A Comprehensive Study of k-Portfolios of Recent SAT Solvers

Jakob Bach[0000−0003−0301−2798], Markus Iser[0000−0003−2904−232X], and Klemens Böhm

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{jakob.bach,markus.iser,klemens.boehm}@kit.edu

Abstract. Hard combinatorial problems such as propositional satisfiability are ubiquitous. The holy grail are methods that show good performance on all problem instances. However, new approaches emerge regularly, some of which are complementary to such 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 existing 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.

Keywords: Propositional satisfiability · Solver portfolios · Runtime prediction · Machine Learning

1 Introduction

Motivation SAT solving is the archetypal NP-complete problem. Its practical applications abound, e.g., verification of hardware and software [19,9], product configuration [17], cryptanalysis [25], or planning [30]. SAT solvers are not only used to solve hard combinatorial problems in industry, but also previously open problems in mathematics [15,14].

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 which 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 which improve performance on only a narrow subset of instances. Such approaches have a merit nevertheless – one can see them as complementary to solvers with good average behavior.

One way to leverage complementarity of solvers is with portfolios, i.e., sets of solvers. One can combine a solver portfolio with a perfect oracle which then 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 then derive a ranking of solvers based on their marginal contribution to the VBS [32].

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*. This solver has not been competitive in the overall evaluation. But due to that achievement, it has been awarded the new *Special Innovation Price*.

Yet another tweak when it comes to performance gains from sets of solvers is as follows: Combining a portfolio with a solver-selection process that includes runtime predictions usually outperforms individual best solvers by much [31]. 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 [32,11]. We argue that evaluations confined to only a few, very specific sets of instances, such as the one in [18], let the solvers appear in a better light than ones covering various domains. Moreover, the construction of a prediction-based solver-selection process is complex and quite time-consuming. 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 important when it comes to the design of portfolios and their evaluation. First, one must decide how many solvers to put in a portfolio, and which ones. 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 [27]. Second, one must decide how to select a solver for each instance. In particular, this includes the choice of instance features used to learn a prediction model as well as of the model type.

Contributions This current article features a study which 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 which one can achieve with off-the-shelf models and instance features – without any tuning. For each portfolio size k, we train random forests as prediction models, using the widely known instance feature records of SATzilla 2012 [33], and then use the 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 SAT Competitions 2020 and 2021. The two sets of benchmarks in these SAT Competitions are a balanced mix of families of interest which have been drawn from the application and crafted instance categories. Our code and data are available online (cf. Section 5.4).

Results Our evaluation yields insightful results regarding the various questions. While it is expected that the marginal utility decreases with portfolio size, we did not expect the decrease to be so early. 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 well-known *beam search* has given way to good approximate solutions to the *K-Portfolio Problem*. These solutions happen to 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 portfolios with prediction models 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; our hypothesis is 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 portfolios with prediction models do not improve further. We take this as a further indication that it might not be beneficial to have many solvers in a portfolio.

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 approaches using portfolios to evaluate the state-of-the-art of SAT solving and continue with instance-specific solver selection, including approaches to automatically configure and analyze such techniques. Finally, we discuss related work on systematic investigation of k-portfolios.

Solver complementarity regarding the runtime of solver portfolios has been examined by Xu et al. [32]. They show that significant contributions to the runtime of a portfolio often come from solvers which are not competitive when evaluated as a standalone solver. This however depends on the dataset under evaluation. In fact, the best performing 2-portfolio of solvers in SAT Competition 2020 are the winners of the SAT and UNSAT Tracks.[1] This has been markedly different in SAT Competition 2021, in which an otherwise not competitive solver is part of the best performing 2-portfolio. All this underlines the importance of portfolio analysis for a complete evaluation of the state-of-the-art. This in turn can also support the development of better stable solvers.

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 [31,33]. In SATZilla, multiple prediction models guide the selection of a solver from their portfolio. Additional components of SATzilla are so-called *pre-solvers*, which are only run for a short time to solve easy instances, and a *backup solver*, which is responsible for instances where computing features for the prediction models takes too long. Other instance-specific solver selection approaches are ISAC, which is based on clustering in the instance feature space [18], and SNNAP, which combines the ISAC approach with runtime prediction models [11]. A recent survey on algorithm-selection approaches – also beyond SAT – is provided by Kerschke et al. [20].

Many portfolio approaches have several stages and configuration options, which makes configuring and selecting the best portfolio approach difficult. To cope with this, Autofolio by Lindauer et al. provides an automatic configurator [23]. One can also analyze the impact of configuration options with tools such as CAVE [7].

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, next to adapting complex SAT portfolio approaches like SATzilla. Amadini et al. focus on different instances and solvers than this current study, though the evaluation procedure has some similarities. 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 [27]. They solve the *K-Portfolio Problem* optimally with an SMT solver and use *greedy search* to generate approximate solutions.

[1] This is the expected result for a benchmark set which contains similar amounts of SAT and UNSAT instances.

They evaluate their approach on anytime algorithms for an allocation problem, focusing on solution quality after a timeout of one second or less. While we build on the theoretical results of Nof and Strichman, our evaluation is broader. In addition to including more solution approaches, we also analyze portfolios with prediction models.

A feature of our current study, which sets it apart from the existing ones, is that it is based on two most recent datasets of the SAT Competitions 2020 and 2021. They contain runtimes of actual state-of-the-art SAT solvers for a very diverse set of hard SAT instances, which are measured with a timeout of 5000 seconds.

3 Preliminaries

Let a set of solvers $S = \{s_1, \dots, s_n\}$, a set of SAT instances $I = \{i_1, \dots, i_l\}$, 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 is a trade-off between solver runtime and the number of solved instances. It is defined as follows:

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

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

A portfolio $P \subseteq S$ is a non-empty set of solvers. To score a solver portfolio P, we assume to have 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 may train a prediction model $m : I \to P$ for a solver portfolio P that recommends a solver for each instance, using features of the instance. The cost of such a prediction model is given by the following function:

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

Clearly, the portfolio cost $c_T(P)$ is a lower bound for the actual cost of a portfolio P which uses a prediction model.

4 Optimal *k*-Portfolios

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

$$\underset{P \subseteq S, |P|=k}{\arg\min} \; c_T(P)$$

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 the following, we outline three solution approaches, an exact one based on *integer programming* and two approximate methods, namely *beam search* and *k-best*.

4.1 Integer Programming

The *K-Portfolio Problem* is not linear due to the use of the min function. However, one can make it an integer linear problem by introducing additional variables. This allows to obtain an exact solution for the problem with an off-the-shelf integer-programming solver.

We introduce two sets of binary decision variables. The binary variables y_s indicate whether a solver $s \in S$ is in the portfolio, and the binary variables $x_{i,s}$ 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.

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

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

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

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

4.2 Beam Search

Beam search is an approximate method that finds portfolios in an iterative manner. Figure 1 specifies the approach. In each iteration, the algorithm combines the portfolios from the previous iteration with individual solvers which are not part of these portfolios. In other words, the algorithm expands existing portfolios by adding individual solvers. 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 as *greedy search*. For reasonably small w, *beam search* has a clear runtime advantage compared to an exhaustive search over all k-portfolios. In

Algorithm 1: *Beam search* algorithm

```
   Input: Solvers $S$, Portfolio size $k$, Portfolio cost $c_T$
   Input: Beam width $w$
   Output: Portfolio $P$ with $|P| = k$
1  $T_0 \leftarrow \{\emptyset\}$
2  for $i \leftarrow 1$ to $k$ do
       // Candidate $i$-portfolios:
3      $U \leftarrow \emptyset$
4      foreach $P \in T_{i-1}$ do
5          foreach $s \in S \setminus P$ do
6              $U \leftarrow U \cup \{P \cup \{s\}\}$
       // Select $w$ best $i$-portfolios:
7      $T_i \leftarrow \emptyset$
8      for $j \leftarrow 1$ to $w$ do
9          $T_i \leftarrow T_i \cup \{\arg\min_{P \in U \setminus T_i} c_T(P)\}$
10 return $\arg\min_{P \in T_k} c_T(P)$
```

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 [26], 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 [27]. We can transform the *K-Portfolio Problem* (minimization) into the *K-Algorithms Max-Sum Problem* (maximization) by replacing the cost with utility 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} [26,21]:

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

This can be transformed into an upper bound on the cost of a portfolio found by *greedy search*:

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

4.3 K-Best

This is a baseline used in [27] and is one of the most obvious naive approaches. 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.

5 Experimental Design

In our experiments, we evaluate the exact and approximate solution approaches just presented. Also, 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 performance.

For evaluation purposes, we conduct five-fold cross-validation over SAT instances. This means we only use SAT instances from the training folds to search for k-portfolios and to subsequently train prediction models. We compute all evaluation metrics on training instances as well as 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 use this approach to search for good portfolios heuristically with varying beam widths $w \in \{1, 2, 3, \ldots, 10, 20, 30, \ldots, 100\}$.
– *K-best*: We use this approach to have a simple baseline for *beam search*.
– *Random sampling*: To get an idea 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, namely 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 PAR-2 score, so we stick 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. As model types, we leverage two powerful ensemble methods. First, we use random forests [8], which are also part of the well-known portfolio approach

SATzilla2012 [33]. Second, we use XGBoost [10], a popular implementation of gradient boosting. In both cases, we train models with 100 trees.

In preliminary experiments, we also tried two adaptations of the prediction approach that are part of SATzilla2012: First, one can train a classifier for each pair of solvers, instead of training one classifier that makes a multi-class prediction. Second, one can weight instances based on the runtime difference between solvers, instead of using an unweighted training set. However, none of these changes has helped to improve prediction performance in our preliminary experiments.

We evaluate prediction performance in two ways:

– *Objective value*: We evaluate the prediction models with respect to the objective value, i.e., the runtime cost function $c_T(m, P)$ (cf. Section 3).
– *MCC*: We evaluate the predictions with Matthews correlation coefficient (MCC) [24,13]. This does not take into account how fast the recommended solvers actually 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 a high accuracy. MCC has a range of $[-1, 1]$. The value zero occurs with both random guessing and always guessing the same solver.

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 SAT Competition 2020 [6,12]. The second dataset, `SC2021`, contains the runtime data of 46 solvers on 400 instances from the Main Track of SAT Competition 2021. In both datasets, we filtered out those instances where no solver finished in time, 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 [31,33]. Feature values that are missing due to the feature extractor having exceeded time- or memory limits are replaced with a constant value that is 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 kinds of models are not affected by monotonic transformations of features, which makes the experimental results more robust.

5.4 Implementation

We implement our experimental design in Python and make our code available online[2]. The code also allows to download and prepare the datasets. Additionally,

[2] https://github.com/Jakob-Bach/Small-Portfolios

we publish the full experimental data, including results[3]. To create the datasets, we use the package *gbd-tools* [16]. For predictions, we use the package *scikit-learn* [28]. To solve the *K-Portfolio Problem* exactly, we use the package *mip* [29].

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 which is fastest for all or even for a majority of the 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. This indicates that combining solvers in portfolios can improve overall runtime.

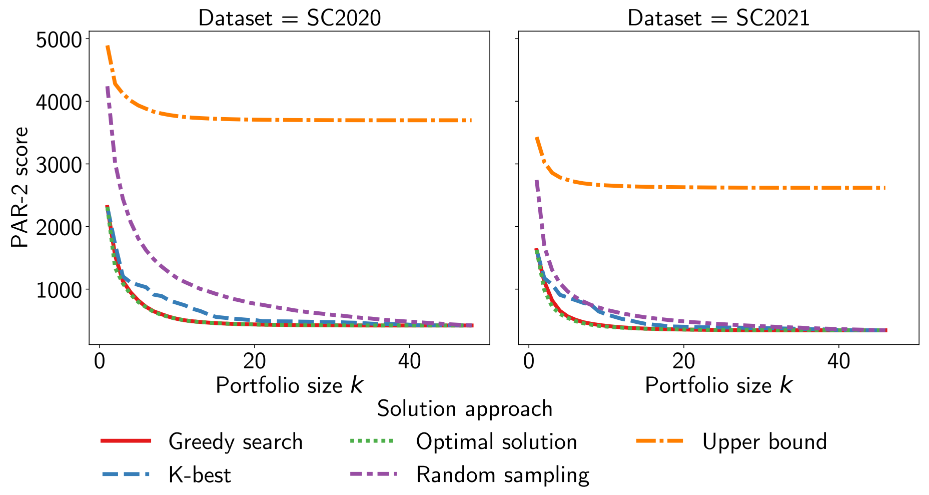

Fig. 1. Training-set VBS performance of k-portfolios determined by different approaches for the `SC2020` dataset (left) and the `SC2021` dataset (right)

General Trend 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

[3] https://bwsyncandshare.kit.edu/s/yKtJ34KTyqBtcJn; will be moved to a public repository after review

the exact optimum. *Greedy search* denotes *beam search* with the smallest beam width $w = 1$. *K-best* stands for portfolios comprised of the top k single-best solvers. For *random sampling*, we average over repeated samples. 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.

For all approaches and both datasets, the PAR-2 score improves rapidly for the first few k, but marginal gains become smaller with increasing k. For the dataset `SC2020`, the optimal 1-portfolio has a penalized runtime 5.51 times as high as the 48-portfolio. 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 $k = 10$.

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, while finding the *optimal solution* is NP-complete. In contrast, the theoretical, submodularity-based *upper bound* for *greedy search* clearly is higher and too loose to serve as a good estimate.

To bring the *beam search* solution even closer to the *optimal solution*, one can increase 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 in each iteration of *beam search*, we observe a convergence of portfolio performance with increasing k. There is a large variance in the PAR-2 score of portfolios in the beam for small k, and 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.

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*. This indicates that building a portfolio of complementary solvers, rather than picking the best individual solvers, may be more important for `SC2021`.

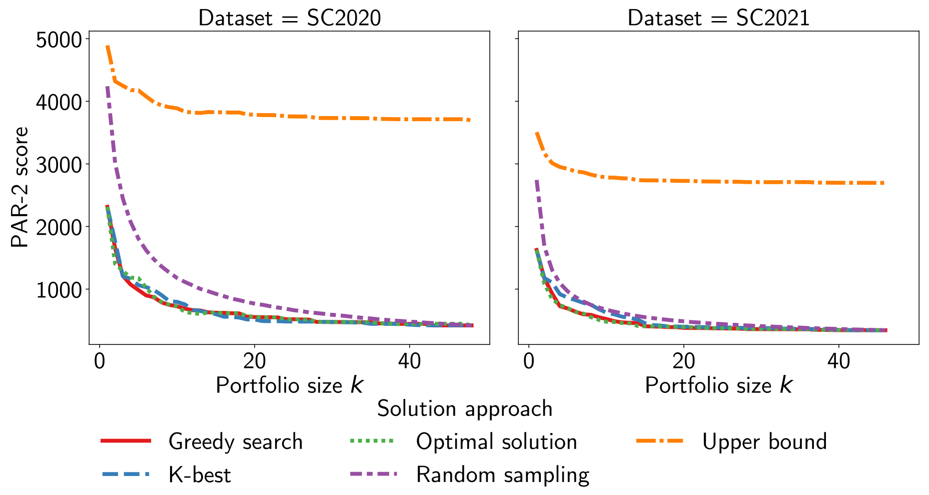

Fig. 2. Test-set VBS performance of k-portfolios determined by different approaches for the `SC2020` dataset (left) and the `SC2021` dataset (right)

Test-Set Performance Figure 2 shows 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 in Figure 1. A notable exception is that *beam 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 those 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 *beam search* and the *optimal solution*. The larger performance gap already on the training set might have caused this effect.

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, 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 be slightly worse than its substitute. This might explain the good performance of *beam search*, which pursues a monotonous approach when building portfolios.

Impact of Single Solvers on Portfolios We have carried out a correlation analysis on the 1000 portfolios which are the result of *random sampling* for a certain 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 $[-0.24, 0.23]$ 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. This effect is stronger for `SC2021`. 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 clearly influence the PAR-2 score in either direction.

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, i.e., the PAR-2 score of the predicted solvers.

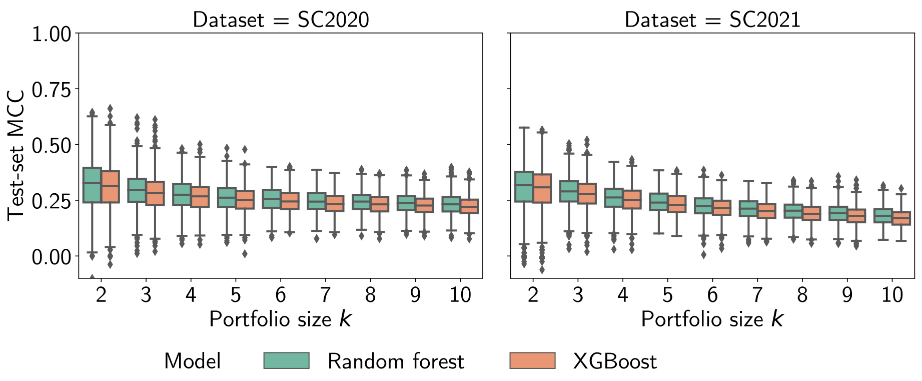

Fig. 3. Prediction performance (test-set MCC) for randomly sampled portfolios, using random forests and XGBoost as models, for the `SC2020` dataset (left) and the `SC2021` dataset (right)

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 here, 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 is close to 1.0.

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. With a nearly perfect training-set performance, but a low test set performance, the models seem to overfit to the training set. This could cause a bad PAR-2 value for model-based portfolios, and we will analyze this 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 very similar, we will only use random-forest results in the following analyses.

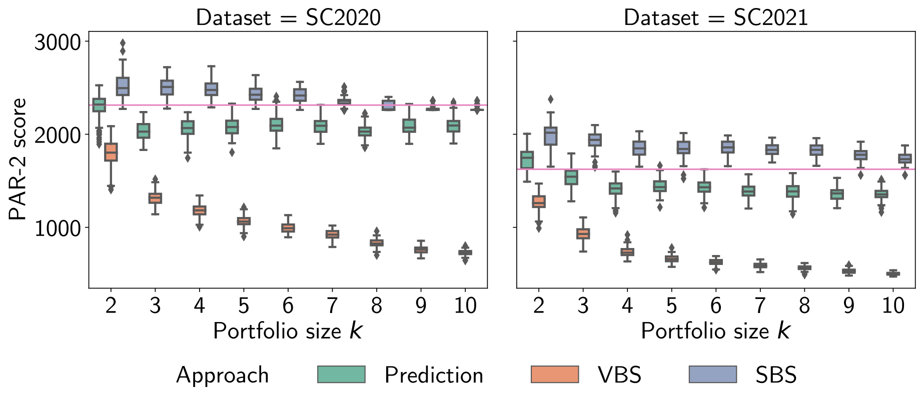

Fig. 4. Performance of prediction models, VBS and SBS for *beam search* portfolios with $w = 100$ for the `SC2020` dataset (left) and the `SC2021` dataset (right). The performance of the the global SBS is shown as a horizontal line.

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 portfolios with prediction model to two competitors without prediction model, computed on the same portfolios. The virtual best solver (VBS) provides a lower bound on cost, as its portfolio performance can only be achieved with perfect prediction. The single best solver (SBS) of each portfolio serves as a baseline, corresponding to always predicting the same 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 on the PAR-2 score here.

As discussed earlier, the VBS score decreases with k. In theory, this also allows portfolios with prediction model to improve their performance. However, in our case, the PAR-2 score of model-based portfolios remains rather stable with increasing values of k, with the biggest improvement from $k = 2$ to $k = 3$. This 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. As a consequence, 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.

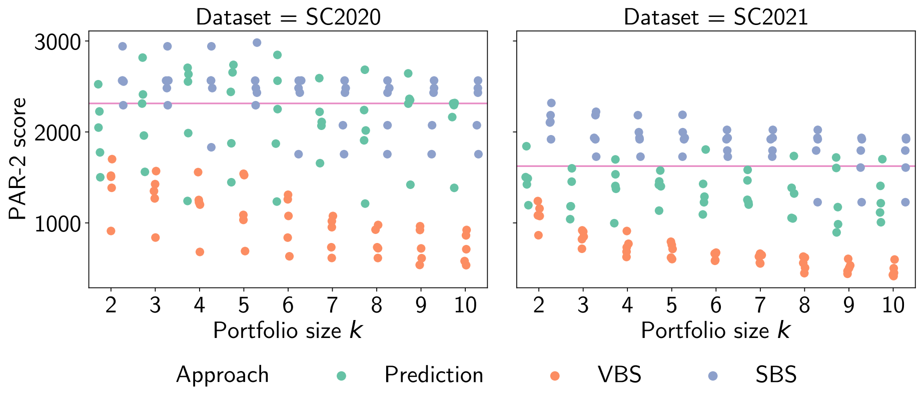

Fig. 5. Performance of prediction models, VBS and SBS for the optimal solution of the k-Portfolio Problem for the `SC2020` dataset (left) and the `SC2021` dataset (right). The performance of the the global SBS is shown as a horizontal line.

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 split of instances.

Feature Importance Averaging importance over all trained random-forest models, the most important feature has an importance of roughly 1.66%, the least important one an importance of roughly 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 even small set of features drives prediction performance. Together with the low prediction performance, this indicates that our dataset lacks discriminating features which are suitable to decide which solver should be used on an instance.

7 Conclusions and Future Work

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 be assets to portfolios nevertheless. In principle, runtime prediction models can help to find out which solvers are complementary.

This article has been a comparative study of portfolios that goes beyond previous work in several respects. One feature of our study is that it includes the runtime data of the latest SAT solvers from the SAT Competitions 2020 and 2021. It has addressed the important question of how large portfolios should actually be. There has been a strong improvement in the objective value for small portfolios, but a small impact once a certain size is reached. We also found that *beam search*, an approximate solution approach, yields close-to-optimal solutions.

In a 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 portfolios with predictions did not continuously improve with portfolio size.

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 [5,3,4,22]. Efficient implementations of feature extractors and studying feature importance on datasets which represent the state-of-the-art are subject to future work. In order to facilitate portfolio analysis for evaluation purposes and for ad-hoc portfolio generation, we also want to integrate our exact portfolio-search approach into the Python package *gbd-tools*, so one can directly use it in queries to the database GBD [16].

References

1. Amadini, R., Gabbrielli, M., Mauro, J.: An empirical evaluation of portfolios approaches for solving CSPs. In: Proc. CPAIOR. pp. 316–324 (2013). https://doi.org/10.1007/978-3-642-38171-3_21
2. Amadini, R., Gabbrielli, M., Mauro, J.: An extensive evaluation of portfolio approaches for constraint satisfaction problems. Int. J. Interact. Multim. Artif. Intell. **3**(7), 81–86 (2016). https://doi.org/10.9781/ijimai.2016.3712
3. Ansótegui, C., Bonet, M.L., Giráldez-Cru, J., Levy, J.: Structure features for SAT instances classification. J. Appl. Log. **23**, 27–39 (2017). https://doi.org/10.1016/j.jal.2016.11.004
4. Ansótegui, C., Bonet, M.L., Giráldez-Cru, J., Levy, J., Simon, L.: Community structure in industrial SAT instances. J. Artif. Intell. Res. **66**, 443–472 (2019)
5. Ansótegui, C., Bonet, M.L., Levy, J.: On the structure of industrial SAT instances. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5732, pp. 127–141. Springer (2009)

6. Balyo, T., Froleyks, N., Heule, M.J.H., Iser, M., Järvisalo, M., Suda, M. (eds.): Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions. Department of Computer Science, University of Helsinki (2020), http://hdl.handle.net/10138/318754
7. Biedenkapp, A., Marben, J., Lindauer, M., Hutter, F.: CAVE: Configuration assessment, visualization and evaluation. In: Proc. LION. pp. 115–130 (2018). https://doi.org/10.1007/978-3-030-05348-2_10
8. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324
9. Büning, M.K., Sinz, C., Faragó, D.: QPR Verify: A static analysis tool for embedded software based on bounded model checking. In: Proc. VSTTE. pp. 21–32 (2020). https://doi.org/10.1007/978-3-030-63618-0_2
10. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proc. KDD. pp. 785–794 (2016). https://doi.org/10.1145/2939672.2939785
11. Collautti, M., Malitsky, Y., Mehta, D., O'Sullivan, B.: SNNAP: solver-based nearest neighbor for algorithm portfolios. In: Proc. ECML PKDD. pp. 435–450 (2013). https://doi.org/10.1007/978-3-642-40994-3_28
12. Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: SAT Competition 2020. Artif. Intell. **301** (2021). https://doi.org/10.1016/j.artint.2021.103572
13. Gorodkin, J.: Comparing two k-category assignments by a k-category correlation coefficient. Comput. Biol. Chem. **28**(5-6), 367–374 (2004). https://doi.org/10.1016/j.compbiolchem.2004.09.006
14. Heule, M.J.H.: Schur number five. In: Proc. AAAI. pp. 6598–6606 (2018), https://ojs.aaai.org/index.php/AAAI/article/view/12209
15. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In: Proc. SAT. pp. 228–245 (2016). https://doi.org/10.1007/978-3-319-40970-2_15
16. Iser, M., Springer, L., Sinz, C.: Collaborative management of benchmark instances and their attributes. arXiv preprint arXiv:2009.02995 (2020), https://arxiv.org/abs/2009.02995
17. Janota, M., Botterweck, G., Marques-Silva, J.: On lazy and eager interactive reconfiguration. In: Proc. VaMoS. pp. 8:1–8:8 (2014). https://doi.org/10.1145/2556624.2556644
18. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC – instance-specific algorithm configuration. In: Proc. ECAI. pp. 751–756 (2010). https://doi.org/10.3233/978-1-60750-606-5-751
19. Kaufmann, D., Biere, A.: AMulet 2.0 for verifying multiplier circuits. In: Proc. TACAS. pp. 357–364 (2021). https://doi.org/10.1007/978-3-030-72013-1_19
20. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: Survey and perspectives. Evol. Comput. **27**(1), 3–45 (2019). https://doi.org/10.1162/evco_a_00242
21. Krause, A., Golovin, D.: Submodular Function Maximization, pp. 71–104. Cambridge University Press (2014). https://doi.org/10.1017/CBO9781139177801.004
22. Li, C., Chung, J., Mukherjee, S., Vinyals, M., Fleming, N., Kolokolova, A., Mu, A., Ganesh, V.: On the hierarchical community structure of practical SAT formulas. In: Proc. SAT (2021). https://doi.org/10.1007/978-3-030-80223-3_25
23. Lindauer, M., Hoos, H.H., Hutter, F., Schaub, T.: AutoFolio: An automatically configured algorithm selector. J. Artif. Intell. Res. **53**, 745–778 (2015). https://doi.org/10.1613/jair.4726

24. Matthews, B.W.: Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochim. Biophys. Acta - Protein Struct. **405**(2), 442–451 (1975). https://doi.org/10.1016/0005-2795(75)90109-9
25. Nejati, S.: CDCL(Crypto) and Machine Learning based SAT Solvers for Cryptanalysis. Ph.D. thesis, University of Waterloo, Ontario, Canada (2020), http://hdl.handle.net/10012/15868
26. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions - I. Math. Program. **14**(1), 265–294 (1978). https://doi.org/10.1007/BF01588971
27. Nof, Y., Strichman, O.: Real-time solving of computationally hard problems using optimal algorithm portfolios. Ann. Math. Artif. Intell. pp. 1–18 (2021). https://doi.org/10.1007/s10472-020-09704-4
28. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Édouard Duchesnay: Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011), http://jmlr.org/papers/v12/pedregosa11a.html
29. Santos, H.G., Toffolo, T.A.M.: Python-MIP, https://python-mip.com/
30. Schreiber, D.: Lilotane: A lifted SAT-based approach to hierarchical planning. J. Artif. Intell. Res. **70**, 1117–1181 (2021). https://doi.org/10.1613/jair.1.12520
31. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**, 565–606 (2008). https://doi.org/10.1613/jair.2490
32. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Evaluating component solver contributions to portfolio-based algorithm selectors. In: Proc. SAT. pp. 228–241 (2012). https://doi.org/10.1007/978-3-642-31612-8_18
33. Xu, L., Hutter, F., Shen, J., Hoos, H.H., Leyton-Brown, K.: SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. In: Proc. SAT Challenge. pp. 57–58 (2012), http://hdl.handle.net/10138/34218