A Comprehensive Study of k-Portfolios of Recent SAT Solvers

Jakob Bach
Karlsruhe Institute of Technology (KIT), Germany

Markus Iser
Karlsruhe Institute of Technology (KIT), Germany

Klemens Böhm
Karlsruhe Institute of Technology (KIT), Germany

Abstract

Hard combinatorial problems such as propositional satisfiability are ubiquitous. The holy grail are solution methods that show good performance on all problem instances. However, new approaches emerge regularly, some of which are complementary to existing solvers in that they only run faster on some instances but not on many others. While portfolios, i.e., sets of solvers, have been touted as useful, putting together such portfolios also needs to be efficient. In particular, it remains an open question how well portfolios can exploit the complementarity of solvers. This paper features a comprehensive analysis of portfolios of recent SAT solvers, the ones from the SAT Competitions 2020 and 2021. We determine optimal portfolios with exact and approximate approaches and study the impact of portfolio size k on performance. We also investigate how effective off-the-shelf prediction models are for instance-specific solver recommendations. One result is that the portfolios found with an approximate approach are as good as the optimal solution in practice. We also observe that marginal returns decrease very quickly with larger k, and our prediction models do not give way to better performance beyond very small portfolio sizes.

2012 ACM Subject Classification Theory of computation → Logic and verification; Computing methodologies → Supervised learning; Theory of computation → Integer programming

Keywords and phrases Propositional satisfiability, solver portfolios, runtime prediction, machine learning, integer programming

Digital Object Identifier 10.4230/LIPIcs.SAT.2022.2

Supplementary Material We provide the source code and all experimental data:
Software (Source Code): `https://github.com/Jakob-Bach/Small-Portfolios`; archived at `swh:1:dir:ca0637d35eaf25019fa40792bbcdb4ea930c3b7a`
Dataset (Experimental Data): `https://doi.org/10.5445/IR/1000146629`

Acknowledgements We want to thank Luc Mercatoris for his support with preliminary experiments.

1 Introduction

1.1 Motivation

Propositional satisfiability (SAT) solving is the archetypal NP-complete problem. Its practical applications abound, e.g., verification of hardware and software [26, 10], product configuration [23], cryptanalysis [33], or planning [42]. SAT solvers are not only used to solve hard combinatorial problems in industry but also previously open problems in mathematics [21, 20].

Nowadays, one can observe continuous progress regarding SAT-solving methods, heuristics, and their implementations in SAT solvers. Evaluation of solvers is commonly based on compilations of benchmark instances that represent diverse, interesting application scenarios. An important design objective behind new SAT solvers is stability, i.e., good performance on many types of instances. At the same time, we observe a recurrent emergence of new

heuristics and methods that improve performance on only a narrow subset of instances. Nevertheless, such approaches have merit – one can see them as complementary to solvers with good average behavior.

One way to leverage the complementarity of solvers is with portfolios, i.e., sets of solvers. One can combine a solver portfolio with a perfect oracle that selects the fastest solver from the portfolio for each instance in the benchmark set. This construction is commonly referred to as the *Virtual Best Solver* (VBS). One can also rank individual solvers based on their marginal contribution to the VBS [47].

▶ **Example 1** (Special Price in SAT Competition 2021)**.** In the SAT Competition 2021, the performance of the VBS of all portfolios of size two over the entirety of participating solvers has been evaluated. The best pair of solvers in terms of their VBS contains the solver *CaDiCaL_PriPro* [7]. This solver has not been competitive in the overall evaluation. However, due to that achievement, it has been awarded the new *Special Innovation Price*.

While the VBS is a theoretical construct, one might use prediction models to select solvers from a portfolio in practice. Solver selection that includes runtime predictions usually outperforms individual best solvers by much [46]. However, the relative performance gains of such approaches on instances in the very diverse *application* category are usually much smaller than the gains in the categories of *crafted* or *randomly generated* instances [47, 13]. We argue that evaluations confined to only a few, very specific sets of instances, such as the one in [25], let the solvers appear in a better light than ones covering various domains. Moreover, constructing a prediction-based solver-selection process is complex and time-consuming, particularly for preparing the necessary training data. In consequence, more often than not, comparative studies in SAT solving do not include implementations applying this principle to state-of-the-art solvers and recent benchmarks of interest.

Given the circumstances sketched so far, the following questions are essential when it comes to the design of portfolios and their evaluation. First, one must decide how many and which solvers to put in a portfolio. The *K-Portfolio Problem* is to determine an optimal k-portfolio (cf. Section 4). Searching for a solution to the *K-Portfolio Problem* is an NP-complete optimization problem [36]. Second, one must decide how to select a solver for each instance. In particular, this includes choosing a type of prediction model and instance features used to learn the model.

1.2 Contributions

This current article features a study that captures and evaluates the state-of-the-art in SAT solving in terms of portfolio performance. Our study is unique in that it addresses all points raised so far, as follows: We systematically construct k-portfolios and analyze the impact of the portfolio size k on portfolio performance. For exact search, we encode the *K-Portfolio Problem* as an integer optimization problem (which is faster than alternative approaches, according to preliminary experiments of ours). Spurred by the complexity of the problem, we also analyze the quality of various approximate and random solutions.

In a subsequent step, we analyze the quality of solver runtime prediction that one can achieve with off-the-shelf prediction models and instance features. For each portfolio size k, we train random forests and boosted trees as prediction models, using the widely known instance feature records of SATzilla 2012 [48]. Next, we apply these models to select portfolio members automatically for each instance.

Our study is based on very recent *competition* datasets, which contain the runtimes of solvers having participated in the SAT Competitions 2020 and 2021. The instances in these SAT competitions were selected by stratified random sampling from various instance

families of interest, considering the categories *application* and *crafted* [16]. With a few exceptions (cf. Example 1), solvers submitted to such competitions are tailored for good overall performance. Our study highlights the extent to which solvers entering current SAT competitions are still complementary and if one can leverage this complementarity with prediction models for solver selection. Our code and data are available online (cf. Section 5.4).

1.3 Results

Our evaluation yields insightful results regarding the various questions. While the marginal utility is expected to decrease with portfolio size, we did not expect the decrease to be so early. Regarding the VBS, portfolios of size five already achieve more than 70% of the maximum possible performance gain over the single best solver, so marginal performance gains from adding single additional solvers become very small in each case. Another interesting observation is that the well-known heuristic *beam search* has given way to good approximate solutions to the *K-Portfolio Problem*. These solutions perform very similarly to the optimal ones, even for a minimal beam width of one.

Relying on an off-the-shelf prediction model has been a mixed bag. On the negative side, prediction performance has been somewhat low. The actual runtime of model-based portfolios is not even close to the theoretical improvement one could have with a perfect oracle. We take this as an indication that more research is necessary; we hypothesize that the instance features available in our dataset are not sufficiently discriminative. On the positive side, we have observed considerable performance gains over single-best-solver performance with runtime prediction for 3-portfolios in both competition datasets. For the solvers in the SAT Competition 2021 dataset, we observe a further gain in performance for $k = 4$. For larger k, the runtimes of model-based portfolios do not improve further. We take this as a further indication that it might not be beneficial to have many solvers in a model-based portfolio.

1.4 Outline

In Section 2, we review related work. Preliminaries are given in Section 3. In Section 4, we introduce the *K-Portfolio Problem* and several solution approaches. We describe the experimental design in Section 5 and present the experimental results in Section 6. We conclude with Section 7.

2 Related Work

In this section, we provide an overview of approaches that deal with portfolios of SAT solvers. We begin with portfolios in general and continue with instance-specific solver selection, including approaches to configure and analyze such techniques automatically. Finally, we discuss related work on systematic investigations of k-portfolios.

In the context of algorithms, the term *portfolio* refers to a heterogeneous family of approaches [27]. Portfolio approaches have in common that they operate on a set of algorithms to exploit the presupposed complementary performance of different methods and strategies. Some sequential approaches generate schedules to interleave runs of different algorithms in a static [17] or dynamic [11, 44] fashion. Others use supervised learning for instance-specific algorithm selection (cf. SATzilla [46]) or rely on unsupervised methods to select the best algorithm configuration for a given instance (cf. ISAC [25]). Parallel approaches encompass *pure portfolios* (cf. PPfolio [24]) as well as *parallel portfolios with information sharing* (cf. ManySAT [19]). Our work considers portfolios from which exactly one solver is selected and run for each instance.

Solver complementarity regarding the runtime of solver portfolios has been examined by Xu et al. [47], who analyze the marginal contributions of solvers, and Fréchette et al. [15], who analyze the Shapley values of solvers. Both these analyses show that significant contributions to the runtime of a portfolio can come from solvers that are not competitive when evaluated as a standalone solver.

Instance-specific solver-selection approaches bring in another perspective on solver complementarity. One of the best-known complex solver-selection approaches is SATzilla, of which multiple versions exist [46, 48]. In SATzilla, multiple prediction models guide the selection of a solver from a portfolio. Additional components of SATzilla are *pre-solvers*, which are only run for a short time to solve easy instances, and a *backup solver*, which runs on instances where computing features for the prediction models takes too long. Other instance-specific solver selection approaches include ISAC, which is based on clustering in the instance-feature space [25], and SNNAP, which extends ISAC with runtime prediction models [13]. Kerschke et al. provide a recent survey on algorithm-selection approaches, also beyond SAT [27]. For example, algorithm selection is an issues for Constraint Satisfaction Problems (CSPs) [2, 37] and Satisfiability Modulo Theories (SMT) [39, 43] as well.

Many portfolio approaches have several stages and configuration options, making configuring and selecting the best approach difficult. To cope with this, Autofolio by Lindauer et al. provides an automatic configurator [30]. One can also analyze the impact of configuration options with tools such as CAVE [8].

A systematic analysis of portfolios with varying size k, which has some similarities to our study, is conducted by Amadini et al. [1, 2]. They evaluate k-portfolios of CSP solvers in terms of their average runtime and solved instances. They compare several classification methods within portfolios, besides adapting complex SAT portfolio approaches like SATzilla. Amadini et al. focus on different instances and solvers than our study. This different focus limits the comparability of actual results, but their evaluation procedure shares similarities with ours. While they only use heuristic search algorithms, our study features a broad comparison of different search strategies for portfolios, including an exact solution, two approximate solutions, and random sampling.

Nof and Strichman formalize the *K-Portfolio Problem* in the form of two maximization problems with different objective functions and prove their submodularity and NP-completeness [36]. They solve the *K-Portfolio Problem* optimally with an SMT solver and use *greedy search* to generate approximate solutions. They evaluate their approach on anytime algorithms for an allocation problem, focusing on solution quality after one second or less. While we build on the theoretical results of Nof and Strichman, our evaluation is broader. We include more solution approaches and analyze portfolios with prediction models.

Our study analyzes two very recent datasets, created from the SAT Competitions 2020 and 2021 [6, 7]. They contain runtimes of state-of-the-art SAT solvers for a very diverse set of hard SAT instances, which are measured up to a timeout of 5000 seconds. Froleyks et al. analyze the results of the SAT Competition 2020 in detail [16]. While they also evaluate portfolios shortly, our study provides an in-depth analysis of portfolios.

3 Preliminaries

Let a set of solvers $S = \{s_1, \ldots, s_n\}$, a set of SAT instances $I = \{i_1, \ldots, i_l\}$, and solver runtimes $r : I \times S \to [0, T]$ with a fixed timeout T be given. A scoring function $c_T : S \to \mathbb{R}$ estimates solver performance. To score a solver, we use the popular penalized-average-runtime measure with a penalization factor of two (PAR-2 score). This score offers a trade-off between solver runtimes and the number of solved instances. It is defined as follows:

$$r_T(i,s) := \begin{cases} 2 \cdot T & \text{if } r(i,s) = T \\ r(i,s) & \text{otherwise} \end{cases}$$ Penalized Runtimes

$$c_T(s) := \frac{1}{|I|} \sum_{i \in I} r_T(i,s)$$ PAR-2 Score

A portfolio $P \subseteq S$ is a non-empty set of solvers. To score a solver portfolio P, we assume an oracle that always selects the fastest solver for each instance. This construction is commonly referred to as the virtual best solver (VBS) for P. Accordingly, we extend the scoring function as follows:

$$c_T(P) := \frac{1}{|I|} \cdot \sum_{i \in I} \min\{r_T(i,s) \mid s \in P\}$$

In the following, we refer to $c_T(P)$ as the *cost* of P.

In reality, one does not have access to an oracle. However, one may train a prediction model $m : I \to P$ for a solver portfolio P. This prediction model recommends a solver for each instance, using features of the instance. The cost of such a model-based portfolio is given by the following function:

$$c_T(P,m) := \frac{1}{|I|} \cdot \sum_{i \in I} r_T(i, m(i))$$

The VBS-based portfolio cost $c_T(P)$ is a lower bound for the actual cost of a model-based portfolio P.

4 Optimal k-Portfolios

The *K-Portfolio Problem* is to find a portfolio P of size $|P| = k$ with minimum cost:

$$\operatorname*{arg\,min}_{P \subseteq S, |P|=k} c_T(P)$$

Note that the portfolio-cost function decreases monotonically under the addition of solvers: $\forall s \in S, c_T(P \cup \{s\}) \leq c_T(P)$. In particular, adding further solvers only increases the number of choices for the oracle, which always chooses optimally. In contrast, monotonicity might not hold if one analyzes $c_T(P,m)$ instead of $c_T(P)$. In particular, a prediction model might choose suboptimal solvers from the portfolio. We will analyze this behavior later.

In the following, we outline three solution approaches for the *K-Portfolio Problem*, an exact one based on *integer programming*, and two approximate methods, namely *beam search* and *k-best*. To the best of our knowledge, the *integer programming* formulation is novel, while Nof and Strichman [36] already used a particular variant of *beam search* to build algorithm portfolios, and *k-best* is a baseline from Nof and Strichman as well.

4.1 Integer Programming

To solve the *K-Portfolio Problem* exactly, we propose an integer linear program. The *K-Portfolio Problem* is not linear due to the use of the min() function. However, reformulating the problem with additional variables can make it linear. In consequence, an off-the-shelf integer-programming solver can find an exact solution for the problem.

Algorithm 1 *Beam search.*

Input: Set of solvers S, Portfolio size k, Portfolio cost function c_T, Beam width w
Output: Portfolio P with $|P| = k$

$T_0 \leftarrow \{\emptyset\}$
for $i \leftarrow 1$ **to** k **do**
  `// Candidate` i`-portfolios:`
  $U \leftarrow \emptyset$
  foreach $P \in T_{i-1}$ **do**
    foreach $s \in S \setminus P$ **do**
      $U \leftarrow U \cup \{P \cup \{s\}\}$
  `// Select` w `best` i`-portfolios:`
  $T_i \leftarrow \emptyset$
  for $j \leftarrow 1$ **to** w **do**
    $T_i \leftarrow T_i \cup \{\arg\min_{P \in U \setminus T_i} c_T(P)\}$
return $\arg\min_{P \in T_k} c_T(P)$

We introduce two sets of binary decision variables. The variables $y_s \in \{0, 1\}$ indicate whether a solver $s \in S$ is in the portfolio, and the variables $x_{i,s} \in \{0, 1\}$ indicate whether solver $s \in S$ is selected for instance $i \in I$. Equation 1 specifies the cardinality constraint on the number of solvers. Equation 2 stipulates that exactly one solver is chosen for each instance. Equation 3 ensures that a solver can only be chosen for an instance if it is part of the portfolio. Ultimately, Equation 4 specifies the optimization target.

$$\sum_{s \in S} y_s \leq k \tag{1}$$

$$\forall i \in I : \sum_{s \in S} x_{i,s} = 1 \tag{2}$$

$$\forall s \in S : \sum_{i \in I} x_{i,s} \leq |I| \cdot y_s \tag{3}$$

$$\min_{x,y} \quad \frac{1}{|I|} \cdot \sum_{i \in I} \sum_{s \in S} r_T(i, s) \cdot x_{i,s} \tag{4}$$

As another exact solution for the *K-Portfolio Problem*, Nof and Strichman present an SMT encoding of the problem and solve it with *Z3* [36]. However, we found our re-implementation of their encoding to be slower than our integer-programming formulation. Also, all exact approaches should yield solutions with the same portfolio cost anyway.

4.2 Beam Search

Beam search is an approximate method that iteratively finds portfolios. Algorithm 1 specifies the approach. In each iteration, the algorithm combines w portfolios from the previous iteration with individual solvers that are not part of these portfolios. In other words, the algorithm expands existing portfolios by adding individual solvers. Thus, the algorithm considers the marginal contribution [47] of solvers to the current portfolios.

Before the next iteration, only the w portfolios with the lowest cost are retained. The beam width w is an input parameter. For $w = 1$, only one portfolio remains at the end of each iteration. We refer to this special case, which also Nof and Strichman used for algorithm portfolios [36], as *greedy search*.

For reasonably small w, *beam search* has a clear runtime advantage compared to an exhaustive search over all k-portfolios. In particular, *beam search* only evaluates $O(|S| \cdot w)$ out of $\binom{|S|}{k}$ possible portfolios per iteration.

Though the algorithm does not necessarily find the optimal solution to the *K-Portfolio Problem*, there is a bound on the cost of a portfolio found by *greedy search*. To this end, one can use a result from [34], which applies to greedy algorithms on non-negative monotone submodular set functions. Nof and Strichman show that their *K-Algorithms Max-Sum Problem* for portfolios is submodular, and thus a bound on greedy algorithms holds [36]. We can transform the *K-Portfolio Problem* (minimization) into the *K-Algorithms Max-Sum Problem* (maximization) by replacing the cost function with a utility function as follows:

$$u_T(P) := c_W - c_T(P)$$

In this transformation, c_W is an upper bound on portfolio cost, the single worst solver:

$$c_W := \max\{c_T(s) \mid s \in S\}$$

As $u_T(P)$ is non-negative, monotone, and submodular, we get a lower bound on the utility of a portfolio found by *greedy search* P_k^{greedy} [34, 28]:

$$u_T(P_k^{greedy}) \geq (1 - \frac{1}{e}) \cdot \max_{|P| \leq k} u_T(P)$$

One can transform this into an upper bound on the cost of a portfolio found by *greedy search*:

$$c_T(P_k^{greedy}) \leq (1 - \frac{1}{e}) \cdot \min_{|P| \leq k} c_T(P) + \frac{1}{e} \cdot c_W \tag{5}$$

4.3 K-Best

K-best is a baseline from the study of Nof and Strichman [36]. It sorts all solvers by their individual performance and then picks the top k from this list. Unlike *beam search*, it does not consider how solvers within a portfolio interact, i.e., if they complement each other. Thus, *k-best* only needs to evaluate $|S|$ intermediate portfolios to determine a solution, compared to the $O(|S| \cdot w \cdot k)$ portfolios of *beam search*.

5 Experimental Design

In our experiments, we evaluate the solution approaches just presented. Besides analyzing the VBS, we combine the found k-portfolios with prediction models for instance-specific solver selection. In both cases, we are interested in the influence of the portfolio size k on portfolio performance, measured as portfolio cost $c_T(P)$ or $c_T(P, m)$, respectively.

For evaluation purposes, we conduct five-fold cross-validation over SAT instances. Thus, we only use SAT instances from the training folds to search for k-portfolios and subsequently train prediction models. In particular, neither portfolio search nor prediction models have access to solver runtimes on the test instances. The prediction models need to recommend a solver from the portfolio only based on instance features. We compute all evaluation metrics on training instances and on test instances. We average evaluation metrics over the cross-validation folds.

5.1 Solution Approaches

We employ the three solution approaches from Section 4 to determine k-portfolios for each $k \in \{1, \ldots, |S|\}$. We also generate an additional baseline via random sampling of k-portfolios.

- *Optimal solution*: We solve the *K-Portfolio Problem* as an integer optimization problem to exactly determine the best portfolio.
- *Beam search*: We search for good portfolios heuristically in a bottom-up manner. We compare $w \in \{1, 2, 3, \ldots, 10, 20, 30, \ldots, 100\}$ as beam widths.
- *K-best*: We build portfolios from the k best individual solvers as a simple baseline.
- *Random sampling*: To get an idea of how the performance of arbitrary portfolios is distributed, we randomly sample 1000 portfolios for each k.

The optimization goal for all approaches is the PAR-2 score $c_T(P)$. In preliminary experiments, we also analyzed two slightly different objectives: the number of unsolved instances and the PAR-2 score normalized for each instance. However, general trends in the results were similar to those with the PAR-2 score, so we stuck to the latter.

5.2 Prediction Approaches

We also analyze the performance of the previously determined k-portfolios with prediction models to select a solver rather than choosing the VBS. For each instance, the prediction target is the best solver out of the given k-portfolio. Thus, we have a multi-class prediction problem with k classes. Inputs for the prediction are numeric features characterizing a SAT instance; we will describe the features later. As prediction models, we leverage two powerful ensemble methods. First, we use random forests [9], which are also part of the well-known portfolio approach SATzilla 2012 [48]. Second, we use XGBoost [12], a popular implementation of gradient boosting. In both cases, we train models with 100 trees.

Our preliminary experiments also included a k-nearest-neighbors classifier, which is popular for portfolio approaches as well [13, 31, 35, 40], and a simple neural network with one hidden layer. However, both performed worse than tree-based ensembles. The former might suffer from the high number of features, while the latter might need parameter tuning, which would have been too expensive in our study. Further, we tried other prediction approaches that are part of SATzilla 2008 or SATzilla 2012: First, one can train a classifier for each pair of solvers instead of one classifier that makes a multi-class prediction. Second, one can weight instances based on the solvers' runtime difference instead of using an unweighted training set. Third, one can train a regression model for each solver from the portfolio, predict the runtime, and choose the solver with the lowest predicted runtime. However, none of these changes has helped improve prediction performance in our preliminary experiments.

We evaluate prediction performance in two ways:

- *Objective value*: We evaluate the prediction models with the cost function $c_T(P, m)$ for the recommended solvers (cf. Section 3).
- *MCC*: We evaluate the predictions with Matthews correlation coefficient (MCC) [32, 18]. This metric does not take into account how fast the recommended solvers are, but only if the fastest solver is recommended or not. We use MCC instead of simpler metrics like accuracy, as the class labels might be imbalanced, i.e., one solver might be the fastest for most of the instances, and always predicting this solver would already yield high accuracy. MCC has a range of $[-1, 1]$. The value zero occurs with both random guessing and constantly guessing the same solver. The value one corresponds to a perfect prediction.

5.3 Dataset

In our experiments, we use two datasets. The first one, `SC2020`, contains the runtime data of 48 solvers on 400 instances from the Main Track of the SAT Competition 2020 [6, 16]. The second dataset, `SC2021`, contains the runtime data of 46 solvers on 400 instances from the Main Track of the SAT Competition 2021. In both datasets, we filtered out those instances where no solver finished within the timeout of 5000 s, such that runtimes for 316 instances remained in `SC2020` and for 325 instances in `SC2021`.

For predictions, we use 138 features to characterize instances, which are from the feature extractor of SATzilla 2012 [45]. Features come from twelve categories that describe different aspects of SAT instances. For example, the number of variables and clauses roughly measure the problem size. Several features summarize graph representations of instances, e.g., the maximum node degree in the clause graph. As the last example, several features base upon probing, i.e., running solvers for a short time and quantifying their progress over that period.

Some feature values are missing since the feature extractor exceeded time- or memory limits. For predictions, we replace missing values with a constant value out of the range of the features.

In both datasets, the number of features is relatively large compared to the number of instances. However, the prediction models in our experiments are tree-based and, therefore, implicitly select features during their training. Further, these models are not affected by monotonic transformations of features, making the experimental results more robust.

5.4 Implementation

We implement our experimental design in Python and make our code available online[1]. The code also allows downloading and preparing the datasets. Additionally, we publish the complete experimental data, including datasets and results[2]. To obtain the instance- and solver data, we use the package *gbd-tools* [22]. For predictions, we use the package *scikit-learn* [38]. To solve the *K-Portfolio Problem* exactly, we use the package *mip* [41] with its integrated mixed integer linear-programming solver *COIN-OR branch-and-cut* (Cbc).

6 Evaluation

We evaluate the solution approaches for the *K-Portfolio Problem* first and the use of prediction models for solver recommendation second.

6.1 Optimization Results

The datasets seem promising for portfolios, as in both datasets, there is no single solver that is fastest for all or even for most instances. For the 316 instances in `SC2020`, the three overall fastest solvers win on only 46, 38, and 26 instances, respectively. For the 325 instances in `SC2021`, the three overall fastest solvers win on only 25, 22, and 20 instances, respectively. These observations indicate that combining solvers in portfolios can improve overall runtime. See [16] for details how single solvers performed in `SC2020`. In this section, we focus on portfolios as a whole, particularly the portfolio performance $c_T(P)$.

[1] `https://github.com/Jakob-Bach/Small-Portfolios`
[2] `https://doi.org/10.5445/IR/1000146629`

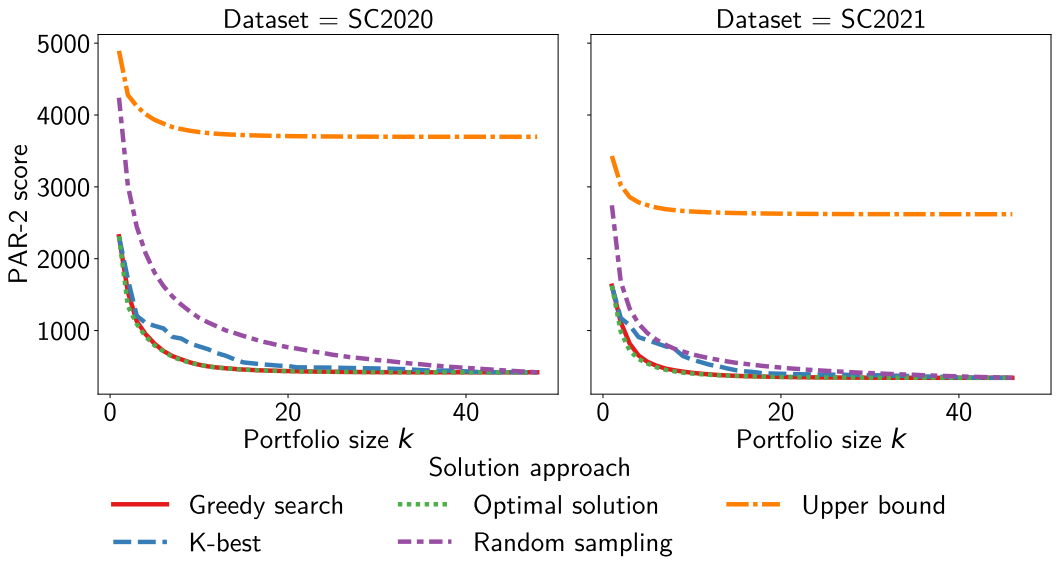

Figure 1 Training-set VBS performance of k-portfolios determined by different solution approaches for the `SC2020` dataset (left) and the `SC2021` dataset (right).

Figure 1 displays the cost in terms of the PAR-2 score of the best k-portfolios for the different solution approaches. The *optimal solution* is the exact optimum. *Greedy search* denotes *beam search* with the smallest beam width $w = 1$. We will discuss other beam widths as well. *K-best* stands for portfolios comprised of the top k single best solvers. For *random sampling*, we average over repeated samples of portfolios. The *upper bound* limits the cost of *greedy search* according to Equation 5. If we report numbers for the solution approaches in the following, we refer to the training set, where all solution approaches conduct their search.

6.1.1 Optimal Solution

The PAR-2 score improves rapidly for the first few k, not only for the optimal solution but for all approaches and both datasets. However, marginal gains become smaller with increasing k. For the dataset `SC2020`, the optimal 1-portfolio, i.e., the single best solver, has a penalized runtime 5.51 times as high as the 48-portfolio, i.e., the set of all solvers. This ratio reduces to 1.89 for the best 5-portfolio and 1.25 for the best 10-portfolio. For `SC2021`, the respective ratios are 4.74 for $k = 1$, 1.58 for $k = 5$ and 1.17 for $k = 10$.

The solver *Cbc* was able to find these optimal solutions within a reasonable time. The mean optimization time was 15 s, and the maximum optimization time was about 5 min. Most of the variance in optimization time occurred for small portfolios. For $k \geq 10$, all optimization runs took less than 14 s.

6.1.2 Beam Search

Figure 1 also shows the cost of the best k-portfolios found by *greedy search* to be very close to that of the *optimal solution*. These results are remarkable, considering that the runtime of *greedy search* is linear in k as well as the total number of solvers n, whereas finding the *optimal solution* is NP-complete. In contrast, the theoretical, submodularity-based *upper bound* for *greedy search* is clearly higher than the actual portfolio cost and, thus, too loose to serve as a reasonable estimate.

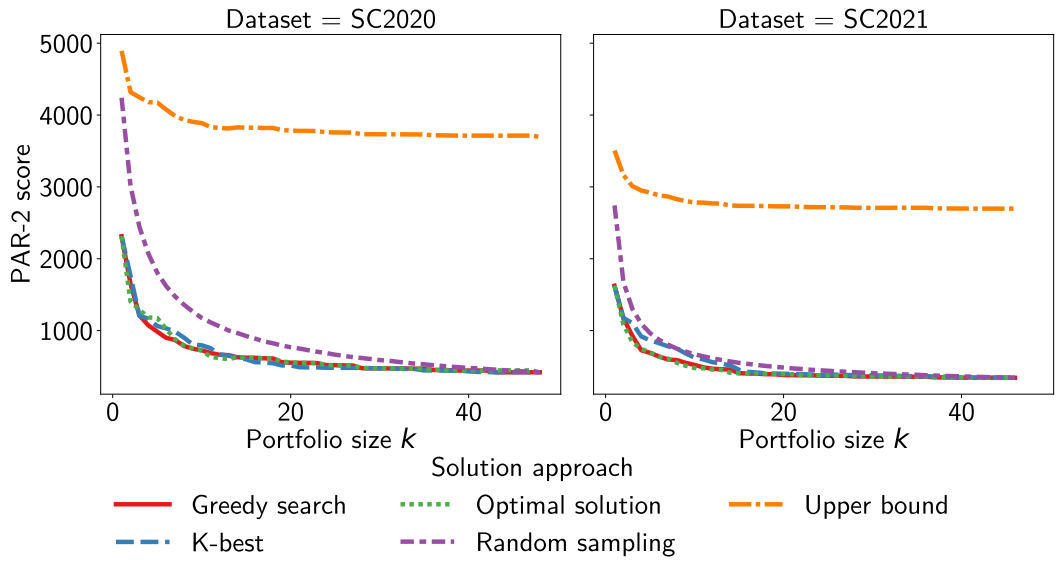

Figure 2 Test-set VBS performance of k-portfolios determined by different solution approaches for the SC2020 dataset (left) and the SC2021 dataset (right).

To bring the *beam search* solution even closer to the *optimal solution*, one can increase beam width w. For example, the best 2-portfolio found by *greedy search* has a 14.7% higher cost than the *optimal solution* for SC2020. For all other k, the *greedy search* portfolio has less than 4% higher cost than the *optimal solution*. In comparison, for $w = 10$ and for all k, cost is never more than 2% higher than for the *optimal solution*.

Regarding the set of the w best portfolios maintained in each iteration of *beam search*, we observe a convergence of portfolio performance with increasing k. In particular, there is a substantial variance in the PAR-2 score of portfolios in the beam for small k. This variance becomes smaller in later iterations, i.e., the top w portfolios become more similar in performance. A similar phenomenon also occurs for *random sampling* portfolios: With increasing k, the standard deviation of the PAR-2 score in a sample of portfolios decreases. The expected value of portfolio performance improves as well. Thus, carefully selecting the solvers for a portfolio matters most for small k.

6.1.3 K-Best

The baseline *k-best* is worse than *greedy search* on the training set, but better than *random sampling*. While *greedy search* always is quite close to the *optimal solution*, the performance gap between *k-best* and the *optimal solution* widens after the first few k and only becomes smaller for large k later. The performance of *k-best* relative to the other approaches also differs between SC2020 and SC2021. For SC2020, *k-best* is closer to the optimal solution, while for SC2021, *k-best* is closer to the expected value of *random sampling*. These results indicate that building a portfolio of complementary solvers, rather than picking the best individual solvers, may be more important for SC2021.

6.1.4 Test-Set Performance

Figure 2 shows the test-set portfolio cost for the different solution approaches. Here, we take the portfolios found on the training set but evaluate them with the test-set instances. The overall trends are the same as on the training set, cf. Figure 1. A notable exception

is that *greedy search*, *k-best*, and the *optimal solution* show similar performance on the test set of SC2020. In particular, there is no clear winner, and the *optimal solution* can even perform worse than portfolios found by the approximate approaches. This is because the best portfolio on the training set is not necessarily the best portfolio on the test set. For the test set of SC2021, *k-best* is markedly worse than *greedy search* and the *optimal solution*. The larger performance gap already on the training set might have caused this effect.

6.1.5 Portfolio Composition

While *beam search* adds solvers iteratively, the *optimal solution* might differ in more than one solver from $k-1$ to k, i.e., existing solvers from the portfolio can be replaced. Indeed, we observe this phenomenon in our results. For example, the optimal 2-portfolios do not contain the optimal 1-portfolios for both datasets. However, this non-monotonous behavior is not strong. On average, only one or two new solvers become part of the optimal portfolio when increasing k by one. I.e., zero or one solver are replaced on average when increasing k by one. In addition, the replaced solver might only have been slightly worse than its substitute. This might explain the good performance of *beam search*, which pursues a monotonous approach when building portfolios.

6.1.6 Impact of Single Solvers on Portfolios

We have carried out a correlation analysis on the 1000 portfolios that result from *random sampling* for a particular value of k. First, we encode the absence or presence of each solver in a portfolio with 0 or 1, respectively. Next, we compute the Spearman rank correlation between this occurrence vector and the PAR-2 score. Our analysis highlights the interaction between solvers. For $k = 5$, all correlations are in $[-0.40, 0.17]$ for SC2020, and in $[-0.24, 0.23]$ for SC2021. The mean correlation is zero in both cases. Similar correlation behavior occurs for other k.

These results indicate that only the presence of some solvers has a moderately negative correlation to the PAR-2 score of the whole portfolio, i.e., only some solvers can improve the portfolio performance of our minimization problem on their own. The positive correlations are even weaker, i.e., there are no solvers which influence portfolio performance in a strongly negative manner. Overall, this means that the influence of single solvers on portfolio performance is limited; one needs a combination of solvers to influence the PAR-2 score in either direction strongly.

6.2 Prediction Results

In this section, we evaluate the prediction performance for instance-specific solver recommendations in the portfolios found earlier. We also evaluate the portfolio performance $c_T(P, m)$, i.e., the PAR-2 score of the recommended solvers.

6.2.1 Matthews Correlation Coefficient

Figure 3 graphs the test-set classification performance of random forests and XGBoost, using the portfolios from *random sampling*. We use *random sampling* results to show the variation of prediction performance over many portfolios. However, classification results are similar for *beam search* with $w = 100$ and for the *optimal solution*. We do not display training-set performance since it consistently is close to the maximum MCC of 1.0.

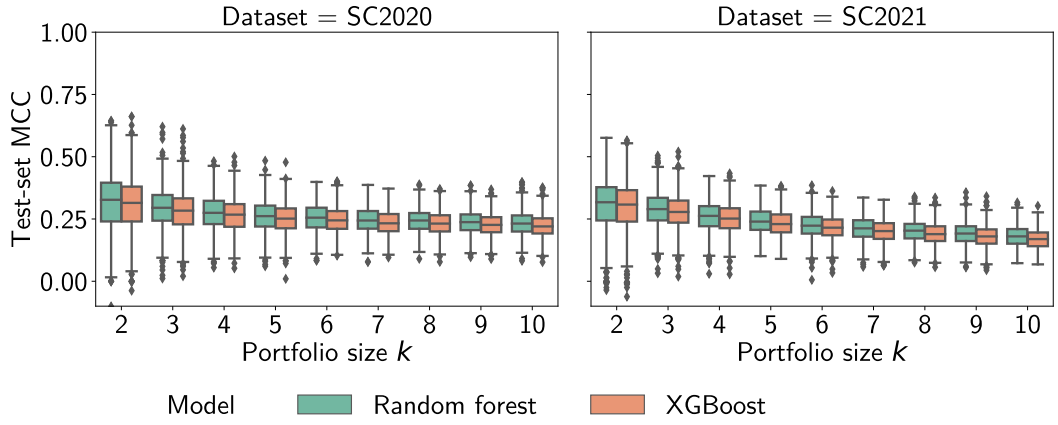

Figure 3 Test-set prediction performance (MCC) for randomly sampled portfolios, using random forests and XGBoost as models, for the `SC2020` dataset (left) and the `SC2021` dataset (right).

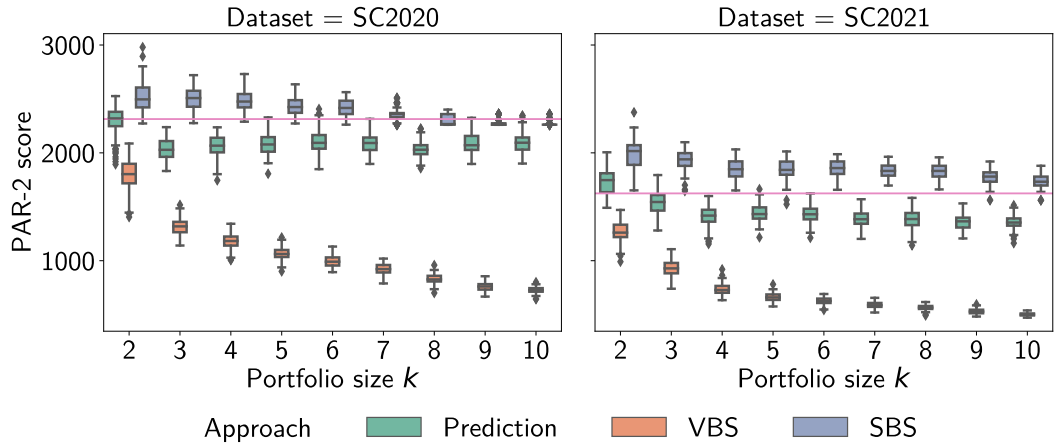

Figure 4 Test-set performance of model-based portfolios, their VBS, and their SBS for *beam search* with $w = 100$ for the `SC2020` dataset (left) and the `SC2021` dataset (right). The performance of the global SBS is shown as a horizontal line.

As Figure 3 shows, test-set prediction performance is rather low for all k and for both types of prediction models. Prediction performance is clearly higher than with random guessing, which would yield an MCC of 0.0, but clearly lower than the optimal MCC of 1.0. Given the nearly perfect training-set performance but a low test set performance, the models seem to overfit the training set. Overfitting could cause a bad PAR-2 value for model-based portfolios, which we will analyze next. For small k, the prediction performance varies stronger between portfolios than for larger k and is slightly higher on average. As random forests and XGBoost perform similarly, we will only use random-forest results in the following analyses.

6.2.2 Portfolio Performance

Figure 4 shows test-set PAR-2 scores for portfolios from *beam search* with $w = 100$. The plot compares the PAR-2 score of solvers recommended by the prediction model to two competitors without a prediction model, computed on the same portfolios. The virtual

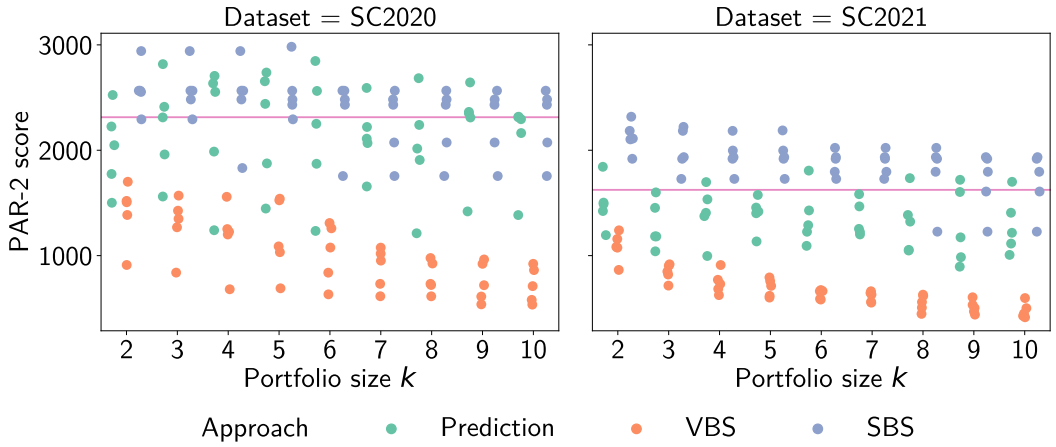

Figure 5 Test-set performance of model-based portfolios, their VBS, and their SBS for the *optimal solution* of the *K-Portfolio Problem* for the `SC2020` dataset (left) and the `SC2021` dataset (right). The performance of the global SBS is shown as a horizontal line.

best solver (VBS) provides a lower bound on cost, as its portfolio performance can only be achieved with perfect prediction. Each portfolio's single best solver (SBS) serves as a baseline, corresponding to always recommending one particular solver. Note that the portfolio performance can even be worse than the SBS, e.g., when always predicting the slowest solver for each instance. However, we do not show this upper bound here.

As discussed earlier, the VBS score decreases with k. In theory, this also allows model-based portfolios to improve their performance. However, in our case, the PAR-2 score of model-based portfolios remains relatively stable with increasing values of k, with the biggest improvement from $k = 2$ to $k = 3$. This observation implies that the prediction model does not improve its selection of solvers even if the portfolio grows. Given the low prediction performance in terms of MCC, as seen in Figure 3, this has been expected. Consequently, the gap between VBS and the predicted solvers grows with k. However, model-based portfolios tend to be better than the single best solver from these portfolios, i.e., the prediction models can discriminate between solvers to some extent. In addition, for $k > 2$, the model-based portfolios are better than the global single best solver, at least on average.

Figure 5 repeats the same comparison as before for the *optimal solution* portfolios. The overall trends remain the same. In contrast to Figure 4, here we have only five portfolios for each k, one for each fold of cross-validation. We still see a considerable variance for each k, i.e., the test set performance also depends on the current train-test split of instances.

6.2.3 Feature Importance

Averaging importance over all trained random-forest models, the most important feature has an importance score of 1.66%, the least important one an importance score of 0.01%. To reach a cumulated average importance of 50%, one needs 38 out of 138 features. On average, random forests use 135 features, i.e., nearly all of them. We conclude that no single feature or small set of features drives prediction performance. Features might be redundant to each other, as the feature extractor often applies several statistical aggregates to summarize the same characteristics of the instances. Also, the low prediction performance indicates that the features are not sufficiently discriminating to recommend solvers on our datasets reliably.

7 Conclusions and Future Work

7.1 Conclusions

Solution methods for the propositional satisfiability problem are an active area of research, with continuous advances in methods, heuristics, and their implementations in SAT solvers. New SAT solvers that improve performance only on a few benchmark instances can nevertheless be assets to solver portfolios. In principle, runtime prediction models can help to leverage such complementary solvers.

This article has been a comparative study of portfolios that goes beyond previous work in several respects. Our study includes the runtime data of the latest SAT solvers from the SAT Competitions 2020 and 2021. It has addressed the essential question of how large portfolios drawn from that set of solvers should be. There has been a substantial improvement in the objective value for small portfolios but a negligible impact once a particular size is reached. We also found that *beam search*, an approximate solution approach, yields close-to-optimal solutions. In order to facilitate portfolio analysis for evaluation purposes and ad-hoc portfolio generation, we integrated our exact portfolio-search approach into the Python package *gbd-tools* [22].

As the next step, we have combined portfolios with prediction models to recommend solvers specific to problem instances. However, these models have not performed satisfactorily on the given datasets: The objective value of model-based portfolios did not continuously improve with portfolio size.

7.2 Future Work

We see room for improved prediction performance by adapting the set of instance features in particular, e.g., by using features describing the community structure of graph representations of SAT instances. Such new features have correlated well with solver performance on application instances in recent studies [5, 3, 4, 29]. Efficient implementations of new feature extractors and studying feature importance on datasets that represent the state-of-the-art are subject to future work.

Traditionally, SAT competitions incentivize the development of so-called *stable* solvers by evaluating the best average performance on benchmarks from diverse applications. This general trend has been mitigated in the competitions considered in our study. These competitions also highlight SAT solvers that perform particularly well on specific instance families [16] or on a (not further specified) large subset of the benchmark instances (cf. Example 1). Such additional evaluations provide reference points for more informed portfolio composition and can even positively impact research on stable solvers [14]. For future analyses, we plan to increase the number of specialized solvers and configurations targeting *well-defined* subsets of benchmark instances.

References

1 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An empirical evaluation of portfolios approaches for solving CSPs. In *Proc. CPAIOR*, pages 316–324, 2013. doi:10.1007/978-3-642-38171-3_21.

2 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An extensive evaluation of portfolio approaches for constraint satisfaction problems. *Int. J. Interact. Multim. Artif. Intell.*, 3(7):81–86, 2016. doi:10.9781/ijimai.2016.3712.

3 Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, and Jordi Levy. Structure features for SAT instances classification. *J. Appl. Log.*, 23:27–39, 2017. doi:10.1016/j.jal.2016.11.004.

4 Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Community structure in industrial SAT instances. *J. Artif. Intell. Res.*, 66:443–472, 2019. doi:10.1613/jair.1.11741.

5 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On the structure of industrial SAT instances. In *Proc. CP*, pages 127–141, 2009. doi:10.1007/978-3-642-04244-7_13.

6 Tomáš Balyo, Nils Froleyks, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2020. URL: http://hdl.handle.net/10138/318754.

7 Tomáš Balyo, Nils Froleyks, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2021: Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2021. URL: http://hdl.handle.net/10138/333647.

8 André Biedenkapp, Joshua Marben, Marius Lindauer, and Frank Hutter. CAVE: Configuration assessment, visualization and evaluation. In *Proc. LION*, pages 115–130, 2018. doi:10.1007/978-3-030-05348-2_10.

9 Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. doi:10.1023/A:1010933404324.

10 Marko Kleine Büning, Carsten Sinz, and David Faragó. QPR Verify: A static analysis tool for embedded software based on bounded model checking. In *Proc. VSTTE*, pages 21–32, 2020. doi:10.1007/978-3-030-63618-0_2.

11 Tom Carchrae and J. Christopher Beck. Applying machine learning to low-knowledge control of optimization algorithms. *Comput. Intell.*, 21(4):372–387, 2005. doi:10.1111/j.1467-8640.2005.00278.x.

12 Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. KDD*, pages 785–794, 2016. doi:10.1145/2939672.2939785.

13 Marco Collautti, Yuri Malitsky, Deepak Mehta, and Barry O'Sullivan. SNNAP: solver-based nearest neighbor for algorithm portfolios. In *Proc. ECML PKDD*, pages 435–450, 2013. doi:10.1007/978-3-642-40994-3_28.

14 Mathias Fleury and Armin Biere. Efficient all-UIP learned clause minimization. In *Proc. SAT*, pages 171–187, 2021. doi:10.1007/978-3-030-80223-3_12.

15 Alexandre Fréchette, Lars Kotthoff, Tomasz Michalak, Talal Rahwan, Holger Hoos, and Kevin Leyton-Brown. Using the shapley value to analyze algorithm portfolios. In *Proc. AAAI*, pages 3397–3403, 2016. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10440.

16 Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT Competition 2020. *Artif. Intell.*, 301, 2021. doi:10.1016/j.artint.2021.103572.

17 Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1–2):43–62, 2001. doi:10.1016/S0004-3702(00)00081-3.

18 Jan Gorodkin. Comparing two k-category assignments by a k-category correlation coefficient. *Comput. Biol. Chem.*, 28(5–6):367–374, 2004. doi:10.1016/j.compbiolchem.2004.09.006.

19 Youssef Hamadi, Said Jabbour, and Lakhdar Sais. ManySAT: a parallel SAT solver. *J. Satisf. Boolean Model. Comput.*, 6(4):245–262, 2009. doi:10.3233/SAT190070.

20 Marijn J. H. Heule. Schur number five. In *Proc. AAAI*, pages 6598–6606, 2018. URL: https://ojs.aaai.org/index.php/AAAI/article/view/12209.

21 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Proc. SAT*, pages 228–245, 2016. doi:10.1007/978-3-319-40970-2_15.

22 Markus Iser, Luca Springer, and Carsten Sinz. Collaborative management of benchmark instances and their attributes. *arXiv preprint arXiv:2009.02995*, 2020. URL: https://arxiv.org/abs/2009.02995.

23 Mikoláš Janota, Goetz Botterweck, and João Marques-Silva. On lazy and eager interactive reconfiguration. In *Proc. VaMoS*, pages 8:1–8:8, 2014. doi:10.1145/2556624.2556644.

24 Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. *AI Mag.*, 33(1):89–92, 2012. doi:10.1609/aimag.v33i1.2395.

25 Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC – instance-specific algorithm configuration. In *Proc. ECAI*, pages 751–756, 2010. doi:10.3233/978-1-60750-606-5-751.

26 Daniela Kaufmann and Armin Biere. AMulet 2.0 for verifying multiplier circuits. In *Proc. TACAS*, pages 357–364, 2021. doi:10.1007/978-3-030-72013-1_19.

27 Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evol. Comput.*, 27(1):3–45, 2019. doi:10.1162/evco_a_00242.

28 Andreas Krause and Daniel Golovin. *Submodular Function Maximization*, pages 71–104. Cambridge University Press, 2014. doi:10.1017/CBO9781139177801.004.

29 Chunxiao Li, Jonathan Chung, Soham Mukherjee, Marc Vinyals, Noah Fleming, Antonina Kolokolova, Alice Mu, and Vijay Ganesh. On the hierarchical community structure of practical SAT formulas. In *Proc. SAT*, 2021. doi:10.1007/978-3-030-80223-3_25.

30 Marius Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. AutoFolio: An automatically configured algorithm selector. *J. Artif. Intell. Res.*, 53:745–778, 2015. doi:10.1613/jair.4726.

31 Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Non-model-based algorithm portfolios for SAT. In *Proc. SAT*, pages 369–370, 2011. doi:10.1007/978-3-642-21581-0_33.

32 Brian W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochim. Biophys. Acta - Protein Struct.*, 405(2):442–451, 1975. doi:10.1016/0005-2795(75)90109-9.

33 Saeed Nejati. *CDCL(Crypto) and Machine Learning based SAT Solvers for Cryptanalysis*. PhD thesis, University of Waterloo, Ontario, Canada, 2020. URL: http://hdl.handle.net/10012/15868.

34 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

35 Mladen Nikolić, Filip Marić, and Predrag Janičić. Simple algorithm portfolio for SAT. *Artif. Intell. Rev.*, 40(4):457–465, 2013. doi:10.1007/s10462-011-9290-2.

36 Yair Nof and Ofer Strichman. Real-time solving of computationally hard problems using optimal algorithm portfolios. *Ann. Math. Artif. Intell.*, pages 1–18, 2021. doi:10.1007/s10472-020-09704-4.

37 Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proc. AICS*, pages 210–216, 2008. URL: https://homepages.laas.fr/ehebrard/papers/aics2008.pdf.

38 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12(85):2825–2830, 2011. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

39 Nikhil Pimpalkhare, Federico Mora, Elizabeth Polgreen, and Sanjit A. Seshia. MedleySolver: Online SMT algorithm selection. In *Proc. SAT*, pages 453–470, 2021. doi:10.1007/978-3-030-80223-3_31.

40 Horst Samulowitz, Chandra Reddy, Ashish Sabharwal, and Meinolf Sellmann. Snappy: A simple algorithm portfolio. In *Proc. SAT*, pages 422–428, 2013. doi:10.1007/978-3-642-39071-5_33.

41 Haroldo G. Santos and Túlio A. M. Toffolo. Python-MIP. URL: https://python-mip.com/.

42 Dominik Schreiber. Lilotane: A lifted SAT-based approach to hierarchical planning. *J. Artif. Intell. Res.*, 70:1117–1181, 2021. doi:10.1613/jair.1.12520.

43 Joseph Scott, Aina Niemetz, Mathias Preiner, Saeed Nejati, and Vijay Ganesh. MachSMT: A machine learning-based algorithm selector for SMT solvers. In *Proc. TACAS*, pages 303–325, 2021. doi:10.1007/978-3-030-72013-1_16.

44 Matthew J. Streeter, Daniel Golovin, and Stephen F. Smith. Combining multiple heuristics online. In *Proc. AAAI*, pages 1197–1203, 2007. URL: https://www.aaai.org/Papers/AAAI/2007/AAAI07-190.pdf.

45 Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Features for SAT. Technical report, University of British Columbia, 2012. URL: https://www.cs.ubc.ca/labs/beta/Projects/SATzilla/Report_SAT_features.pdf.

46 Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008. doi:10.1613/jair.2490.

47 Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proc. SAT*, pages 228–241, 2012. doi:10.1007/978-3-642-31612-8_18.

48 Lin Xu, Frank Hutter, Jonathan Shen, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. In *Proc. SAT Challenge*, pages 57–58, 2012. URL: http://hdl.handle.net/10138/34218.