# SATzilla2008: an Automatic Algorithm Portfolio for SAT

Lin Xu, Frank Hutter, Holger H. Hoos and Kevin Leyton-Brown
Computer Science Dept., University of British Columbia
Vancouver, BC, Canada
{xulin730, hutter, hoos, kevinlb}@cs.ubc.ca

## 1 Introduction

Empirical studies often observe that the performance of algorithms across problem domains can be quite uncorrelated. When this occurs, it seems practical to investigate the use of algorithm portfolios that draw on the strengths of multiple algorithms. SATzilla is such an algorithm portfolio for SAT problems; it was first deployed in the 2004 SAT competition [13], and recently an updated version, SATzilla2007, won a number of prizes in the 2007 SAT competition [23], including the gold medals for the SAT+UNSAT categories of both the random and handmade categories. SATzilla is based on *empirical hardness models* [10, 14], learned predictors that estimate each algorithm's runtime on a given SAT instance.

In SATzilla2007, we integrated new research on empirical hardness models: regression based on partly censored data (i.e. including runs that time out); probabilistic prediction of instance satisfiability; and hierarchical models (separate models for satisfiable and unsatisfiable instances, and probabilistic combination of the two for prediction); see [23] and [24] for a description of these features).

The new features in SATzilla2008 are as follows:

- Automatic selection of pre-solvers
- Prediction of performance score instead of runtime
- Randomized solver subset selection for a large set of candidate solvers

Much of the improvements aims at automating the process of portfolio building. As a result, after obtaining candidate solvers and measuring their runtimes for our training and validation instances, the construction of our SATzilla2008 portfolio took very little time. SATzilla2008's metholodology can be outlined as follows:

**Offline, as part of algorithm development:**

1. Identify a target distribution of problem instances.
2. Select a set of candidate solvers that are known or expected to perform well on at least a subset of the target distribution.
3. Use domain knowledge to identify features that characterize problem instances. To be usable effectively for automated algorithm selection, these features must be related to instance hardness and relatively cheap to compute.
4. On a training set of problem instances, compute these features and run each algorithm to determine its running times. We use the term *performance score* to refer to the quantity we aim to minimize, e.g. the runtime of the algorithm.
5. Automatically determine the best combination of pre-solvers and their runtimes; pre-solvers will later be run for a short amount of time before features are computed (step 1 below), in order to ensure good performance on very easy instances and to allow the predictive models to focus exclusively on harder instances.
6. Using a validation data set, determine which solver achieves the best performance for all instances that are not solved by the pre-solvers and on which the feature computation times out. We refer to this solver as the *backup solver*.
7. Construct a predictive model for each algorithm in the portfolio, which predicts the algorithm's performance score based on instance features.
8. Automatically choose the best subset of solvers to use in the final portfolio.

**Then, online, to solve a given problem instance, the following steps are performed:**

1. Run the pre-solvers in the pre-determined order for up to their pre-determined fixed cutoff times.
2. Compute feature values. If feature computation cannot be completed for some reason (error or timeout), select the backup solver identified in step 6 above.
3. Otherwise, predict each algorithm's performance score using the predictive models from step 7 above.
4. Run the algorithm predicted to be the best. If a solver fails to complete its run (e.g., it crashes), run the algorithm predicted to be next-best.

## 2 New Technologies

SATzilla2008 implements a number of improvements over SATzilla2007, detailed in a recently submitted article [22].

**Automatic selection of pre-solvers.** While for SATzilla2007, we selected pre-solvers manually, this step is now largely automated. Our method assumes that

we can commit to a fixed maximum of pre-solvers, as well as to a small number of possible cutoff times for them. For all possible combinations of pre-solvers and runtimes, we then evaluate SATzilla2008's performance on the validation data by performing the above steps 6 (determining backup solver), 7 (building predictive models) and 8 (solver subset selection), and choose the combination with the best performance.

**Prediction of performance score instead of runtime.** Although previous versions of SATzilla always predicted runtime, our framework is general enough to deal with a variety of cost functions. In a competition context, we aim to maximize the *score* our portfolio achieves in the competition. For the SAT Race, this score rewards the number of instances solved, as well as the speed for solving them. We thus computed the partial score for every solver and instance in the training set, and learned predictive models mapping instance characteristics to scores.

**Local search for solver subset selection.** Because our runtime predictions are not perfect, dropping a solver from the portfolio entirely can increase overall performance. However, the number of possible solver subsets is exponential in the number of component solvers, so picking the best subset may be costly; also keep in mind that we repeat this step for a possibly large number of pre-solver combinations. Thus, when the number of component solvers is large, we perform a randomized local search to select a good solver subset.

In [22], we present two further technologies: the inclusion of local search solvers as component solvers, and more general hierarchical hardness models that allow using the combination of multiple specialized models. Here, we allowed local search solvers as component solvers, but they were not selected by the automatic solver subset selection. We only used hierarchical hardness models that combine models for satisfiable and unsatisfiable instances; separate models could be build for subclasses of industrial instances, but due to a lack of training data we chose not to do this.

## 3 The Resulting Algorithm Portfolio

In order to approximate the target instance distribution of industrial SAT instances, we selected all instances from previous SAT competitions in the industrial category, as well as all instances from the SAT Race 2006. 905 instances were left after excluding all instances not solved by any solver we considered.

SATzilla's performance depends crucially on its component solvers. We considered a number of state-of-the-art SAT solvers from previous competitions as candidate solvers, in particular the fifteen complete solvers `Eureka`[12], `Kcnfs06`[5], `March_dl04`[8], `Minisat 2.0`[6], `Rsat 1.03`[17], `Vallst`[20], `Zchaff_Rand`[11], `Kcnfs04`[4], `TTS`[19], `Picosat5.35`[1], `MXC`[2], `March_ks`[7], `TinisatElite`[9], `Minisat07`[18], `Rsat 2.0`. We also considered the four local search solvers `Ranov`[15], `Ag2wsat0`[3], `Ag2wsat+`[21], and `Gnovelty+`[16].

For each training instance we ran each algorithm and recorded its runtime. The timeout for each algorithm run was set to at least 20 minutes. We computed 44 characteristic features for each instance (similar to the set we used in [24] except four `variable graph` features. These comprised 29 basic features, 7 features from DPLL probes, and 8 features from local search probes. All features were normalized to mean zero and standard deviation one on the training set. The time required for computing basic features was instance-dependent with a timeout of 60 seconds. Two seconds were allocated to computing the local search features for each instance, and one second for DPLL probes. The average complete feature computation time on our training data was 6 seconds (no feature time-out on our reference machine).

For pre-solving, we committed in advance to using a maximum of two pre-solvers. We allowed a number of possible cutoff times, namely 2, 5, 10, and 30 CPU seconds, as well as 0 seconds (i.e., the pre-solver is not run at all) and considered all orders in which to run the three pre-solvers. The automatically chosen combination is to first run `Minisat07` for two seconds, and then run `Picosat5.35` for ten seconds. Together, this pre-solver combination already solved 40.1% of our training instances. Since there is no feature time-out in training phase, the automatically chosen backup solver was the winner-take-all solver `Picosat5.35`.

Next, we learned predictive models for performance score. As in SATzilla2007, we used simple linear regression with quadratic basis functions and forward selection. We selected a model using up to 8 basis functions in order not to overfit on the fairly sparse training data.

In order to choose a solver subset out of our 19 candidate solvers, we performed local search as described in Section 2. This led us to select the following component solvers: `Eureka`, `Minisat 2.0`, `Rsat 1.03`, `Picosat5.35`, `MXC`, `TinisatElite`, `Rsat 2.0`.

In summary, when SATzilla2008 is asked to solve an instance, it first runs `Minisat07` for two seconds, then runs `Picosat5.35` for ten seconds, then computes features (on average in 6 seconds), feeds the computed features into each of the predictive models to get predictions of performance scores (a matter of milliseconds) and then runs the best predicted solver until timeout. If feature computation times out, a default solver is used. If a solver crashes, the next best predicted one is run using the time that remains.

## References


[1] A. Biere. Picosat version 535. Solver description, SAT competition 2007, 2007.

[2] D. R. Bregman and D. G. Mitchell. The SAT solver MXC, version 0.5. Solver description, SAT competition 2007, 2007.

[3] W. Wei C. M. Li and H. Zhang. Combining adaptive noise and promising decreasing variables in local search for SAT. Solver description, SAT competition 2007, 2007.

[4] G. Dequen and O. Dubois. kcnfs. Solver description, SAT competition 2007, 2007.

[5] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proc. of IJCAI-01*, pages 248–253, 2001.

[6] N. Eén and N. Sörensson. Minisat v2.0 (beta). Solver description, SAT Race 2006, 2006.

[7] M. Heule and H. v. Maaren. march_ks. Solver description, SAT competition 2007, 2007.

[8] M. Heule, J. Zwieten, M. Dufour, and H. Maaren. March_eq: implementing additional reasoning into an efficient lookahead SAT solver. pages 345–359, 2004.

[9] J. Huang. TINISAT in SAT competition 2007. Solver description, SAT competition 2007, 2007.

[10] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proc. of CP-02*, pages 556–572, 2002.

[11] Y. S. Mahajan, Z. Fu, and S. Malik. Zchaff2004: an efficient SAT solver. pages 360–375, 2005.

[12] A. Nadel, M. Gordon, A. Palti, and Z. Hanna. Eureka-2006 SAT solver. Solver description, SAT Race 2006, 2006.

[13] E. Nudelman, A. Devkar, Y. Shoham, K. Leyton-Brown, and H. Hoos. SATzilla: An algorithm portfolio for SAT, 2004.

[14] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proc. of CP-04*, pages 438–452, 2004.

[15] D. N. Pham and Anbulagan. Resolution enhanced SLS solver: R+AdaptNovelty+. Solver description, SAT competition 2007, 2007.

[16] D. N. Pham and C. Gretton. gNovelty+. Solver description, SAT competition 2007, 2007.

[17] K. Pipatsrisawat and A. Darwiche. Rsat 1.03: SAT solver description. Technical Report D-152, Automated Reasoning Group, UCLA, 2006.

[18] N. Sörensson and N. Eén. Minisat2007. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/, 2007.

[19] I. Spence. Ternary tree solver (tts-4-0). Solver description, SAT competition 2007, 2007.

[20] D. Vallstrom. Vallst documentation. http://vallst.satcompetition.org/index.html, 2005.

[21] W. Wei, C. M. Li, and H. Zhang. Deterministic and random selection of variables in local search for SAT. Solver description, SAT competition 2007, 2007.

[22] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. Submitted to JAIR, December 2007.

[23] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Satzilla2007: a new & improved algorithm portfolio for SAT. Solver description, SAT competition 2007, 2004.

[24] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Satzilla-07: The design and analysis of an algorithm portfolio for SAT. In *Proc. of CP-07*, pages 712–727, 2007.