Boosting Sequential Solver Portfolios: Knowledge Sharing and Accuracy Prediction

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

[1] Dept. of Computer Science, Brown University, 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

Abstract. Sequential algorithm portfolios for satisfiability testing (SAT), such as `SATzilla` and `3S`, have enjoyed much success in the last decade. By leveraging the differing strengths of individual SAT solvers, portfolios employing older solvers have often fared as well or better than newly designed ones, in several categories of the annual SAT Competitions and Races. We propose two simple yet powerful techniques to further boost the performance of sequential portfolios, namely, a generic way of knowledge sharing suitable for sequential SAT solver schedules which is commonly employed in parallel SAT solvers, and a meta-level guardian classifier for judging whether to switch the main solver suggested by the portfolio with a recourse action solver. With these additions, we show that the performance of the sequential portfolio solver `3S`, which dominated other sequential categories but was ranked 10th in the application category of the 2011 SAT Competition, can be boosted significantly, bringing it just one instance short of matching the performance of the winning application track solver, while still outperforming all other solvers submitted to the crafted and random categories.

1 Introduction

Significant advances in solution techniques for propositional satisfiability testing, or SAT, in the past two decades have resulted in wide adoption of the SAT technology for solving problems from a variety of fields such as design automation, hardware and software verification, cryptography, electronic commerce, AI planning, and bioinformatics. This has also resulted in a wide array of challenging problem instances that continually keep pushing the design of better and faster SAT solvers to the next level. The annual SAT Competitions and SAT Races have played a key role in this advancement, posing as a challenge a set of so-called "application" category (previously known as the "industrial" category) instances, along with equally but differently challenging "crafted" and "random" instances.

Given the large diversity in the characteristics of problems as well as specific instances one would like to solve by translation to SAT, it is no surprise that different SAT solvers, some of which were designed with a specific set of application domains in mind, work better on different kinds of instances. Algorithm

portfolios [cf. 7] attempt to leverage this diversity by employing several individual solvers and, at runtime, dynamically selecting what appears to be the most promising solver — or a schedule of solvers — for the given instance. This has allowed sequential SAT portfolios such as `SATzilla` [16] and `3S` [8, 10] to perform very well in the annual SAT Competitions and Races.

Most of the state-of-the-art sequential algorithm portfolios are based on two main components: (a) a schedule of "short running" solvers to be run first in sequence for some small amount of time (usual some fixed percentage of the total available time such as 10%) and (b) a "long running" solver to be executed for the remainder of the time which is selected by one or the other Machine Learning technique (e.g., logistic regression, nearest neighbor search, or decision forest). If one of the short running solvers succeeds in solving the instance, then the portfolio terminates successfully. However, all work performed by each short running solver in this execution sequence is completely wasted unless it manages to fully solve the instance. If none of the short running solvers in the schedule succeeds, all faith is put in the one long running solver.

Given this typical sequential portfolio setup, it is natural to consider an extension that attempts to utilize information gained by short running solvers even if they all fail to solve the instance. Further, one may also consider an automated way to carefully revisit the choice of the long running solver whose improper selection may substantially harm the overall portfolio performance. We propose two relatively simple yet powerful techniques towards this end, namely, learnt clause forwarding and accuracy prediction.

We remark that one limitation of current algorithm portfolios is that their performance can never be better than that of the oracle or "virtual best solver" which, for each given instance, (magically) selects an individual solver that will perform best on it. By sharing knowledge, we allow portfolio solvers to, in principle, go beyond VBS performance. Specifically, a distinguishing strength of our proposed clause forwarding scheme is that it enables the portfolio solver to potentially succeed in solving an instance that no constituent SAT solver can.

Learnt clause forwarding focuses on avoiding waste of effort by the short running solvers in the schedule. We propose to share, or "forward," the knowledge gained by the first k solvers in the form of a selection of short learned clauses, which are passed on to the $k+1^{\text{st}}$ solver. Conflict-directed clause learning (CDCL) is a very powerful technique in SAT solving, often regarded as the single most important element that allows these solvers to tackle real-life problems with millions of variables and constraints. Forwarding learnt clauses is a cheap but promising way to share knowledge *between solvers* and is commonly employed in parallel SAT solving. We demonstrate that sharing learnt clauses can improve performance in sequential SAT solver portfolios as well.[3]

Accuracy prediction and recourse aims to use meta-level learning to correct errors made by the portfolio solver when selecting the "primary" or long running solver. Typically, effective schedules allocate a fairly large fraction of

[3] For the specific case of population-based algorithm portfolios, Peng et al. [11] have proposed sharing information through migration of individuals across populations.

the available runtime to one solver, as not doing so would limit the best-case performance of the portfolio to that of an oracle portfolio with a relatively short timeout. This, of course, poses a risk, as a substantial amount of time is wasted if the portfolio selects the "wrong" primary solver. We present a scheme to generate large amounts of training data from existing solver performance data, in order to create a machine learning model that aims to predict the accuracy of the portfolio's primary solver selector. We call this meta-level classifier as a *guardian classifier*. We also use this training data to determine the most promising *recourse action*, i.e., which solver should replace the suggested primary solver.

These techniques are general and may be applied to various portfolio algorithms. However, unlike the development of portfolio solvers that do not share information, experimentation in our setting is much more cumbersome and time consuming. It involves modifying individual SAT solvers and running the designed portfolio solver on each test instance in real time, rather than simply reading off performance numbers from a pre-computed runtime matrix.

We here demonstrate the effectiveness of our techniques using one base portfolio solver, namely `3S`, which had shown very good performance in SAT Competition 2011 in the crafted and random categories but was ranked 10^{th} in the application category. Note also that the instances and solvers participating in the 2011 Competition were designed with a 5,000 second time limit in mind, compared to instances and solvers in the 2012 Challenge where the time limit was only 900 seconds. Our focus is on utilizing algorithm portfolios and our techniques for solving hard instances. Our results, with a time limit roughly equivalent to 5,000 seconds of the competition machines, show that applying these techniques can boost the performance of `3S` on the 2011 competition instances to a point where it is only one instance short of matching the performance of the winning solver, Glucose 2.0 [1], on the 300 application track instances. Moreover, the resulting solver, `3S+fp`, continues to dominate all other solvers from the 2011 Competition in the crafted and random categories in which `3S` had excelled.

We note that our portfolio solver built using these techniques, called ISS or Industrial SAT Solver, was declared the Best Interacting Multi-Engine SAT Solver in the 2012 SAT Challenge, a category that specifically compared portfolios that share information among multiple SAT engines.

2 Background

We briefly review some essential concepts in constraint satisfaction, SAT, and portfolio research.

Definition 1. *Given a Boolean variable $X \in \{true, false\}$, we call X and $\neg X$ (speak: not X)* literals (over X). *Given literals $L_1, \ldots, L_k$ over Boolean variables $X_1, \ldots, X_n$, we call $(\bigvee_a L_a)$ a* clause (over variables $X_1, \ldots, X_n$). *Given clauses $C_1, \ldots, C_m$ over variables $X_1, \ldots, X_n$, we call $\bigwedge_a C_a$ a* formula in conjunctive normal form (CNF).

Definition 2. *Given Boolean variables $X_1, \ldots, X_n$, a* valuation *is an assignment of values "true" or "false" to each variable: $\sigma : \{X_1, \ldots, X_n\} \to \{\text{true}, \text{false}\}$. A literal X* evaluates *to "true" under σ iff $\sigma(X) = \text{true}$ (otherwise it evaluates to "false"). A literal $\neg X$* evaluates *to "true" under σ iff $\sigma(X) = \text{false}$. A clause C evaluates to true under σ iff at least one of its literals evaluates to "true." A formula evaluates to "true" under σ iff all its clauses evaluate to "true."*

Definition 3. *The* Boolean Satisfiability or SAT Problem *is to determine whether, for any given formula F in CNF, there exists a valuation σ such that F evaluates to "true."*

The SAT problem has played a prominent role in theoretical computer science where it was the first to be proven to be NP-hard [3]. At the same time, it has driven research in combinatorial problem solving for decades. Moreover, the SAT problem has great practical relevance in a variety of areas, in particular in cryptography and in verification.

2.1 SAT Solvers

While algorithmic approaches for SAT have been developed as early as the beginning of AI research, a boost in SAT solving performance has been achieved since the mid-nineties. Problems with a couple of hundred Boolean variables frequently posed a challenge back then. Today, many problems with hundreds of thousands of variables can be solved as a matter of course. While there exist very different algorithmic approaches to solving SAT problems, the performance of most systematic SAT solvers (i.e., those that can prove unsatisfiability) is frequently attributed to three ingredients:

1. Randomized search decisions and systematically restarting search when it exceeds some dynamic fail limit,
2. Very fast inference engines which only consider clauses which may actually allow us to infer a new Boolean variable for a variable, and
3. Conflict analysis and clause learning.

The last point regards the idea of inferring new clauses during search that are redundant to the given formula but encode, often in a succinct way, the reason why a certain partial truth assignment cannot be extended to any solution. These redundant constraints strengthen our inference algorithm when a different partial valuation cannot be extended to a full valuation that satisfies the given formula for a "similar" reason. One of the ideas that we pursue in this paper is to inform a solver about the clauses learnt by another solver that was invoked previously to try and solve the same CNF formula. This technique is standard in parallel SAT solving but, surprisingly, has not been considered for solver portfolios.

2.2 Solver Portfolios

Another important contribution was the inception of algorithm portfolios [4, 9, 15]. Based on the observation that solvers have complementary strengths and

thus exhibit incomparable behavior on different problem instances, the ideas of running multiple solvers in parallel or to select one solver based on the features of a given instance were introduced. Portfolio research has led to a wealth of different approaches and an amazing boost in solver performance in the past decade [8, 16].

Solver Selection: The challenge when devising a solver portfolio is to develop a learning algorithm that, for a given set of training instances, builds a dynamic mechanism that selects a "good" solver for any given SAT instance. To this end, we need a way to characterize a given SAT instance, which is achieved by computing so-called "features." These could be, e.g., the number of clauses or variables, statistics over the number of negated over positive variables per clause, or the clause over variable ratio. Features can also include dynamic properties of the given instance, obtained by running a solver for a very short period of time as a probe and collecting statistics. As the goal of this paper is to devise techniques to improve *existing* portfolios, a full understanding of instance features is unnecessary. We refer the reader to Xu et al. [16] for a comprehensive study of features suitable for SAT.

Solver Scheduling: Recent versions of `SATzilla` and `3S` no longer just choose one among the portfolio's constituent solvers. While still selecting one long running primary solver, they first schedule a sequence of several other solvers for a shorter amount of time. In particular, `3S`, our base solver for experimentation, employs a semi-static schedule of solvers, given a test instance F and an total time limit T. It runs a static schedule (independent of F, based solely on prior knowledge from training data) for an internal time limit t (with $t \approx 10\%$ of T) in which several different solvers with different (short) time limits are used. This is followed by a long running solver, scheduled for time $T - t$, based on the features of F computed at runtime.

We will refer to these two components of `3S`'s scheduling strategy as the *pre-schedule* and the *primary solver*. In this paper, we tackle precisely these two aspects: How can we improve the interplay between the short-running solvers in the pre-schedule while also passing knowledge on to the primary solver, and how can we improve the selection of the long-running primary solver itself.

3 Sharing Knowledge Among Solvers

A motivating factor behind the use of a pre-schedule used in sequential portfolios is *diversity*. By employing very different search strategies, one increases the likelihood of covering instances that may be challenging for some solvers and very easy for others. Diversity has also been an important factor in the design of *parallel* SAT solvers, such as `ManySAT` [6] and `Plingeling` [2]. When designing these parallel solvers, it has been observed that the overall performance can be improved by carefully sharing a limited amount of knowledge between the

search efforts led by different threads. This knowledge sharing must be carefully done, as it must balance usefulness of the information against the effort of communicating and incorporating it. One effective strategy has been to share information in the form of very short learned clauses, often just unit clauses, i.e., clauses with only one literal (e.g., the winning parallel solver [2] at the 2011 SAT Competition).

3.1 Knowledge Sharing Among Clause-Learning Systematic Solvers

In contrast, current sequential portfolios, while also relying on diversity through the use of a pre-schedule, do not exploit any kind of knowledge sharing. If the first k solvers in the pre-schedule fail to solve the instance, the time they spent is wasted. We propose to avoid this waste by employing the same technique that is used in parallel SAT, namely by *forwarding* a subset of the clauses learned by one solver in the pre-schedule to all solvers that follow it. In our implementation, clause forwarding is parameterized by two positive integer parameters, L and M. Each clause forwarding solver outputs all learned clauses containing up to L literals. Out of this list, the M shortest ones (or fewer, if not enough such clauses are generated) are forwarded to the next solver in the schedule, which then treats these clauses as part of the input formula. While we solely base our choice on what clauses to forward on their lengths, one could also consider more sophisticated measures (e.g., [1]). Note that, unlike clause sharing in today's parallel SAT solvers, in the sequential case clause forwarding incurs a relatively low communication overhead. Nonetheless, it needs to be balanced out with the potential benefits. We implemented clause forwarding in three conflict directed clause learning (CDCL) solvers, henceforth referred to as the *clause forwarding solvers*.

3.2 Impact of Knowledge Sharing on Other Solvers

In addition to CDCL solvers, pre-schedules typically also employ two other kinds of solvers: incomplete local search solvers and "lookahead" based complete solvers. The former usually perform very well on random and some crafted instances, and the latter usually excel in the crafted category and sometimes on unsatisfiable random instances. Since these solvers are not designed to generate or use conflict directed learned clauses, it is not clear *a priori* whether such clauses — which are redundant with respect to the underlying SAT theory — would help these two kinds of solvers as well. In our experiments, we found it best to run these solvers *before* our clause forwarding solvers are used.

The exceptions to this rule were two solvers: `march_hi` and `mxc-sat09`, which showed a mixed impact of incorporating forwarded learned clauses. We thus chose to run them both before the forwarding solvers, as in the base portfolio `3S`, and also after forwarding. Our overall pre-schedule was composed of the original one used by `3S` in the 2011 SAT Competition, scaled appropriately to take the difference in machine speeds into account, enhanced with clause forwarding solvers, and reordered to have non-forwarding CDCL solvers appear after the

Table 1. Gap closed to the virtual best solver (VBS) by using clause forwarding.

	2009	**2010**	**2011**	**Average**
% closed over `3S` / VBS gap	12.5	16.67	5.41	11.53

forwarding ones. We note that changing the pre-schedule itself did not significantly alter the performance of `3S`. E.g., in the application category, as we will later see in Table 3, the performances of `3S` with the original and the updated pre-schedules were very similar.

3.3 Formula Simplification

One other consideration that has a significant impact in practice is, whether to simplify the CNF formula before handing it to the next solver in the schedule, after (up to) M forwarded clauses have been added to it. With some experimentation, we found that minimal simplification of the formula after adding forwarded clauses, performed using `SatElite` [13] in our case, was the most rewarding. We thus used this clause forwarding setup for the experiments reported in this paper.

3.4 Practical Impact of Clause Forwarding

We will demonstrate in the experiments section that clause forwarding allows `3S` to close a significant part of the gap in performance when compared to the best solvers for application instances of the 2011 Competition, along with more information on the choice of training/test splits we consider and the experimental setup we use. We here provide an additional preview of the impact of clause forwarding when using the latest SAT solver available prior to the 2012 Challenge. For this evaluation we consider three train/test splits of instances: the first split uses the 2009 competition instances as test instances and every instance available before 2009 for training; the second and third split are defined similarly but for the 2010 race and 2011 competition, respectively.

Results are presented in Table 1. Here, we consider the *gap in performance* between the portfolio without clause forwarding and the best possible no-knowledge-sharing portfolio, VBS, which uses an oracle to invoke the fastest solver for each given instance. In the table, we show how much of that gap is closed by using clause forwarding. Of course, the portfolio that uses knowledge sharing between solvers is no longer limited in performance by the oracle portfolio, as remarked earlier. However, using the oracle portfolio gives us a good baseline to compare with. As we see, clause forwarding significantly helps on all three competition splits clause. On average, using this technique we are able to close over 10% of the gap between the pure portfolio and the oracle portfolio.

4 Accuracy Prediction and Recourse

Studying the results of the SAT Competition 2011 one can observe that the best sequential portfolio, `3S` only solved 200 out of 300 instances in the application category. However, when analyzing the performance of the solvers the `3S` portfolio is composed of, one can also see that the virtual best solver (VBS) based on those solvers can actually solve more than 220 application instances. Hence, the suggestions made by the portfolio are clearly wrong in more than 10% of all cases. The objective of this section is to lower this performance gap. In the following we first try to determine when the suggestion of a portfolio results in a loss in performance, and second what to do when we believe the portfolio's choice is wrong.

4.1 Accuracy Prediction

One way to potentially improve performance would be to improve the portfolio selector itself (e.g., by multi-class learning). Nonetheless, most classifiers often cannot represent exactly the concept class they are used for. One standard way out in machine learning is to conduct classification in multiple stages, which is what we consider here. Basic classifiers providing a confidence or trust value can function as their own guardian. In Ensemble Learning, more complex recourse classifiers are considered. Our goal here is to design such an approach in the specific context of machine learning methods for SAT solver selection.

We propose a two-stage approach where we augment the existing SAT portfolio classifier by accompanying it with a "guardian" classifier, which aims to predict when the first classifier errs, and a second "selector" classifier that selects an alternative solver whenever the guardian finds that the first selector is probably not right.

To train a guardian and a replacement selector classifier, we first need to capture some characteristics that correlate with the quality of the decision of the portfolio. To that end we propose to create a set of features and label the portfolio's suggestion as "good" or "bad" ($\mathcal{L} = \{\text{good}, \text{bad}\}$). A key question is, how should these two labels be defined. Inspired by the SAT competition context, a "good" decision will be defined as one where an instance can be solved within the given time limit and a "bad" one is when it cannot be.[4]

The definition of a feature vector $\boldsymbol{f}$ to use for a guardian classifier is unfortunately far less straightforward. We, of course, first tried the original features used by `3S` but that did not result in an overall improvement in performance. As is typically done in machine learning, we experimented with a few additions and variations, and settled on the following:

[4] We also tried labels that identify top performer (e.g., not more than x% slower than the best solver, for various x), but obtained much worse results. The issue here is that it is more ambitious than necessary to predict which solver is best or close to best. Instead, we need to be able to distinguish solvers that are good enough from those that fail. That is, rather than aiming for speed, we optimize for solver robustness.

List of employed features	
F1	Distance to closest cluster center
F2	k used for test instance
F3-F7	Min/Max/Average/Median/Variance of distance to closest cluster center
F8	Solver ID selected by k-NN
F9	Solver type: incomplete or complete
F10	Average Distance to solved instances by top-2 solvers
F11	VBS time on k-neighborhood of test instance
F12	Number of instances solved by top-5 ranked solvers
F13-F23	PAR10 score/instances solved by top-5 ranked solvers
F24-F34	10 test instance features

Table 2. Description of features used by guardian classifier. Solver rank is based on average PAR10 score on neighborhood.

We selected 34 features composed of: the first 10 features of the test instance, the Euclidean based distance measures of training instances in the neighborhood to the test instance, and runtime measures of the five best solvers on a restricted neighborhood (see Table 4.1 for details). These features are inspired by the k-nearest-neighbor classifier that `3S` employs.

Consequently, for the guardian we need to learn a classifier function: $\boldsymbol{f} \longmapsto \mathcal{L}$. To this end we require training data. The `3S` portfolio is based on data T that is composed of features of and runtimes on $5,467$ SAT instances appearing in earlier competitions. We can split T into a training set T_{train} and test set T_{test}. Now, we can run the portfolio restricting its knowledge base to T_{train} and test its performance on T_{test}. For each test instance $i \in T_{test}$ we can compute the corresponding feature vector $\boldsymbol{f_i}$ and obtain the label $\mathcal{L}_\rangle$. Hence, the number of training instances we obtain for the classifier is i. Obviously, one can split T differently over and over by random subsampling, and each time one creates new training data to train the "guardian" classifier.

The question arises whether different splits will not merely regenerate existing knowledge. This depends on the features chosen, but here the feature vector will actually have a high probability to be different for each single split since in each split the neighborhood of a test instance will be different. A thought experiment that makes this more apparent is the following: Assume that, for a single instance i, we sort all other instances according to the distance to i (neighborhood of i). Assume further we select training instances from the neighborhood of i with probability $1/k$ until we have selected k instances (where k is the desired neighborhood size). When $k > 10$ it is obviously very unlikely for an instance to have exactly the same neighbors.

In order to determine an appropriate amount of training data we first randomly split the data set T in a training split $T_{train'}$ and test split $T_{test'}$, before generating the data for the classifier. We then perform the aforementioned splitting to generate training data for the classifier on $T_{train'}$ and test it on the data generated by running k-NN with data $T_{train'}$ on the test set $T_{test'}$. We use 10

different random splits of type $T_{train'}$ and $T_{test'}$ and try to determine the best number of splits for generating training data for the classifier.

While normally one could essentially look at the plain accuracy of the classifier and select the number of splits that result in the highest accuracy, we propose to employ another measure based on the following reasoning. The classifier's "confusion matrix" looks in our context like this (denoting the solver that was selected by the portfolio on instance I with S):

(a) S solves I and classifier predicts that it can
(b) S solves I , but classifier predicts that it cannot
(c) S can't solve I, but classifier predicts that it can
(d) S can't solve I, and classifier predicts that it cannot

Instances that fall in category (a) reflect a "good" choice by the portfolio (our original selector) and, while correctly detected, there is also nothing for us to gain. In case (c) we cannot exploit the wrong choice of the portfolio since the guardian classifier does not detect it. However, we will also not degenerate the performance of the portfolio. Case (b) and (d) are the interesting cases. In (b) we collect the false-positives where the classifier predicts that the portfolio's choice was wrong while it was not. Consequently it could be the case that we degrade the performance of the original portfolio selector by altering its decision. All instances falling in category (d) represent the correctly labeled decisions of the primary selector that should be overturned. In (d) lies the *potential* of our method: all instances that fall in this category cannot be solved by solver S that the primary selector chose, and the guardian classifier correctly detected it. Since cases (a) and (c) are somewhat irrelevant to any potential recourse action, we focus on keeping the ratio $\frac{(b)}{(d)}$ as small as possible in order to favorably balance potential losses and wins. Based on this quality measure we determined that roughly 100 splits achieve the most favorable trade off on our data.

4.2 Recourse

When the guardian classifier triggers, we need to select an alternative solver. For this purpose we need to devise a second "recourse" classifier. While we clearly do not want to select the same solver that was suggested by the original portfolio selector, the choices for possible recourse actions is vast and their benefits hardly apparent. We introduce the following recourse strategy:

Since we want to replace the suggested solver S, we assume S is not suitable for the given test instance I. Based on this *conditional probability* we can also infer that the instances solved by S in the neighborhood of size k of I can be removed from its neighborhood. Now, it can be the case that the entire neighborhood of I can be solved by S and therefore we extend the size of the neighborhood by 30%. If on this extended neighborhood S cannot solve all instances, we choose the solver with the lowest PAR10-score on the instances in the extended neighborhood not solved by S. Otherwise, we choose the solver with the second best ranking by the original portfolio selector. In the context of

3S this is the solver that has the second lowest PAR10-score on the neighborhood of the test instance.

Designing a good recourse strategy poses a challenge. As we will see later in Section 5.3, our proposed recourse strategy resulted in solving 209 instances on the 2011 SAT Competition application benchmark, compared to the 204 that 3S solved. We tried a few other simpler strategies as well, which did not fare as well. We briefly mention them here: First, we used the solver that has the second best ranking in terms of the original classifier. For 3S this means choosing the solver with the second lowest PAR10-score on the neighborhood of the test instance. This showed only a marginal improvement, solving 206 instances. We then tried to leverage diversity by mixing-and-matching the two recourse strategies mentioned above, giving each exactly half the remaining time. This resulted in overall performance to drop below 3S without accuracy prediction. Finally, we computed offline a static replacement map that, for each solver S, specifies one fixed solver $f(S)$ that works the best across all training data whenever S is selected by the original classifier but does not solve the instance. This static, feature-independent strategy also resulted in degrading performance. For the rest of this paper, we will not consider these alternative replacement strategies.

5 Empirical Evaluation

In order to evaluate the impact of our two proposed techniques on an existing portfolio solver, we applied them to 3S [8], the best performing sequential portfolio solver at the 2011 SAT Competition.[5] We refer to the resulting enhanced portfolio solver as 3S+f when clause forwarding is used, as 3S+p when accuracy prediction and recourse classifiers are used, and as 3S+fp when both new techniques are applied. We compare their performance to the original 3S, which was the winner in the crafted and random categories of the main sequential track of the 2011 SAT Competition.

As remarked earlier, our techniques are by no means limited to 3S and may be applied to more recent portfolios. However, these techniques are likely to pay off more on harder instances and thus we focus here on the 2011 Competition in which both instance selection and solver design was done with a 5,000 seconds time limit in mind.

For evaluation, we use the 2011 competition split, i.e., we use the same application (300), crafted (300), and random (600) category instances as the ones used in the main phase of the Competition. The enhanced variants of 3S rely only on the pre-2011 training data that comes with the original 3S. We note that we did conduct experiments using random splits after mixing all instances, but there the performance of the original k-NN classifier of 3S is typically almost perfect, leaving little to no room for improvement. Competition splits exhibit a completely different and perhaps arguably more realistic behavior, as the suboptimal performance of 3S in the application category shows. We thus focused

[5] The source code of 3S can be obtained from http://www.satcompetition.org/

on splits that were neither random nor hand-crafted by us and experimented on competition splits to evaluate the techniques.

All experiments were conducted on 2.3 GHz AMD Opteron 6134 machines with 8 4-core CPUs and 64 GB memory, running Scientific Linux release 6.1. We used a time limit of 6,500 sec, which roughly matched the 5,000 sec timeout that was used on the 2011 Competition machines. As performance measures we consider the number of instances solved, average runtime, and PAR10 score. PAR10 stands for penalized average runtime, where instances that time out are penalized with 10 times the timeout.

5.1 Implementation Details on Clause Forwarding

We implemented learned clause forwarding in three CDCL SAT solvers that were used by `3S` in the 2011 Competition: `CryptoMiniSat 2.9.0` [12], `Glucose 1.0` [1], and `MiniSat 2.2.0` [14]. The pre-schedule was modified to prolong the time these three clause-learning solvers are run, as discussed earlier. With clause forwarding disabled, `3S` with this modified pre-schedule resulted in roughly the same performance on our testbed as `3S` with the original pre-schedule used in the Competition (henceforth referred to as `3S-C`). In other words, any performance differences we observe can be attributed to clause forwarding and accuracy prediction and recourse, not to the change in the pre-schedule itself.

For clause forwarding, we used parameter values $L = 10$ and $M = 10,000$, i.e., each of the three solvers may share up to 10,000 clauses of size up to 10 for the next solver to be run. The maximum amount of clauses shared is therefore 30,000. We note that these parameters are by no means optimized. Among other variations, we tried sharing an unlimited number of (small) clauses, but this un-surprisingly degraded performance. We expect that these parameters can be tuned better. Nevertheless, the above choices worked well enough to demonstrate the benefits of clause sharing, which is the main purpose of this experimentation.

5.2 Implementation Details on Accuracy Prediction

To predict how good the primary solver suggested by `3S` is likely to be, we experimented with several classifiers available in the Weka 3.7.5 data mining and machine learning Java library [5].[6] The results presented here are for the REP-Tree classifier, which is a fast decision tree learner that uses information gain and variance reduction for building the tree, and applies reduced-error pruning. Using training data based on splitting $5,464$ pre-2011 instances (the ones `3S` is based on) 100 times, as described earlier, we trained a REP-Tree and obtained the following confusion matrix for instances[7] of the 2011 SAT Competition application test data:

[6] `http://www.cs.waikato.ac.nz/ml/weka/`

[7] Note that the numbers do not add up to 300 since, with the classifier, we only consider instances that have not been solved yet by the pre-scheduler and can be solved by at least one of the solvers in our portfolio.

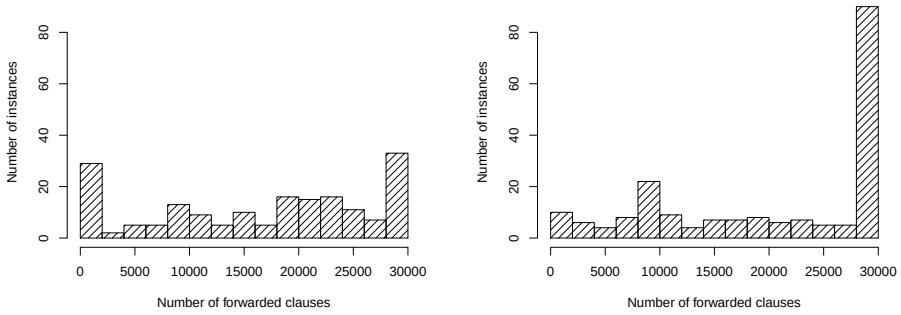

Fig. 1. Histogram showing how often N clauses are forwarded. Left: Crafted instances. Right: Application instances.

(a) 61	(b) 7
(c) 25	(d) 14

Hence, the best possible outcome for a recourse action would be to solve the previously unsolved 14 instances ($\approx 5\%$ of all the 2011 application instances) under (c) and to still be able to solve the 7 instances ($\approx 2\%$) under (b). While in the best case we could gain 14 instances and lose none, it is obviously not clear whether one would achieve *any* gain at all, or even solve at least the 7 instances that originally used to be solved. Fortunately, with our recourse strategy, we witness a significant gain in overall performance. We integrated the classifier in `3S` in the following way: When `3S` suggests the primary solver, if indicated by our guardian REP-Tree model, we intercept its decision and alter it as proposed by our recourse strategy.

5.3 Results on 2011 SAT Competition Data

Since our base portfolio solver, `3S`, already works best on random on crafted instances considered, the objective is to close the large gap between the best sequential portfolios and the best individual solvers in the application track, while not degrading the performance of the portfolio on crafted and random categories.

To this end, let us first note that adding the methods proposed in this paper have no significant impact on `3S` performance on random and crafted instances. On random instances, knowledge sharing hardly takes place since CDCL based complete solvers are barely able to learn any short clauses on these instances.

For crafted instances, a limited amount of clause forwarding does happen, but much less so than in application instances. In Figure 1 we show how many instances in out test set share how many clauses. On the left we see that, on crafted instances, we mostly share a modest amount of clauses between solvers, if any. The plot on the right shows the situation for application instances. Here it is usually the case that the solvers share the fully allowed 30,000 clauses.

Table 3. Performance Comparison of `3S-C` from the competition and its four new variants: `3S`, `3S+f`, `3S+p`, and `3S+fp` on application.

Application	`3S-C`	`3S`	`3S+f`	`3S+p`	`3S+fp`
# Solved	205	204	213	209	214
# Unsolved	95	96	87	91	86
% Solved	68.3	68.0	71.0	69.7	71.3
Avg Runtime	2,764	2,744	2,537	2,707	2,524
PAR10 Score	22,676	22,311	20,693	21,437	20,485

Interestingly, the clause sharing in crafted instance causes a slight decrease in performance, but this is outweighed by the positive impact of our prediction and recourse classifiers which actually improve the performance of the solver presented here over `3S` on crafted instances. In summary, the solver presented here works as well as `3S` on random instances, and insignificantly better than `3S` on crafted instances.

It remains to test if the methods proposed here can boost `3S` performance to a point where it is competitive on application instances as well.

Table 3, column 1, shows the performance of the `3S` version available from the competition website (`3S-C`). The number of instances it solves here differs slightly from the competition results due to hardware and experimental differences. Comparing `3S-C` with `3S` (where we changed the pre-schedule to allow more clauses to be learned), we observe that the difference is very small. `3S-C` solves just 1 instance more than `3S`, letting us conclude that the subsequently reported performance gains are not due to differences in the pre-schedule itself.

Both `3S+f` (`3S` with clause forwarding) and `3S+p` (`3S` with prediction and recourse) are able to improve on `3S` in a significant fashion: `3S+f` is able to solve 9 more instances, and `3S+p` solves 5 more. Note that in the application category it is usually the case that the winning solver only solves a couple of more instances than the second-ranked solver. Indeed, the difference between the 10-th ranked `3S` in the competition and the winner was only 15 instances. That is to say, prediction and recourse closes 33% of the gap to the winning solver, and clause forwarding even over 60%.

The combination of clause forwarding and prediction and recourse in `3S+fp` is able to solve 214 instances. This is just one instance shy of the best sequential solver `Glucose 2.0` at the 2011 SAT Competition which we re-ran on our hardware using the same experimental settings. Note that `3S` uses only pre-2011 solvers. Furthermore, we found that the average runtime of `3S+fp` is close to `Glucose 2.0` as well, which also indicates that in contexts where objectives other than the number of solved instances are of interest, `3S+fp` is very competitive.

6 Conclusion

We presented two novel generic techniques for boosting the performance of SAT portfolios. The first approach shares the knowledge discovered by SAT solvers that run in sequence, while the second improves solver selection accuracy by detecting when a selection is likely to be inferior and proposing a more promising recourse selection. Applying these generic techniques to the SAT portfolio `3S` resulted in significantly better performance on application instances while not reducing performance on crafted and random categories, making the resulting solver, `3S+fp` excel on all categories in our evaluation using the 2011 SAT Competition data and solvers.

References

[1] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *21st IJCAI*, pp. 399–404, Pasadena, CA, July 2009.

[2] A. Biere. Plingeling: Solver description, 2010. SAT Race.

[3] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pp. 151–158. ACM, 1971.

[4] C. P. Gomes and B. Selman. Algorithm portfolios. *AI J.*, 126(1-2):43–62, 2001.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An update. *SIGKDD Explorations*, 11(1), 2009.

[6] Y. Hamadi and L. Sais. ManySAT: a parallel SAT solver. *JSAT*, 6, 2009.

[7] J.R.Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[8] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Algorithm selection and scheduling. In *17th CP*, vol. 6876 of *LNCS*, pp. 454–469, Perugia, Italy, Sept. 2011.

[9] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *IJCAI*, pp. 1542–1543, 2003.

[10] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Non-model-based algorithm portfolios for SAT. In *14th SAT*, vol. 6695 of *LNCS*, pp. 369–370, Ann Arbor, MI, June 2011.

[11] F. Peng, K. Tang, G. Chen, and X. Yao. Population-based algorithm portfolios for numerical optimization. *IEEE Transactions on Evolutionary Computation*, 14 (5):782–800, Oct. 2010.

[12] M. Soos. CryptoMiniSat 2.9.0, 2010. *http://www.msoos.org/cryptominisat2*.

[13] N. Sorensson and N. Een. SatELite 1.0, 2005. *http://minisat.se*.

[14] N. Sorensson and N. Een. MiniSAT 2.2.0, 2010. *http://minisat.se*.

[15] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla-07: The design and analysis of an algorithm portfolio for sat. In *CP*, pp. 712–727, 2007.

[16] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *JAIR*, 32(1):565–606, 2008.