, on the average, more than two times faster per problem than MINISAT based on the geometric mean speedup measure and 50% faster per problem based on the arithmetic mean speedup measure. [1]

On many hard SAT instances AVATARSAT is hundreds to thousands of times faster than MINISAT, and is never more than twenty times slower than MINISAT on any SAT instance.

Availibility: AVATARSAT's source code, experimental data, and results are available at `http://people.csail.mit.edu/vganesh/avatarsat.html`.

[1] The geometric mean speedup is the geometric mean of the speedups for the individual tests. The arithmetic mean speedup is the ratio of the total time taken by MINISAT divided by the total time taken by AVATARSAT over all tests.

Solver	Instances	Solved	Time-outs	Total Time (in seconds)	Geometric Speedup	Arithmetic Speedup
AVATARSAT	75	53	22	242,090.48	1.51X	2.23X
MINISAT 2.0	75	52	23	366,353.35	-	-

Fig. 1. Number of problems solved and time taken by AVATARSAT and MINISAT.

Contributions:

- **AVATARSAT:** AVATARSAT is a modified version of MINISAT 2.0 that uses machine learning classifiers to choose parameter settings for the tunable heuristics that control several aspects of MINISAT's search algorithm.
- **Course Correction During Search:** A key novelty in AVATARSAT is the course correction step. Modern SAT solvers accumulate conflict clauses and drop input clauses during their search, which can change the structure of the problem considerably. The optimal parameter settings for this new problem may be significantly different from those of the original input problem.
  AVATARSAT therefore first selects a set of parameters for MINISAT to use during the initial part of the search. When the number of new clauses accumulated during this part of the search crosses a threshold, the course correction step examines the new clauses to select a new set of parameters for MINISAT to use during the remaining part of the search. In this way AVATARSAT dynamically adapts the parameter settings to the potentially changing characteristics of the SAT problem.
- **Use of Support-Vector Machines:** AVATARSAT uses multiclass support-vector machines (SVM) [4], a supervised machine-learning technique, to learn a function from the features of SAT instances to discretized parameter settings.
- **Experimental Results:** We present experimental results comparing the performance of AVATARSAT and MINISAT on 75 randomly selected SAT instances (both crafted and industrial) from the SAT 2007 and 2008 competitions [1].

2 The Architecture of AVATARSAT

AVATARSAT is a modified version of MINISAT that relies on two machine-learning classifiers to automatically tune its heuristic search. The architecture of AVATARSAT is summarized in Figure 2. The *preprocessing classifier* is invoked on the input SAT instance to generate initial parameter settings for the solver's heuristics, while the *course correction classifier* is invoked on both the original set of input clauses and the conflict clauses generated during the search in an attempt to adapt the heuristic search to the changing problem instance. Both classifiers are built using the well-known LIBSVM library [3]. The solver itself works as follows:

- **Initial Classification:** AVATARSAT computes features of the input SAT instance. The preprocessing classifier uses these features to select initial parameters for MINISAT's heuristics.

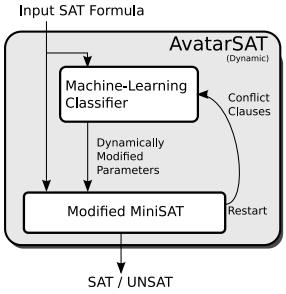

Fig. 2. AVATARSAT Architecture

AVATARSAT Speedup Over MINISAT 2.0	Number of Tests	AVATARSAT Avg. Time (Seconds)	MINISAT 2.0 Avg. Time (Seconds)
1000–50000x	2	3.41	43,200.00
100–1000x	3	57.50	28,850.61
10–100x	3	109.15	4,588.56
1.5–10x	12	1,106.8	3,707.90
1/10–1/1.5x	9	11,636.95	2,941.76
1/20–1/10x	1	4,332.67	260.18

Fig. 3. Number of SAT instances in which AVATARSAT exhibits speedups over MINISAT; AVATARSAT never slows down below $1/20$x of MINISAT. The timeout was 12 hours.

- **Heuristic Search:** AVATARSAT invokes the modified MINISAT with the selected parameter settings.
- **Course Correction:** When the conflict clauses accumulated during the solver's search reach a pre-determined fraction of the input clauses (80%), AVATARSAT invokes the course correction classifier on the input and conflict clauses to select new search parameters for the remainder of the search.

Machine Learning Technique Used: AVATARSAT uses the multi-class *support vector machine* (SVM) [4] supervised machine-learning technique. Supervised machine-learning techniques attempt to learn an unknown function based on a *training set* consisting of input vectors and the corresponding outputs. The SVM technique trains a classifier that maps features of SAT instances to one of a finite set of classes, where each class corresponds to a different parameter configuration. Since the generation of training data requires discretization of parameter values, a multi-class SVM is a natural choice for classification.

The SAT instance features used in training are designed to characterize the corresponding SAT problem instance as closely as possible. They are also designed to be of reasonable length and to be efficient to compute. Our feature vector consists of 58 different log-normalized features of the input SAT instance (e.g., clause/var ratio, var/clause ratio, number of variables, clauses etc. In addition to novel features such as *clause-weighted* positive and negative literal occurrences for distinguishing shuffled SAT instances, we also use some of the features in [12]).

MiniSAT Heuristics Chosen for Automatic Tuning: The SAT solver parameters that are dynamically tuned in AVATARSAT are the *variable decay* parameter of the VSIDS heuristic [11], and *restart increments* [5]. Of the ten tunable parameters in MINISAT, we selected these two because they have been empirically observed to have the most significant influence on solver performance [2].

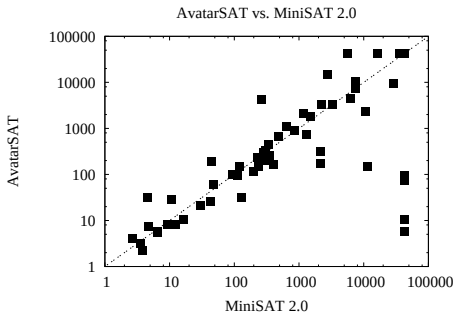

Fig. 4. Log-scale scatterplot of running times (in seconds) of AVATARSAT vs. MINISAT. Each point represents at least one test case; the timeout was set at 12 hours.

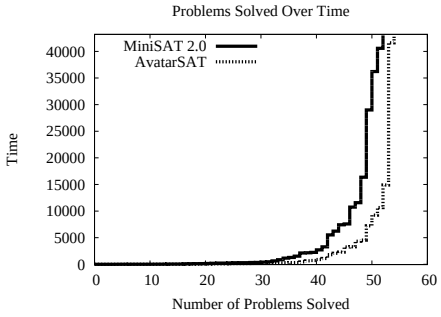

Fig. 5. Number of problems solved by AVATARSAT and MINISAT over time. In our experiments, AVATARSAT solves more problems than MINISAT for any given amount of time.

3 Results

We discuss the experimental setup, guidelines used to conduct the experiments, training setup of the machine-learning classifiers, and results of the comparison of AVATARSAT with MINISAT.

Experimental Setup: All experiments for comparing AVATARSAT with MINISAT, and training of the machine-learning classifiers were conducted on a cluster of 75 Linux machines each with 1.8GHz Intel Xeon processor, 4MB Cache, and 2 GB RAM. To train the preprocessing classifier, we randomly split the SAT 2007 and 2008 competition benchmarks [1] into disjoint training and test sets; the training set contained 177 instances, while the test set contained 75. To train the course correction classifier, we dumped 391 internal conflict clause instances generated during the execution of MINISAT on the instances used to train the preprocessing classifier. The test set was not involved in training either classifier; no distinctions between industrial and crafted or satisfiable and unsatisfiable examples were made.

Experimental Guidelines Used: We followed the guidelines presented by Zarpas [15] for SAT solver benchmarking experiments. In particular, the timeouts for testing the solvers were set relatively high at 12 hours (43,200 seconds). Low timeouts are used in SAT competitions for logistical reasons. However, in real applications solvers may be given much higher timeouts. Furthermore, higher timeouts help differentiate one solver from another on harder examples. In our speedup computation, we excluded those test cases for which both MINISAT and AVATARSAT timed out.

Training The Classifiers: The first step in the training process is collection of raw-data: We ran different *configurations* of MINISAT on the SAT instances in our training set. Each configuration is a pair of parameter settings[2] for the variable decay and restart

[2] VAR-DECAY $\in \{0.50, 0.75, 0.85, 0.91, 0.93, 0.95, 0.97, 0.99, 0.999\}$
RESTART-INC $\in \{1, 1.25, 1.5\}$

increment parameters of MINISAT. For each instance in the training set, MINISAT was invoked with 27 different configurations (9 different values for variable decay, and 3 for restart-increment) and a timeout of 900 seconds per configuration. For each training instance, and for each MINISAT configuration, the corresponding runtimes were recorded. The raw data for each training instance was compiled into a training example comprising the features of the instance and the parameter configuration for which MINISAT finished fastest. The preprocessing classifier was trained using the training examples compiled from the 177 training instances discussed above, while the course correction classifier was trained using the examples compiled from the 391 conflict clause training instances. Both classifiers were trained using the Radial Basis Function (RBF) Kernel [3].

3.1 Comparison of AVATARSAT with MINISAT

Figure 4 is a scatterplot (in log-scale) of the times taken by AVATARSAT (Y-axis) and MINISAT (X-axis) over all the 75 SAT instances in the test suite. There is one point for each solved SAT instance. For each point, the ratio of the X-axis to the Y-axis is the speedup of AVATARSAT over MINISAT for the corresponding SAT instance. AVATARSAT is faster for points below the diagonal; MINISAT is faster for points above the diagonal.

These data show that AVATARSAT is faster than MINISAT for 34 of the 75 instances, with MINISAT faster on 24 others. For the remaining examples both AVATARSAT and MINISAT time out without solving the instance. AVATARSAT is hundreds to thousands of times faster than MINISAT 2.0 on 10 hard SAT instances out of a total of 75 instances (note that some of the corresponding points in the lower right corner overlap). On at least 5 of these examples MINISAT times out at 43,200 seconds, while AVATARSAT finishes relatively quickly in a few seconds with the correct answer. A summary of the distribution of differences in solving time is given in Figure 3.

We attribute the fact that MINISAT is sometimes faster than AVATARSAT to classification errors on the part of the machine learning classifiers. Note, however, that these classification errors never cause AVATARSAT to execute more than twenty times slower than MINISAT, with the vast majority of the misclassifications resulting in much smaller performance differences between the two solvers. The end result is that AVATARSAT is on average more than two times faster per problem than MINISAT based on the geometric mean speedup measure, and 50% faster based on the arithmetic speedup measure.

Figure 5 shows the number of examples solved (X-axis) as a function of the cumulative solution time (Y-axis). At every point in time, AVATARSAT has solved at least as many problems as MINISAT. Note that, as shown in Figure 1, AVATARSAT solves 53 SAT instances out of a total of 75, compared to 52 by MINISAT. Moreover, AVATARSAT solves these instances much faster.

We also performed experiments with only the initial classification step (i.e., the course correction step was disabled so that the solver used the initially selected parameters for the entire search). The course correction step was found to be important for obtaining good performance on some of the harder instances. It usually makes little to no difference for examples that solve quickly. This reflects the substantial difference in

problem characteristics that can arise between the original problem and the set of derived conflict clauses that the solver accumulates as it attempts to solve a hard instance.

Note that even though the classifiers were trained with data from executions with small timeouts, they generalize well to tests with much larger timeouts.

4 Related Work

There has been recent work in combining machine-learning preprocessors with SAT solvers for the purposes of predicting parameter values and for portfolio selection. In portfolio selection, a classifier predicts the best solver from a fixed set for a particular SAT instance [12, 14, 6, 13]. Previous work in predicting parameter values has used linear regression classifiers to learn a function from features and parameter configurations to solver running time [8]. Other work has focused on per-distribution tuning of parameters [7, 9], in which a fixed set of parameters is chosen for a given distribution based on average running time.

All of these techniques use some kind of a classifier as a pre-processor. By contrast, AVATARSAT calls a machine-learning classifier as a preprocessor, and another classifier internally to correct the course of the solver's search. Additionally, the strategy of learning a function from features and parameter configuration to runtime does not scale as well as our approach of learning a function from features to parameter configuration. The former strategy requires invoking the classifier once per possible parameter configuration—a number of invocations possibly exponential in the number of parameters. Our strategy, by contrast, requires invoking the classifier only once, regardless of the number of parameters.

The use of machine learning techniques to dynamically adjust the search strategy of a SAT solver has been examined [10], but existing approaches are neither as general nor as effective as ours. These approaches have not been tested on the kind of varied problem sets represented by those collected for the SAT competitions, and even so achieve only modest performance improvements. They also rely on branching rules specific to the contemporary SAT solvers of several years ago—rules that have since become obsolete. Our technique, in contrast, allows the tuning of *any* parameterizable feature of a SAT solver, and so will remain relevant even as solvers improve.

References

1. SAT competition website. `http://www.satcompetition.org/`.
2. G. Audemard and L. Simon. Experimenting with small changes in conflict-driven clause learning algorithms. In *Proceedings of the 14th international conference on Principles and Practice of Constraint Programming (CP)*, pages 630–634, Berlin, Heidelberg, 2008. Springer-Verlag.
3. C. Chang and C. Lin. LIBSVM: a library for support vector machines, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.
4. C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

5. N. Eén and N. Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 502–518. Springer, 2003.
6. S. Haim and T. Walsh. Online estimation of sat solving runtime. In *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 133–138. Springer, 2008.
7. F. Hutter, D. Babić, H. H. Hoos, and A. J. Hu. Boosting Verification by Automatic Tuning of Decision Procedures. In *Proceedings of Formal Methods in Computer Aided Design (FMCAD'07)*, pages 27–34, Washington, DC, USA, 2007. IEEE Computer Society.
8. F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 213–228, 2006.
9. F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 1152–1157. AAAI Press, 2007.
10. M. G. Lagoudakis and M. L. Littman. Learning to select branching rules in the dpll procedure for satisfiability. In *In LICS/SAT*, pages 344–359, 2001.
11. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th conference on Design automation (DAC)*, pages 530–535, 2001.
12. E. Nudelman, A. Devkar, Y. Shoham, K. Leyton-brown, and H. Hoos. Satzilla: An algorithm portfolio for sat. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 13–14, 2004.
13. S. A. Seshia. Adaptive eager Boolean encoding for arithmetic reasoning in verification. Tech. Rep. CMU-CS-05-134, School of Computer Science, Carnegie Mellon University, 2005.
14. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla07: The design and analysis of an algorithm portfolio for SAT. In *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 712–727, 2007.
15. E. Zarpas. Benchmarking sat solvers for bounded model checking. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 340–354, 2005.