Non-Model-Based Algorithm Portfolios for SAT
(Extended Abstract)

Yuri Malitsky[1], Ashish Sabharwal[2], Horst Samulowitz[2], and Meinolf Sellmann[2]

[1] Brown University, Dept. of Computer Science, Providence, RI 02912, USA
ynm@cs.brown.edu
[2] IBM Watson Research Center, Yorktown Heights, NY 10598, USA
{ashish.sabharwal,samulowitz,meinolf}@us.ibm.com

When tackling a computationally challenging combinatorial problem, one often observes that some solution approaches work well on some instances, while other approaches work better on other instances. This observation has given rise to the idea of building algorithm portfolios [5]. Leyton-Brown et al. [1], for instance, proposed to select one of the algorithms in the portfolio based on some features of the instance to be solved. This approach has been blessed with tremendous success in the past. Especially in SAT, the SATzilla portfolios [7] have performed extremely well in past SAT Competitions [6].

We investigate alternate ways of building algorithm portfolios that differ substantially from the way SATzilla assembles a portfolio. The key idea behind SATzilla is to train a runtime prediction model for each constituent solver, based on a number of well-engineered features of SAT instances. Given a new instance, SATzilla predicts the runtime of each candidate solver based on instance features and the trained models, and chooses the solver that is predicted to perform the best. In contrast, we consider *non-model-based machine learning techniques* such as simple k-nearest-neighbor (k-NN) classification to determine which solver to use to tackle a given instance.

Our motivation stems from two observations: (a) accurately predicting the runtime of sophisticated SAT solvers is a very challenging task; indeed, the runtime predictor underlying SATzilla can be even orders of magnitude off from the true runtime; and (b) while fast and accurate runtime prediction is certainly sufficient for building a solver portfolio, it is by no means necessary. In fact, it would suffice entirely if we could predict the fastest solver without having any knowledge of how long it will actually take to solve the given instance. This idea has found success in fields adjoining SAT, for example in portfolios for the quantified Boolean formula (QBF) problem [4], for general constraint satisfaction problems (CSPs) [3], and to some extent even for SAT itself [2].

Our portfolio works as follows. In the learning phase, we are given a pool $\mathcal{T}$ of training instances, a function that provides features for any given problem instance (we use the 48 core SATzilla features here), a set $\mathcal{S}$ of constituent solvers forming the portfolio, and a timeout t. We compute the runtime (with cutoff t) for all solvers on all instances as well as normalization parameters so that all features for all instances in the training set populate the interval $[0, 1]$.

At runtime, given a new instance I, we compute its features, normalize them, and compute the set $T_I \subset \mathcal{T}$ consisting of k training instances closest to I in

Table 1. Performance comparison of pure solvers, portfolios, and virtual best solver

	Pure Solvers							Portfolios		VBS
	agw-sat0	agw-sat+	gnov-elty+	**kcnfs**	march	pico-sat	SAT-enstein	SAT-zilla	**12-NN**	
PAR10	6400	6667	6362	**5813**	6524	7384	7089	4399	**3940**	3454
Avg Time	678	698	677	**659**	688	752	722	534	**529**	480
# Solved	268	255	270	**298**	262	220	234	366	**390**	413
% Solved	47.0	44.7	47.4	**52.3**	46.0	38.6	41.1	64.2	**68.4**	72.5

terms of Euclidean distance. Then, for each solver $S \in \mathcal{S}$, we compute the penalized runtime (PAR10 score) of S on T_I, and select the solver that has the lowest PAR10 score as our recommended solver to use on I. The choice of k can have an impact on the performance of the portfolio. We therefore learn a "good" value of k for the training set $\mathcal{T}$ by performing cross-validation with 100 random sub-samples of base-validation splits in a 70-30 ratio.

Extensive empirical results are omitted due to lack of space. Table 1 shows one representative sample of our results, comparing against SATzilla2009_R, the Gold Medal winning solver in the random category of SAT Competition 2009 [6]. We base our portfolio on the same set of solvers as SATzilla2009_R, use the 2,247 random category instances from SAT Competitions 2002-2007 as our training set and the 570 random category instances from SAT Competition 2009 as the test set, and a 1,200 second timeout. Experiments were run on Intel dual-core, dual processor, Dell Poweredge 1855 blade servers with 8GB of memory each.

As Table 1 shows, SATzilla outperforms individual solvers dramatically, solving 68 more instances (366) than the best performing individual solver, kcnfs. Our k-NN approach pushes the performance level substantially further, solving 390 instances within 1200 seconds whereby the VBS can solve only 23 more. In other words, SATzilla closes 55% of the gap between the best individual solver and the best possible portfolio. Simple k-NN closes 80% of this gap. We conclude that this easy non-model-based approach marks a significant improvement over a portfolio approach that has dominated SAT Competitions for half a decade.

References

1. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A Portfolio Approach to Algorithm Selection. *IJCAI*, 1542–1543, 2003.
2. M. Nikolić, F. Marić, and P. Janičić. Instance-Based Selection of Policies for SAT Solvers. *SAT*, 326–340, 2009.
3. E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving. *Irish Conference on AI and Cognitive Science*, 2008.
4. L. Pulina and A. Tacchella. A Multi-Engine Solver for Quantified Boolean Formulas. *CP*, 574–589, 2007.
5. J. R. Rice The algorithm selection problem. *Advances in computers*, 65–118, 1976.
6. SAT Competition. *http://www.satcomptition.org*.
7. L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *JAIR*, 32(1):565–606, 2008.