Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors

Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown

Department of Computer Science, University of British Columbia
{xulin730, hutter, hoos, kevinlb}@cs.ubc.ca

Abstract. Portfolio-based methods exploit the complementary strengths of a set of algorithms and—as evidenced in recent competitions—represent the state of the art for solving many NP-hard problems, including SAT. In this work, we argue that a state-of-the-art method for constructing portfolio-based algorithm selectors, SATzilla, also gives rise to an automated method for quantifying the importance of each of a set of available solvers. We entered a substantially improved version of SATzilla to the inaugural "analysis track" of the 2011 SAT competition, and draw two main conclusions from the results that we obtained. First, automatically-constructed portfolios of sequential, non-portfolio competition entries perform substantially better than the winners of all three sequential categories. Second, and more importantly, a detailed analysis of these portfolios yields valuable insights into the nature of successful solver designs in the different categories. For example, we show that the solvers contributing most to SATzilla were often *not* the overall best-performing solvers, but instead solvers that exploit novel solution strategies to solve instances that would remain unsolved without them.

1 Introduction

The propositional satisfiability problem (SAT) is among the most widely studied NP-complete problems, with applications to problems as diverse as scheduling [4], planning [15], graph colouring [30], bounded model checking [1], and formal verification [25]. The resulting diversity of SAT instances fueled the development of a multitude of solvers with complementary strengths.

Recent results, reported in the literature as well as in solver competitions, have demonstrated that this complementarity can be exploited by so-called "algorithm portfolios" that combine different SAT algorithms (or, indeed, algorithms for solving other hard combinatorial problems). Such algorithm portfolios include methods that select a single algorithm on a per-instance basis [21, 7, 10, 31, 24, 26], methods that make online decisions between algorithms [16, 3, 23], and methods that run multiple algorithms independently on one instance, either in parallel or sequentially [13, 9, 20, 6, 27, 14, 22, 11].

To show that such methods work in practice as well as in theory—that despite their overhead and potential to make mistakes, portfolios can outperform their constituent solvers—we submitted our SATzilla portfolio-based algorithm

selectors to the international SAT competition (`www.satcompetition.org`), starting in 2003 and 2004 [18, 19]. After substantial improvements [31], `SATzilla` won three gold and two other medals (out of nine categories overall) in each of the 2007 and 2009 SAT competitions. Having established `SATzilla`'s effectiveness, we decided not to compete in the main track of the 2011 competition, to avoid discouraging new work on (non-portfolio) solvers. Instead, we entered `SATzilla` in a new "analysis track". However, other portfolio-based methods did feature prominently among the winners of the main track: the algorithm selection and scheduling system `3S` [14] and the simple, yet efficient parallel portfolio `ppfolio` [22] won a total of seven gold and 16 other medals (out of 18 categories overall).

One of the main reasons for holding a SAT competition is to answer the question, *What is the current state of the art (SOTA) in SAT solving?*. The traditional answer to this question has been *the winner of the respective category of the SAT competition*; we call such a winner a *single best solver (`SBS`)*. However, as clearly demonstrated by the efficacy of algorithm portfolios, different solver strategies are (at least sometimes) complementary. This fact suggests a second answer to the SOTA question: *the virtual best solver (`VBS`),* defined as the best SAT competition entry on a per-instance basis. The `VBS` typically achieves much better performance than the `SBS`, and does provide a useful theoretical upper bound on the performance currently achievable. However, this bound is typically not tight: the `VBS` is not an actual solver, because it only tells us which underlying solver to run after the performance of each solver on a given instance has been measured, and thus the `VBS` cannot be run on new instances. Here, we propose a third answer to the SOTA question: *a state-of-the-art portfolio that can be constructed in a fully automatically fashion from available solvers* (using an existing piece of code); we call such a portfolio a *SOTA portfolio*. Unlike the `VBS`, a SOTA portfolio is an actual solver that can be run on novel instances. We show how to build a SOTA portfolio based on an improved version of our `SATzilla` procedure, and then demonstrate techniques for analyzing the extent to which its performance depends on each of its component solvers. While (to the best of our knowledge) `SATzilla` is the only fully automated and publicly available portfolio construction method, our measures of solver contributions may also be applied to other portfolio approaches, including parallel portfolios.

In this paper, we first verify that our automatically constructed `SATzilla 2011` portfolios yielded cutting-edge performance (they closed 27.7% to 90.1% of the gap between the winners of each sequential category and the `VBS`). Next, we perform a detailed, quantitative analysis of the performance of `SATzilla 2011`'s component solvers and their pairwise correlations, their frequency of selection and success within `SATzilla 2011`, and their overall benefit to the portfolio. Overall, our analysis reveals that the solvers contributing most to `SATzilla 2011` were often *not* the overall best-performing solvers (`SBS`s), but instead solvers that exploit novel solution strategies to solve instances that would remain unsolved without them. We also provide a quantitative basis for the folk knowledge that—due to the dominance of MiniSAT-like architectures—the performance of most solvers for `Application` instances is tightly correlated. Our results suggest a shift

away from rewarding solvers that perform well on average and towards rewarding creative approaches that can solve instance types not solved well by other solvers (even if they perform poorly on many other types of instances).

2 Improved Automated Construction of Portfolio-Based Algorithm Selectors: SATzilla 2011

In this work, we use an improved version of the construction method underlying the portfolio-based algorithm selection system SATzilla. We first describe SATzilla's automated construction procedure (which, at a high level, is unchanged since 2008; see [31] for details) and then discuss the improvements to its algorithm selection core we made in 2011.

How SATzilla works. We first describe the automated SATzilla construction procedure and then describe the pipeline that the resulting portfolio executes on a new instance. First, SATzilla's construction procedure uses a set of very cheap features (*e.g.*, number of variables and clauses) to learn a classifier $\mathcal{M}$ that predicts whether the computation of a more comprehensive set of features will succeed within a time threshold t_f; SATzilla also selects a backup solver B that has the best performance on instances with large feature cost ($> t_f$ or failed). Second, it chooses a set of candidate presolvers and candidate presolving time cutoffs based on solver performance on the training set. Then, for all possible combinations of up to two presolvers and given presolving time cutoffs, it learns an algorithm selector based on all training instances with feature cost $\leq t_f$ that cannot be solved by presolvers. The final portfolio-based selector is chosen as the combination of presolvers, presolving cutoffs, and the corresponding algorithm selector with the best training performance.

When asked to solve a given instance π, SATzilla first inspects π to gather very cheap features and uses them with $\mathcal{M}$ to predict feature computation time. If this prediction exceeds t_f, it runs solver B for the remaining time. Otherwise, it sequentially runs its presolvers up to their presolving cutoffs. If a presolver succeeds, SATzilla terminates. Otherwise, it computes features (falling back on B if feature computation fails or exceeds t_f), uses them in its algorithm selector to pick the most promising algorithm and runs that for the remaining time.

Improvements to SATzilla's Algorithm Selection Core. For our entry to the 2011 SAT competition analysis track, we improved SATzilla's models by basing them on cost-sensitive decision forests, rather than linear or quadratic ridge regression. (This improvement is described in [32].) Our new selection procedure uses an explicit cost-sensitive loss function—punishing misclassifications in direct proportion to their impact on portfolio performance—without predicting runtime. Such an approach has never before been applied to algorithm selection: all existing classification approaches use a simple 0–1 loss function that penalizes all misclassifications equally (e.g., [23, 10, 12]), whereas previous versions of SATzilla used regression-based runtime predictions. Our cost-sensitive classification approach based on decision forests (DFs) has the advantage that it effectively partitions the feature space into qualitatively different parts; fur-

thermore, in contrast to clustering methods, DFs take the response variable (here "algorithm A performs better/worse than algorithm B") into account when determining that partitioning.

We construct cost-sensitive DFs as collections of 99 cost-sensitive decision trees [29], following standard random forest methodology [2]. Given n training data points with k features each, for each tree we construct a bootstrap sample of n training data points sampled uniformly at random with replacement. During tree construction, we sample a random subset of $\lceil \log_2(k) + 1 \rceil$ features at each internal node to be considered for splitting the data at that node. Predictions are based on majority votes across all trees. Given a set of m algorithms $\{A_1, \ldots, A_m\}$, an $n \times k$ matrix holding the values of k features for each of n training instances, and an $n \times m$ matrix P holding the performance of the m algorithms on the n instances, we construct our selector based on $m(m-1)/2$ pairwise cost-sensitive decision forests, determining the labels and costs as follows. For any pair of algorithms (A_i, A_j), we train a cost-sensitive decision forest $DF(i,j)$ on the following weighted training data: we label an instance q as i if $P(q,i)$ is better than $P(q,j)$, and as j otherwise; the weight for that instance is $|P(q,i) - P(q,j)|$. For each test instance, we apply each $DF(i,j)$ to vote for either A_i or A_j and select the algorithm with the most votes as the best algorithm for that instance. Ties are broken by only counting the votes from those decision forests that involve algorithms which received equal votes; further ties are broken randomly. Our implementation of `SATzilla 2011`, integrating cost-sensitive decision forests based on Matlab R2010a's implementation of cost-sensitive decision trees, is available online at `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla`. Our experimental results show that `SATzilla 2011` always outperformed `SATzilla 2009`; this gap was particularly substantial in the `Application` category (see Table 1).

3 Measuring the Value of a Solver

We believe that the SAT community could benefit from rethinking how the value of individual solvers is measured. The most natural way of assessing the performance of a solver is by means of some statistic of its performance over a set (or distribution) of instances, such as the number of instances solved within a given time budget, or its average runtime on an instance set. While we see value in these performance measures, we believe that they are not sufficient for capturing the value a solver brings to the community. Take, for example, two solvers `MiniSAT′++` and `NewSAT`, where `MiniSAT′++` is based on MiniSAT [5] and improves some of its components, while `NewSAT` is a (hypothetical) radically different solver that performs extremely well on a limited class of instances and poorly elsewhere.[1] While `MiniSAT′++` has a good chance of winning medals in

[1] In the 2007 SAT competition, the solver closest to `NewSAT` was `TTS`: it solved 12 instances unsolved by all others and thus received a silver medal under the purse scoring scheme [8] (discussed below), even though it solved many fewer instances than the bronze medalist `Minisat` (39 instances vs 72). Purse scoring was abandoned for the 2009 competition.

the SAT competition's `Application` track, `NewSAT` might not even be submitted, since (due to its poor average performance) it would be unlikely even to qualify for the final phase of the competition. However, `MiniSAT′++` might only very slightly improve on the previous (MiniSAT-based) incumbent's performance, while `NewSAT` might represent deep new insights into the solution of instances that are intractable for all other known techniques.

One way of evaluating the value a solver brings to the community is through the notion of *state-of-the-art (SOTA) contributors* [28]. It ranks the contribution of a constituent solver A by the performance decrease of `VBS` when omitting A and reflects the added value due to A much more effectively than A's average performance. A related method for scoring algorithms is the "purse score" [8], which rewards solving instances that cannot be solved by other solvers; it was used in the 2005 and 2007 SAT competitions. However, both of these methods describe idealized solver contributions rather than contributions to an actual executable method. Instead, we propose to measure the SOTA contribution of a solver as its contribution to a SOTA portfolio that can be automatically constructed from available solvers. This notion resembles the prior notion of SOTA contributors, but directly quantifies their contributions to an *executable* portfolio solver, rather than to an abstract virtual best solver (`VBS`). Thus, we argue that an additional scoring rule should be employed in SAT competitions to recognize solvers (potentially with weak overall performance) for the contributions they make to a *SOTA portfolio*.

We must still describe exactly how we should assess a solver A's contribution to a portfolio. We might measure the frequency with which the portfolio selects A, or the number of instances the portfolio solves using A. However, neither of these measures accounts for the fact that if A were not available other solvers would be chosen instead, and might perform nearly as well. (Consider again `MiniSAT′++`, and assume that it is chosen frequently by a portfolio-based selector. However, if it had not been created, the set of instances solved might be the same, and the portfolio's performance might be only slightly less.) We argue that a solver A should be judged by its *marginal contribution* to the SOTA: the difference between the SOTA portfolio's performance including A and the portfolio's performance excluding A. (Here, we measure portfolio performance as the percentage of instances solved, since this is the main performance metric in the SAT competition.)

4 Experimental setup

Solvers. In order to evaluate the SOTA portfolio contributions of the SAT competition solvers, we constructed `SATzilla` portfolios using all sequential, non-portfolio solvers from Phase 2 of the 2011 SAT Competition as component solvers: 9, 15, and 18 candidate solvers for the `Random`, `Crafted`, and `Application` categories, respectively. (These solvers and their individual performance are shown in Figures 1(a), 2(a), and 3(a); see [17] for detailed information.) We hope that in the future, construction procedures will also be made publicly

available for other portfolio builders, such as 3S [14]; if so, our analysis could be easily and automatically repeated for them. For each category, we also computed the performance of an *oracle* over sequential non-portfolio solvers (an idealized algorithm selector that picks the best solver for each instance) and the *virtual best solver* (*VBS*, an oracle over all 17, 25 and 31 entrants for the Random, Crafted and Application categories, respectively). These oracles do not represent the current state of the art in SAT solving, since they cannot be run on new instances; however, they serve as upper bounds on the performance that any portfolio-based selector over these solvers could achieve. We also compared to the performance of the winners of all three categories (including other portfolio-based solvers).

Features. We used 115 features similar to those used by SATzilla in the 2009 SAT Competition. They fall into 9 categories: problem size, variable graph, clause graph, variable-clause graph, balance, proximity to Horn formula, local search probing, clause learning, and survey propagation. Feature computation averaged 31.4, 51.8 and 158.5 CPU seconds on Random, Crafted, and Application instances, respectively; this time counted as part of SATzilla's runtime budget.

Methods. We constructed SATzilla 2011 portfolios using the improved procedure described in Section 2. We set the feature computation cutoff t_f to 500 CPU seconds (a tenth of the time allocated for solving an instance). To demonstrate the effectiveness of our improvement, we also constructed a version of SATzilla 2009 (which uses ridge regression models), using the same training data.

We used 10-fold cross-validation to obtain an unbiased estimate of SATzilla's performance. First, we eliminated all instances that could not be solved by any candidate solver (we denote this instance set as U). Then, we randomly partitioned the remaining instances (denoted S) into 10 disjoint sets. Treating each of these sets in turn as the test set, we constructed SATzilla using the union of the other 9 sets as training data and measured SATzilla's runtime on the test set. Finally, we computed SATzilla's average performance across the 10 test sets.

To evaluate how important each solver was for SATzilla, for each category we quantified the marginal contribution of each candidate solver, as well as the percentage of instances solved by each solver during SATzilla's presolving (Pre1 or Pre2), backup, and main stages. Note that our use of cross-validation means that we constructed 10 different SATzilla portfolios using 10 different subsets ("folds") of instances. These 10 portfolios can be qualitatively different (e.g., selecting different presolvers); we report aggregates over the 10 folds.

Data. Runtime data was provided by the organizers of the 2011 SAT competition. All feature computations were performed by Daniel Le Berre on a quad-core computer with 4GB of RAM and running Linux, using our code. Four out of 1200 instances (from the Crafted category) had no feature values, due to a database problem caused by duplicated file name. We treated these instances as timeouts for SATzilla, thus obtaining a lower bound on SATzilla's true performance.

Solver	Application Runtime (Solved)	Crafted Runtime (Solved)	Random Runtime (Solved)
VBS	1104 (84.7%)	1542 (76.3%)	1074 (82.2%)
Oracle	1138 (84.3%)	1667 (73.7%)	1087 (82.0%)
SATzilla 2011	1685 (75.3%)	2096 (66.0%)	1172 (80.8%)
SATzilla 2009	1905 (70.3%)	2219(63.0%)	1205 (80.3%)
Gold medalist	Glucose2: 1856 (71.7%)	3S: 2602 (54.3%)	3S: 1836 (68.0%)
Best comp.	Glucose2: 1856 (71.7%)	Clasp2: 2996 (49.7%)	Sparrow: 2066 (60.3%)

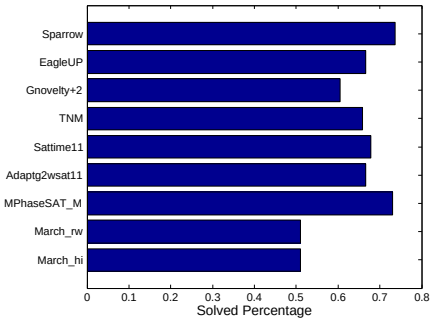
Table 1. *Comparison of SATzilla 2011 to the VBS, an Oracle over its component solvers, SATzilla 2009, the 2011 SAT competition winners, and the best single SATzilla 2011 component solver for each category. We counted timed-out runs as 5000 CPU seconds (the cutoff). Because the construction procedure for 3S portfolios is not publicly available, we used the original 2011 SAT-competition-winning version of 3S, which was trained on pre-competition data; it is likely that 3S would have achieved better performance if we had been able to retrain it on the same data we used to train SATzilla.*

5 Experimental Results

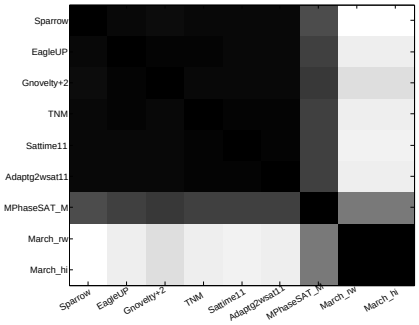
We begin by assessing the performance of our SATzilla portfolios, to confirm that they did indeed yield SOTA performance. Table 1 compares SATzilla 2011 to the other solvers mentioned above. SATzilla 2011 outperformed all of its component solvers in all categories . It also always outperformed SATzilla 2009, which in turn was slightly worse than the best component solver on Application.

SATzilla 2011 also outperformed each category's gold medalist (including portfolio solvers, such as 3S and ppfolio). Note that this does not constitute a fair comparison of the underlying portfolio construction procedures, as SATzilla had access to data and solvers unavailable to portfolios that competed in the competition. This finding does, however, give us reason to believe that SATzilla portfolios either represent or at least closely approximate the best performance reachable by current methods. Indeed, in terms of instances solved, SATzilla 2011 reduced the gap between the gold medalists and the (upper performance bound defined by the) VBS by 27.7% on Application, by 53.2% on Crafted, and by 90.1% on Random. The remainder of our analysis concerns the contributions of each component solver to these portfolios. To substantiate our previous claim that marginal contribution is the most informative measure, here we contrast it with various other measures.

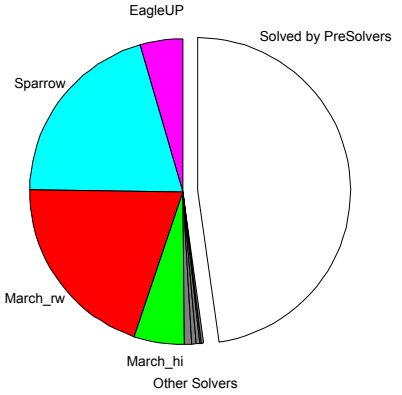
Random. Figure 1 presents a comprehensive visualization of our findings for the Random category. (Table 2 in supplemental material at http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/SAT12-EvaluatingSolverContributions.pdf shows the data underlying all figures.) First, Figure 1(a) considers the set of instances that could be solved by at least one solver, and shows the percentage that each component solver is able to solve. By this measure, the two best solvers were Sparrow and MPhaseSAT_M. The former is a local search algorithm; it solved 362 + 0 satisfiable and unsatisfiable instances, respectively. The latter is a complete search algorithm; it solved 255 + 104 = 359 instances. Neither of these solvers



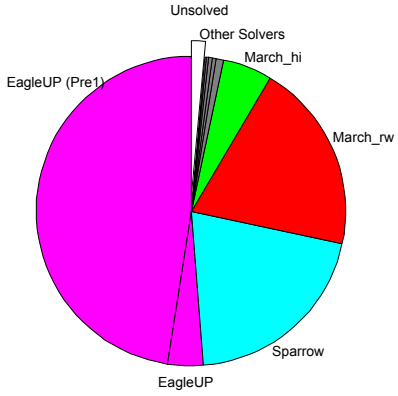
(a) Percentage of solvable instances solved by each component solver alone



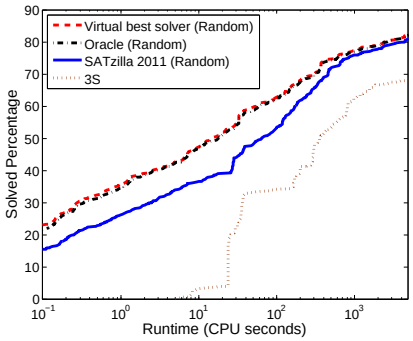
(b) Correlation matrix of solver runtimes (darker = more correlated)



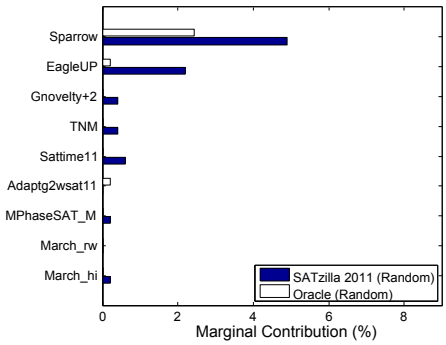
(c) Solver selection frequencies for `SATzilla 2011`; names only shown for solvers picked for $> 5\%$ of the instances



(d) Fraction of instances solved by `SATzilla 2011` pre-solver(s), backup solver, and main stage solvers



(e) Runtime CDF for `SATzilla 2011`, `VBS`, `Oracle`, and the gold medalist `3S`



(f) Marginal contribution to `Oracle` and `SATzilla 2011`

Fig. 1. Visualization of results for category `Random`. In Figure 1(e), we used the original 2011 SAT-competition-winning version of `3S`, which was trained on pre-competition data; thus, this figure is not a fair comparison of the `SATzilla` and `3S` portfolio-building strategies. The data underlying this figure can be found in Table 2 of the supplemental material at `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/SAT12-EvaluatingSolverContributions.pdf`.

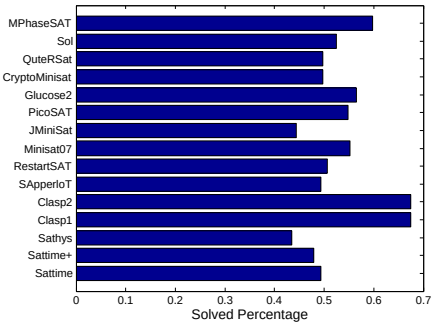
won medals in the combined SAT + UNSAT `Random` category, since all medals went to portfolio solvers that combined local search and complete solvers.

Figure 1(b) shows a correlation matrix of component solver performance: the entry for solver pair (A, B) is computed as the Spearman rank correlation coefficient between A's and B's runtime, with black and white representing perfect correlation and perfect independence respectively. Two clusters are apparent: six local search solvers (`EagleUP`, `Sparrow`, `Gnovelty+2`, `Sattime11`, `Adaptg2wsat11`, and `TNM`), and two versions of the complete solver `March`, which achieved almost identical performance. `MPhaseSAT_M` performed well on both satisfiable and unsatisfiable solvers; it was strongly correlated with local search solvers on the satisfiable instance subset and very strongly correlated with the `March` variants on the unsatisfiable subset.

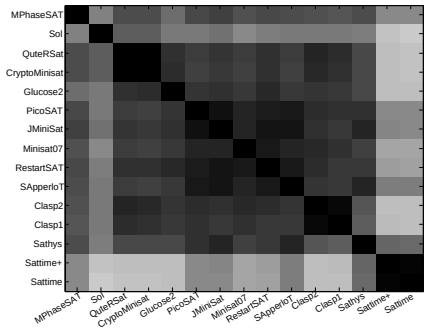
Figure 1(c) shows the frequency with which different solvers were selected in `SATzilla 2011`. The main solvers selected in `SATzilla 2011`'s main phase were the best-performing local search solver `Sparrow` and the best-performing complete solver `March`. As shown in Figure 1(d), the local search solver `EagleUP` was consistently chosen as a presolver and was responsible for more than half (51.3%) of the instances solved by `SATzilla 2011` overall. We observe that `MPhaseSAT_M` did not play a large role in `SATzilla 2011`: it was only run for 2 out of 492 instances (0.4%). Although `MPhaseSAT_M` achieved very strong overall performance, its versatility appears to have come at the price of not excelling on either satisfiable or unsatisfiable instances, being largely dominated by local search solvers on the former and by `March` variants on the latter. Figure 1(e) shows that `SATzilla 2011` closely approximated both the `Oracle` over its component solvers and the `VBS`, and stochastically dominated the category's gold medalist `3S`.

Finally, Figure 1(f) shows the metric that we previously argued is the most important: each solver's marginal contribution to `SATzilla 2011`'s performance. The most important portfolio contributor was `Sparrow`, with a marginal contribution of 4.9%, followed by `EagleUP` with a marginal contribution of 2.2%. `EagleUP`'s low marginal contribution may be surprising at first glance (recall that it solved 51.3% of the instances `SATzilla 2011` solved overall); however, most of these instances (49.1% out of 51.3%) were also solvable by other local search solvers. Similarly, both `March` variants had very low marginal contribution (0% and 0.2%, respectively) since they were essentially interchangeable (correlation coefficient 0.9974). Further insight can be gained by examining the marginal contribution of *sets* of highly correlated solvers. The marginal contribution of the set of both `March` variants was 4.0% (`MPhaseSAT_M` could still solve the most instances), while the marginal contribution of the set of six local search solvers was 22.5% (nearly one-third of the satisfiable instances were not solvable by any complete solver).

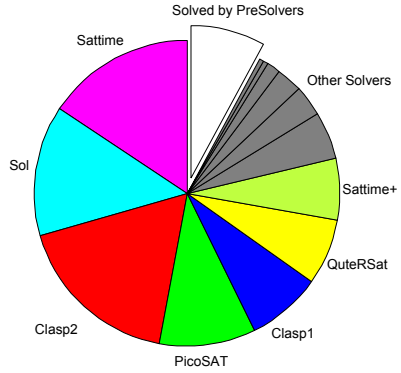
Crafted. Overall, sufficiently many solvers were relatively uncorrelated in the `Crafted` category (Figure 2) to yield a portfolio with many important contributors. The most important of these was `Sol`, which solved all of the 13.7% of the instances for which `SATzilla 2011` selected it; without it, `SATzilla 2011` would have solved 8.1% fewer instances! We observe that `Sol` was not identified as an important solver in the SAT competition results, ranking 11th of 24 solvers in



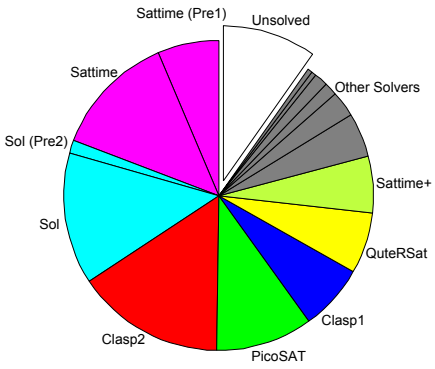
(a) Percent of solvable instances solved by each component solver alone



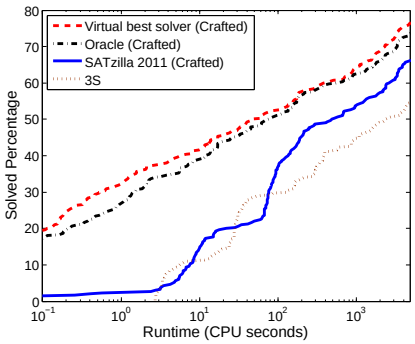
(b) Correlation matrix of solver runtimes (darker = more correlated)



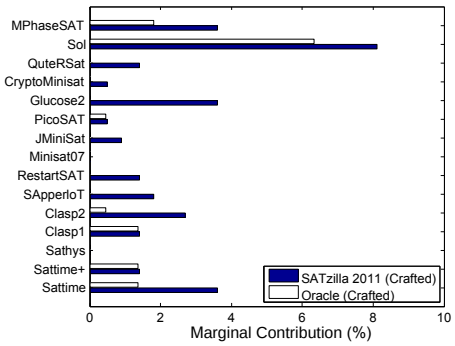
(c) Solver selection frequencies for `SATzilla 2011`; names only shown for solvers picked for $> 5\%$ of the instances



(d) Fraction of instances solved by `SATzilla 2011` pre-solver(s), backup solver, and main stage solvers



(e) Runtime CDF for `SATzilla 2011`, `VBS`, `Oracle`, and the gold medalist `3S`



(f) Marginal contribution for `Oracle` and `SATzilla 2011`

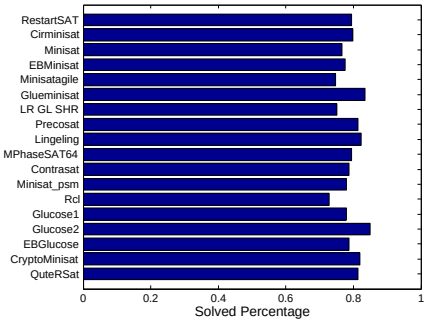
Fig. 2. Visualization of results for category `Crafted`. In Figure 2(e), we used the original 2011 SAT-competition-winning version of `3S`, which was trained on pre-competition data; thus, this figure is not a fair comparison of the `SATzilla` and `3S` portfolio-building strategies. The data underlying this figure can be found in Table 2 of the supplemental material at `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/SAT12-EvaluatingSolverContributions.pdf`.

the SAT+UNSAT category. Similarly, MPhaseSAT_M, Glucose2, and Sattime each solved a 3.6% fraction of instances that would have gone unsolved without them. (This is particularly noteworthy for MPhaseSAT_M, which was only selected for 5% of the instances in the first place.) Considering the marginal contributions of sets of highly correlated solvers, we observed that {Clasp1, Clasp2} was the most important at 6.3%, followed by {Sattime, Sattime11} at 5.4%. {QuteRSat, CryptoMiniSat} and {PicoSAT, JMiniSat, Minisat07, RestartSAT, SApperloT} were relatively unimportant even as sets, with marginal contributions of 0.5% and 1.8% respectively.

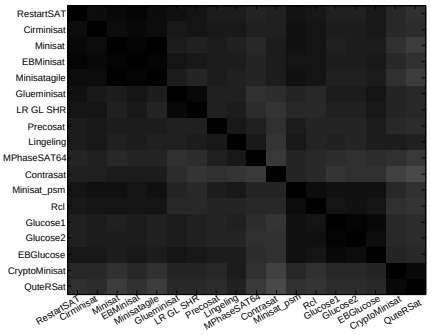
Application. All solvers in the Application category (Figure 3) exhibited rather highly correlated performance. It is thus not surprising that in 2011, no medals went to portfolio solvers in the sequential Application track, and that in 2007 and 2009, SATzilla versions performed worst in this track, only winning a single gold medal in the 2009 satisfiable category. As mentioned earlier, SATzilla 2011 did outperform all competition solvers, but here the margin was only 3.6% (as compared to 12.8% and 11.7% for Random and Crafted, respectively). All solvers were rather strongly correlated, and each solver could be replaced in SATzilla 2011 without a large decrease in performance; for example, dropping the competition winner only decreased SATzilla 2011's percentage of solved instances by 0.4%. The highest marginal contribution across all 18 solvers was four times larger: 1.6% for MPhaseSAT64. Like MPhaseSAT in the Crafted category, it was selected infrequently (only for 3.6% of the instances) but was the only solver able to solve about half of these instances. We conjecture that this was due to its unique phase selection mechanism. Both MPhaseSAT64 and Sol (in the Crafted category) thus come close to the hypothetical solver NewSAT mentioned earlier: they showed outstanding performance on certain instances and thus contributed substantially to a portfolio, but achieved unremarkable rankings in the competition (9th of 26 for MPhaseSAT64, 11th of 24 for Sol). We did observe one set of solvers that achieved a larger marginal contribution than that of MPhaseSAT64: 2.3% for {Glueminisat, LR GL SHR}. The other three highly correlated clusters also gave rise to relatively high marginal contributions: 1.5% for {CryptoMiniSat, QuteRSat}, 1.5% for {Glucose1,Glucose2,EBGlucose}, and 1.2% for {Minisat,EBMiniSAT, MiniSATagile}.

6 Conclusions

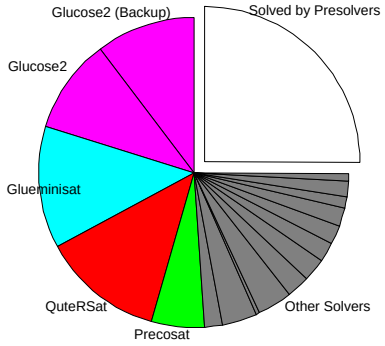
In this work, we investigated the question of assessing the contributions of individual SAT solvers by examining their value to SATzilla, a portfolio-based algorithm selector. SATzilla 2011 is an improved version of this procedure based on cost-based decision forests, which we entered into the new analysis track of the 2011 SAT competition. Its automatically generated portfolios achieved state of the art performance across all competition categories, and consistently outperformed its constituent solvers, other competition entrants, and our previous version of SATzilla. We observed that the frequency with which a component solver was selected is a poor measure of that solver's contribution to SATzilla 2011's



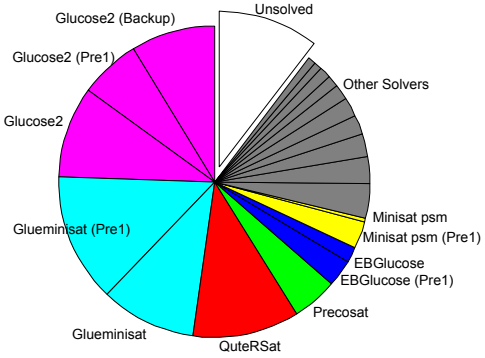
(a) Percent of solvable instances solved by each component solver alone



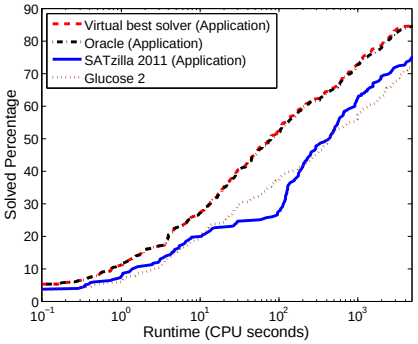
(b) Correlation matrix of solver runtimes (darker = more correlated)



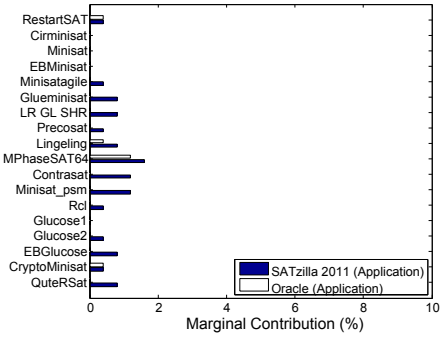
(c) Solver selection frequencies for `SATzilla 2011`; names only shown for solvers picked for $> 5\%$ of the instances



(d) Fraction of instances solved by `SATzilla 2011` pre-solver(s), backup solver, and main stage solvers



(e) Runtime CDF for `SATzilla 2011`, `VBS`, `Oracle`, and the gold medalist `Glucose2`



(f) Marginal contribution for `Oracle` and `SATzilla 2011`

Fig. 3. Visualization of results for category `Application`. The data underlying this figure can be found in Table 2 of the supplemental material, available at `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/SAT12-EvaluatingSolverContributions.pdf`.

performance. Instead, we advocate assessing solvers in terms of their marginal contributions to the state of the art in SAT solving.

We found that the solvers with the largest marginal contributions to `SATzilla` were often not competition winners (*e.g.*, `MPhaseSAT64` in `Application` SAT+UNSAT; `Sol` in `Crafted` SAT+UNSAT). To encourage improvements to the state of the art in SAT solving and taking into account the practical effectiveness of portfolio-based approaches, we suggest rethinking the way future SAT competitions are conducted. In particular, we suggest that all solvers that are able to solve instances not solved by any other entrant pass Phase 1 of the competition, and that solvers contributing most to the best-performing portfolio-based approaches be given formal recognition, for example by means of "portfolio contributor" or "uniqueness" medals. We also recommend that portfolio-based solvers be evaluated separately—and with access to all submitted solvers as components—rather than competing with traditional solvers. We hope that our analysis serves as an encouragement to the community to focus on creative approaches to SAT solving that complement the strengths of existing solvers, even though they may (at least initially) be effective only on certain classes of instances.

Acknowledgments This research was supported by NSERC and Compute Canada / Westgrid. We thank Daniel Le Berre and Olivier Roussel for organizing the new SAT competition analysis track, for providing us with competition data and for running our feature computation code.

References

1. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of DAC-99*, pages 317–320, 1999.
2. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
3. T. Carchrae and J. C. Beck. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence*, 21(4):372–387, 2005.
4. J. M. Crawford and A. B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proc. of AAAI-94*, pages 1092–1097, 1994.
5. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. of SAT-04*, volume 2919, pages 502–518, 2004.
6. M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, 2007.
7. C. Gebruers, B. Hnich, D. Bridge, and E. Freuder. Using CBR to select solution strategies in constraint programming. In *Proc. of ICCBR-05*, pages 222–236, 2005.
8. A. Van Gelder, D. Le Berre, A. Biere, O. Kullmann, and L. Simon. Purse-based scoring for comparison of exponential-time programs. In *Proc. of SAT-05*, 2005.
9. C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.
10. A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In *Proc. of ECAI-04*, pages 475–479, 2004.
11. M. Helmert, G. Róger, and E. Karpas. Fast downward stone soup: A baseline for building planner portfolios. In *Proc. of ICAPS-PAL-11*, pages 28–35, 2011.

12. E. Horvitz, Y. Ruan, C. P. Gomes, H. Kautz, B. Selman, and D. M. Chickering. A Bayesian approach to tackling hard computational problems. In *Proc. of UAI-01*, pages 235–244, 2001.
13. B.A. Huberman, R.M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 265:51–54, 1997.
14. S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Algorithm selection and scheduling. In *Proc. of CP-11*, pages 454–469, 2011.
15. Henry A. Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Proc. of IJCAI-99*, pages 318–325, 1999.
16. M. G. Lagoudakis and M. L. Littman. Learning to select branching rules in the DPLL procedure for satisfiability. In *Electronic Notes in Discrete Mathematics*, pages 344–359, 2001.
17. D. Le Berre, O. Roussel, and L. Simon. The international SAT Competitions web page. `www.satcompetition.org`, 2012. Last visited on Jan 29, 2012.
18. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proc. of IJCAI-03*, pages 1542–1543, 2003.
19. E. Nudelman, K. Leyton-Brown, A. Devkar, Y. Shoham, and H. Hoos. Satzilla: An algorithm portfolio for SAT. Solver description, SAT competition 2004, 2004.
20. M. Petri and S. Zilberstein. Learning parallel portfolios of algorithms. *Annals of Mathematics and Artificial Intelligence*, 48(1-2):85–106, 2006.
21. J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.
22. O. Roussel. Description of ppfolio. `www.cril.univ-artois.fr/~roussel/ppfolio/solver1.pdf`, 2011. Solver description, last visited on May 1, 2012.
23. H. Samulowitz and R. Memisevic. Learning to solve QBF. In *Proc. of AAAI-07*, pages 255–260, 2007.
24. K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1), 2008.
25. P. Stephan, R. Brayton, and A. Sangiovanni-Vencentelli. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:1167–1176, 1996.
26. D. Stern, R. Herbrich, T. Graepel, H. Samulowitz, L. Pulina, and A. Tacchella. Collaborative expert portfolio management. In *Proc. of AAAI-10*, pages 210–216, 2010.
27. M. J. Streeter and S. F. Smith. New techniques for algorithm portfolio design. In *Proc. of UAI-08*, pages 519–527, 2008.
28. G. Sutcliffe and C. B. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence Journal*, 131(1-2):39–54, 2001.
29. K. M. Ting. An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):659–665, 2002.
30. A. van Gelder. Another look at graph coloring via propositional satisfiability. In *Proc. of COLOR-02*, pages 48–54, 2002.
31. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, June 2008.
32. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proc. of IJCAI-RCRA-11*, 2011.

Supplementary Material

	Solver	Indiv. Perform. Runtime (solved)	Average Correlation	Used as Backup (Solved)	Used as Pre (Solved)	Picked by Model (Solved)	Marg. contrib. SATzilla(Oracle)
	EagleUP	1761 (66.7%)	0.70	- (-)	10/10 (47.6%)	4.5% (3.7%)	2.2% (0.2%)
R	Sparrow	1422 (73.6%)	0.67	- (-)	- (-)	20.3% (20.3%)	4.9% (2.4%)
A	March_rw	2714 (51.0%)	0.23	- (-)	- (-)	20.1% (19.9%)	0.0% (0.0%)
N	March_hi	2725 (51.0%)	0.23	- (-)	- (-)	5.3% (5.1%)	0.2% (0.0%)
D	Gnovelty+2	2146 (60.6%)	0.71	- (-)	- (-)	0.8% (0.8%)	0.4% (0.0%)
O	MPhaseSAT_M	1510 (73.0%)	0.67	- (-)	- (-)	0.4% (0.4%)	0.2% (0.0%)
M	Sattime11	1850 (67.9%)	0.70	- (-)	- (-)	0.4% (0.4%)	0.6% (0.0%)
	Adaptg2wsat11	1847 (66.7%)	0.70	- (-)	- (-)	0.4% (0.2%)	-0.4% (0.2%)
	TNM	1938 (65.9%)	0.70	- (-)	- (-)	0.2% (0.2%)	0.4% (0.0%)
	Sattime	2638 (49.3%)	0.39	- (-)	2/10 (6.4%)	15.5% (12.8%)	3.6% (1.4%)
	Sol	2563 (52.5%)	0.48	- (-)	1/10 (1.4%)	13.7% (13.7%)	8.1% (6.4%)
	Clasp2	2280 (67.4%)	0.69	- (-)	- (-)	17.4% (15.5%)	2.7% (0.4%)
	PicoSAT	2729 (54.8%)	0.73	- (-)	- (-)	10.1% (10.1%)	0.5% (0.4%)
C	Clasp1	2419 (67.4%)	0.67	- (-)	- (-)	7.8% (6.9%)	1.4% (1.4%)
R	QuteRSat	2793 (49.8%)	0.69	- (-)	- (-)	6.9% (6.4%)	1.4% (0.0%)
A	Sattime+	2681 (48.0%)	0.40	- (-)	- (-)	6.4% (5.9%)	1.4% (1.4%)
F	MPhaseSAT	2398 (59.7%)	0.62	- (-)	- (-)	5.0% (4.6%)	3.6% (1.8%)
T	CryptoMiniSat	2766 (49.8%)	0.68	- (-)	- (-)	3.2% (2.7%)	0.5% (0.0%)
E	RestartSAT	2773 (50.7%)	0.73	- (-)	- (-)	2.7% (1.4%)	1.4% (0.0%)
D	SApperloT	2798 (49.3%)	0.73	- (-)	- (-)	1.4% (1.4%)	1.8% (0.0%)
	Glucose2	2644 (56.6%)	0.66	- (-)	- (-)	0.5% (0.5%)	3.6% (0.0%)
	JMiniSat	3026 (44.3%)	0.74	- (-)	- (-)	0.5% (0.5%)	0.9% (0.0%)
	Minisat07	2738 (55.2%)	0.70	- (-)	- (-)	0.0% (0.0%)	0.0% (0.0%)
	Sathys	2955 (43.4%)	0.69	- (-)	- (-)	0.0% (0.0%)	0.0% (0.0%)
	Glucose2	1272 (85.0%)	0.86	10/10 (8.7%)	3/10 (6.3%)	9.9% (9.5%)	0.4% (0.0%)
	Glueminisat	1391 (83.4%)	0.86	- (-)	5/10 (13.4%)	12.7% (9.9%)	0.8% (0.0%)
	QuteRSat	1380 (81.4%)	0.80	- (-)	- (-)	12.7% (11.1%)	0.8% (0.0%)
A	Precosat	1411 (81.4%)	0.85	- (-)	- (-)	5.5% (4.7%)	0.4% (0.0%)
P	EBGlucose	1630 (78.7%)	0.87	- (-)	1/10 (2.8%)	1.9% (1.6%)	0.8% (0.0%)
P	CryptoMiniSat	1328 (81.8%)	0.82	- (-)	- (-)	3.6% (3.6%)	0.4% (0.4%)
L	Minisat psm	1564 (77.9%)	0.88	- (-)	1/10 (2.8%)	0.4% (0.4%)	1.2% (0.0%)
I	MPhaseSAT64	1529 (79.4%)	0.82	- (-)	- (-)	3.6% (2.8%)	1.6% (1.2%)
C	Lingeling	1355 (82.2%)	0.86	- (-)	- (-)	2.4% (2.4%)	0.8% (0.4%)
A	Contrasat	1592 (78.7%)	0.80	- (-)	- (-)	2.4% (2.0%)	1.2% (0.0%)
T	Minisat	1567 (76.7%)	0.88	- (-)	- (-)	2.0% (2.0%)	-0.4% (0.0%)
I	LR GL SHR	1667 (75.1%)	0.85	- (-)	- (-)	2.0% (1.6%)	0.8% (0.0%)
O	RestartSAT	1437 (79.4%)	0.88	- (-)	- (-)	1.9% (1.2%)	0.4% (0.4%)
N	Rcl	1752 (72.7%)	0.86	- (-)	- (-)	1.2% (1.2%)	0.4% (0.0%)
	MiniSATagile	1626 (74.7%))	0.87	- (-)	- (-)	1.6% (0.8%)	0.4% (0.0%)
	Cirminisat	1514 (79.8%)	0.88	- (-)	- (-)	0.8% (0.8%)	0.0% (0.0%)
	Glucose1	1614 (77.8%)	0.86	- (-)	- (-)	0.0% (0.0%)	0.0% (0.0%)
	EBMiniSAT	1552 (77.5%)	0.89	- (-)	- (-)	0.0% (0.0%)	0.0% (0.0%)

Table 2. *Performance of* `SATzilla 2011` *component solvers, disregarding instances that could not be solved by any component solver. We counted timed-out runs as 5 000 CPU seconds (the cutoff). Average correlation for s is the mean of Spearman correlation coefficients between s and all other solvers. Marginal contribution for s is negative if dropping s improved test set performance. (Usually,* `SATzilla`*'s solver subset selection avoids such solvers, but they can slip through when the training set is too small.)* `SATzilla 2011(Application)` *ran its backup solver* `Glucose2` *for 10.3% of the instances (and thereby solved 8.7%).* `SATzilla 2011` *only chose one presolver for all folds of* `Random` *and* `Application`*; for* `Crafted`*, it chose* `Sattime` *as the first presolver in 2 folds, and* `Sol` *as the second presolver in 1 of these; for the remaining 8 folds, it did not select presolvers.*