SATzilla2009: 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 [12], and recently an updated version, SATzilla2007, won a number of prizes in the 2007 SAT competition [21], including the gold medals for the SAT+UNSAT categories of both the random and handmade categories. SATzilla2008, submitted to the 2008 SAT Race, did not perform as well. We attribute this mainly to the lack of publicly available high-performance component solvers as well as to overheads in computing instance features for huge industrial instances; we addressed this latter point in SATzilla2009.

SATzilla is based on *empirical hardness models* [10, 13], learned predictors that estimate each algorithm's performance on a given SAT instance. Over the years, we have added several features to SATzilla. We integrated regression methods based on partly censored data, probabilistic prediction of instance satisfiability, and hierarchical hardness models [21, 22]. We also almost entirely automated the portfolio construction process based on automatic procedures for selecting pre-solvers and candidate component solvers [23].

The new features in SATzilla2009 are as follows:

- New instance features
- Prediction of feature computation time
- New component algorithms

Due to the automatic procedures we used since SATzilla2008, after obtaining candidate solvers and measuring their runtime for our training and validation instances, the construction of our SATzilla2009 solvers took very little time after we knew the scoring function. SATzilla2009's methodology 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 instances in 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 optimize.
5. Automatically determine the best-scoring combination of pre-solvers and their corresponding performance scored. 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. **New:** Construct a predictive model for feature computation time, given the number of variables and clauses in an instance.
8. Construct a model for each algorithm in the portfolio, predicting the algorithm's performance score on a given instance based on instance features.
9. Automatically choose the best-scoring 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 presolvers in the predetermined order for up to their predetermined fixed cutoff times.
2. **New:** Predict time required for feature computation. If that prediction exceeds two minutes, run the backup solver identified in step 6 above; otherwise continue with the following steps.
3. Compute feature values. If feature computation cannot be completed due to an error, select the backup solver identified in step 6 above; otherwise continue with the following steps.
4. Predict each algorithm's performance score using the predictive models from step 8 above.

5. 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 SATzilla2009 vs SATzilla2008

SATzilla2009 implements a number of improvements over SATzilla2008.

New instance features. We introduced several new classes of instance features: 18 features based on clause-learning [11], 18 based on survey propagation [9], and five based on graph diameter [8]. For the *Industrial* category, we discarded 12 computationally expensive features based on unit propagation, lobjois probing, and graph diameter.

Prediction of feature computation time. In order to predict the feature computation time for an instance based on its number of variables and clauses, we built a simple linear regression model with quadratic basis functions. This was motivated by the fact that in the industrial category of the 2007 SAT competition, as well as in the SAT Race 2008, SATzilla's feature computation timed out on over 50% of the instances, forcing SATzilla to use a default solver; we also discarded some expensive features trading off cost *vs* benefit.

New component algorithms. We updated the component solvers used in SATzilla2008 with the newest publicly-available versions and included a number of local search solvers based on the `SATenstein` solver framework [1]. For the *Industrial* category, one limiting factor is that many high-performance industrial solvers are not publicly available, such that we cannot use them as component solvers.

3 The SATzilla2009 solvers

SATzilla's performance depends crucially on its component solvers. We considered a number of state-of-the-art SAT solvers as candidate solvers, in particular the eleven complete solvers `March_dl04`[8], `March_pl`[7], `Minisat 2.0`[6], `Vallst`[19], `Zchaff_Rand`[11], `Kcnfs04`[5], `TTS 4.0`[18], `Picosat8.46`[2], `MXC08`[3], `Minisat 2007`[17] and `Rsat 2.0`[16].

We also considered the five local search solvers `gnovelty+`[15], `Ranov`[14], `Ag2wsat0`[4], `Ag2wsat+`[20] and `SATenstein`[1] (seven automatically configured versions).

As training data, we used all available SAT instances from previous SAT competitions (2002 until 2005, and 2007) and from the SAT Races 2006 and 2008. Based on these instances, we built three data sets:

- *Random*: all 2,821 random instances;
- *Crafted*: all 1,686 handmade/crafted instances;
- *Industrial*: all 1,376 industrial instances.

For each training instance we ran each solver for one hour and recorded its runtime. (Local search solvers were only run on unsatisfiable instances.) Unlike in previous SATzilla versions, we did not use any preprocessing. We computed 96 features for each instance in categories *Random* and *Crafted*, and 84 features for category *Industrial*. In each category, as a training set we used all previously mentioned instances, and as a validation set the 2007 SAT competition instances from that category (note that this validation is a subset of the training; this was motivated by the relative scarcity of available data and our expectation that the 2009 SAT competition instances resemble more closely those from the 2007 competition than those from earlier competitions).

For presolving, we committed in advance to using a maximum of two presolvers. We allowed a number of possible cutoff times, namely 5, 10, and 30 CPU seconds, as well as 0 seconds (i.e., the presolver is not run at all) and considered all orders in which to run the three presolvers. Automated presolver selection then chose the following presolving strategies:

- *Random*: `SATenstein`(T7) for 30 seconds, then `MXC08` for 30 seconds;
- *Crafted*: `March_dl04` for 5 seconds, then `MXC08` for 5 seconds;
- *Industrial*: `MXC08` for 10 seconds, then `Picosat8.46` for 5 seconds.

Automated solver subset selection [23] chose the following component solvers:

- *Random*: `Kcnfs04`, `March_dl04`, `Picosat8.46`, `Ag2wsat0`, `Ag2wsat+`, `gnovelty+`, `SATenstein`(QCP)
- *Crafted*: `March_dl04`, `Minisat 2.0`, `Minisat 2007`, `Vallst`, `Zchaff_Rand`, `TTS 4.0`, `MXC08`
- *Industrial*: `March_dl04`, `Minisat 2007`, `Zchaff_Rand`, `Picosat8.46`, `MXC08`

The automatically-selected backup solvers were `Ag2wsat0`, `Minisat 2007`, and `MXC08` for *Random*, *Handmade*, *Industrial*, respectively.

4 Expected Behaviour

We submit three different versions of SATzilla, specifically designed to perform well in each of the categories: SATzilla2009_R (*Random*), SATzilla2009_C (*Crafted*), and SATzilla2009_I (*Industrial*). In order to run properly, subdirectory `satzilla_Solvers` should contain all binaries for SATzilla's component solvers and its feature computation.

References

[1] Anonymous. Satenstein: Automatically building local search sat solvers from components. Under double-blind review., 2009.

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

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

[4] 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.

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

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

[7] M. Heule and H. v. Maaren. March_pl. http://www.st.ewi.tudelft.nl/sat/download.php, 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] E. I. Hsu and S. A. McIlraith. Characterizing propagation methods for boolean satisfiability. pages 325–338, 2006.

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

[13] 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.

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

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

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

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

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

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

[20] 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.

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

[22] 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.

[23] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, June 2008.