Active Learning for SAT Solver Benchmarking

Tobias Fuchs[0000−0001−9727−2878] (✉),
Jakob Bach[0000−0003−0301−2798], and
Markus Iser[0000−0003−2904−232X]

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
info@tobiasfuchs.de, jakob.bach@kit.edu, markus.iser@kit.edu

Abstract. Benchmarking is one of the most important phases when developing algorithms. This also applies to solutions to the SAT (propositional satisfiability) problem. Benchmark selection chooses representative instances from a pool of instances that reliably discriminate SAT solvers based on their runtime. In this paper, we present a dynamic benchmark selection approach based on active learning. Our approach predicts the rank of a new solver among its competitors with minimum running time and maximum rank prediction accuracy. We evaluated this approach using the Anniversary Track dataset from the 2022 SAT Competition. Our selection approach can predict the rank of a new solver after about 10 % of the time it would take to run the solver on all instances of this dataset, with a prediction accuracy of about 92 %. In this paper, we also discuss the importance of instance families in the selection process. Overall, our tool provides a reliable way for SAT solver engineers to efficiently determine the performance of a new SAT solver.

Keywords: Propositional satisfiability · Benchmarking · Active learning

1 Introduction

One of the main phases of algorithm engineering is benchmarking. This is also true for the propositional satisfiability problem (SAT), the archetypal $\mathcal{NP}$-complete problem. Benchmarking is, however, quite expensive regarding the runtime of experiments. Competitive SAT solvers are the result of extensive experimentation with a variety of ideas and considerations [7,2]. While the field of SAT solver benchmarking is well established, traditional benchmark selection approaches do not optimize for benchmark runtime. Instead, the primary goal of traditional approaches is to select a representative set of instances for ranking solvers with a scoring scheme [9,14]. SAT Competitions typically use the PAR-2 score, i.e., the average runtime with a penalty of 2τ for timeouts with time-limit τ [7].

In this paper, we present a novel benchmark selection approach based on active learning. Our approach can predict the rank of a new solver with high accuracy in only a fraction of the time needed to evaluate the full benchmark. The problem we solve is specified in Definition 1.

Definition 1 (New-Solver Problem). *Given solvers $\mathcal{A}$, instances $\mathcal{I}$, runtimes $r\colon \mathcal{A} \times \mathcal{I} \rightarrow [0, \tau]$ with time-limit τ, and a new solver $\hat{a} \notin \mathcal{A}$, incrementally select benchmark instances in $\mathcal{I}$ to maximize the confidence in predicting the rank of $\hat{a}$ while minimizing the total benchmark runtime.*

Note that our scenario assumes that we know the runtimes of all solvers, except the new one, on all instances. One could also imagine a collaborative filtering scenario, where runtimes are only partially known [22,24].

Our approach satisfies several desirable criteria for benchmarking: Rather than outputting a binary classification, i.e., whether the new solver is worse than an existing solver or not, we provide a *scoring* function that shows by which margin a solver is worse and how similar it is to existing solvers. In particular, our approach enables *ranking* the new solver amidst a set of existing solvers. For this ranking, we do not even need to predict exact solver runtimes, which is trickier. Further, we optimize the *runtime* that our strategy needs to arrive at its conclusion. We use instance and runtime *features*. Moreover, we select instances *non-randomly* and *incrementally*. In particular, we consider runtime information from already done experiments when choosing the next. By doing so, we can control the properties of the benchmarking approach, such as its required runtime. Our approach is *scalable* in that it ranks a new solver $\hat{a}$ among any number of known solvers $\mathcal{A}$. In particular, we only subsample the benchmark once instead of comparing pairwise against each other solver [20]. Since a new solver idea is rarely best on the first try, it is desired to rank it fast. In this way, one can discard the new solver idea if it performs poorly across the board or may tweak it further if it shows promising results at least in some cases.

We evaluate our approach with the SAT Competition 2022 Anniversary Track benchmark [2], consisting of 5355 instances and complete runtimes of 28 solvers. We perform cross-validation by treating each solver as new once and predicting these solvers' PAR-2 rank. On average, our predictions reach about 92 % accuracy with only about 10 % of the runtime that would be needed to evaluate these solvers on the complete set of instances.

All our source code[1] and and data[2] are available on GitHub.

2 Related Work

Benchmarking is not only of high interest in many fields but also an active research area on its own. Recent studies show that benchmark selection is challenging for multiple reasons. Biased benchmarks can easily lead to fallacious interpretations [6]. Also, benchmark selection has many movable parts, e.g., performance measures, aggregation, and handling of missing values. Questionable research practices might modify these elements a-posteriori to fit expectations, biasing results [26]. In the following, we discuss related work from the areas of static benchmark selection, algorithm configuration, incremental benchmark selection, and active learning. Table 1 compares the most relevant approaches.

[1] temporary, anonymized version for review: xxx
[2] temporary, anonymized version for review: xxx

Table 1: Comparison of features of our benchmark-selection approach, the static benchmark-selection approach by Hoos et al. [14], the algorithm configuration system SMAC [15], and the active-learning approaches by Matricon et al. [20].

Feature	Hoos [14]	SMAC [15]	Matricon [20]	Our approach
Ranking/Scoring	✓	✗	(✓)	✓
Runtime Minimization	✗	✓	✓	✓
Incremental/Non-Random	✗	✗	✓	✓
Scalability	✓	✓	✗	✓

Static Benchmark Selection. Benchmark selection is essential for competitions, e.g., the SAT Competition. In such competitions, the organizers define the rules for composing the benchmarks. These selection strategies are mostly static, i.e., they do not depend on particular solvers to distinguish. Balint et al. provide an overview of benchmark-selection criteria in different solver competitions [1]. Froleyks et al. describe benchmark selection as it is done in recent SAT competitions [7]. Manthey and Möhle find that competition benchmarks might contain redundant instances and propose a feature-based approach to remove redundancy [19]. Mısır presents a feature-based approach to reduce benchmarks by matrix factorization and clustering [23].

Hoos et al. [14] discuss which properties are most desirable when selecting SAT benchmark instances. Selection criteria are instance variety to avoid over-fitting, adapted instance hardness (not too easy but also not too hard), and avoidance of duplicate instances. To filter too similar instances, they use a distance-based approach with the SATzilla features [34,35]. The approach does, however, not optimize for benchmark *runtime* and selects instances *randomly*, apart from constraints on the instance hardness and feature distance.

Algorithm Configuration. Further related work can be found within the field of algorithm configuration [13,31], e.g., the configuration system SMAC [15]. Thereby, the goal is to tune SAT solvers for a given sub-domain of problem instances. Although this task is different from our goal, e.g., we do not need to navigate configuration space, there are similarities to our approach as well. For example, SMAC also employs an iterative, model-based selection procedure, though for configurations rather than instances. An algorithm configurator, however, cannot be used to *rank/score* a new solver since algorithm configuration solemnly seeks to find the best-performing configuration. Also, while using a model-based selection strategy to sample configurations, instance selection is made *randomly*, i.e., without building a model over instances.

Incremental Benchmark Selection. Matricon et al. present an incremental benchmark selection approach [20]. Their *per-set efficient algorithm selection problem* (PSEAS) is similar to our *New-Solver Problem* (cf. Definition 1). Given a pair of SAT solvers, they iteratively select a subset of instances until the desired confidence level is reached to decide which of the two solvers is better. The selection of instances depends on the choice of the solvers to distinguish. They calculate a scoring metric for all unselected instances, run the experiment with the highest score, and update the confidence. Their approach ticks off most of our desired features in Table 1. However, the approach only compares solvers binarily rather than providing a *scoring*. Thus, it is unclear how similar two given solvers are or on which instances they behave similarly. Moreover, a significant shortcoming is the lacking *scalability* with the number of solvers. Comparing only pairs of solvers, evaluating a new solver requires sampling a separate benchmark for each existing solver. In contrast, our approach allows comparing a new solver against a set of existing solvers by sampling only one benchmark.

Active Learning. The posed *New-Solver Problem* has stark similarities to the well-studied field of active learning (AL) within recommender systems, in particular the *new-user problem* [28]. On the one hand, we want to maximize the utility an instance provides to our model, i.e., rank prediction accuracy, and on the other hand, minimize the cost, i.e., runtime, that is associated with its acquisition. In contrast to traditional passive machine-learning methods with given instance labels, active learning allows for selecting instances for which to acquire labels. AL algorithms can be categorized into *synthesis-based* [4,8,32] and *pool-based* approaches [10,12,18]. While synthesis-based methods generate instances for labeling, pool-based methods rely on a fixed set of unlabeled instances from which to sample.

Recent synthesis-based methods within the field of SAT solving show how to generate problem instances with desired properties. This goal is, however, orthogonal to ours [4,8]. While those approaches want to generate instances on which a solver is good or bad, we want to predict whether a solver is good or bad on an existing benchmark. Volpato and Guangyan use pool-based AL to learn an instance-specific algorithm selector [33]. Rather than benchmarking a solver's overall performance, their goal is to recommend the best solver out of a set of solvers for each SAT instance.

3 Active Learning for SAT Solver Benchmarking

The outline of our benchmarking framework is depicted in Algorithm 1. Given a set of known solvers $\mathcal{A}$, instances $\mathcal{I}$ and runtimes r, we first initialize a prediction model $\mathcal{M}$ for the new solver $\hat{a}$ (Line 1). The prediction model $\mathcal{M}$ is used to repeatedly select an instance (Line 4) for benchmarking $\hat{a}$ (Line 5). The acquired result is subsequently used to update the prediction model $\mathcal{M}$ (Line 7). When the stopping criterion is met (Line 3), we quit the benchmarking loop and predict the final score of $\hat{a}$ (Line 8). Algorithm 1 returns the predicted score of $\hat{a}$ as well as the acquired instances and runtime measurements (Line 9).

Section 3.1 describes the underlying prediction model $\mathcal{M}$ and specifies how we may derive a solver ranking from it. We discuss criteria for selecting instances in Section 3.2. Section 3.3 concludes with possible stopping conditions.

Algorithm 1: Incremental Benchmarking Framework

```
Input: Solvers $\mathcal{A}$, Instances $\mathcal{I}$, Runtimes $r : \mathcal{A} \times \mathcal{I} \rightarrow [0, \tau]$, Solver $\hat{a}$
Output: Predicted Score of $\hat{a}$, Measured Runtimes $\mathcal{R}$
1 $\mathcal{M} \leftarrow$ initModel ($\mathcal{A}$, $\mathcal{I}$, $r$, $\hat{a}$)                    // cf. Section 3.1
2 $\mathcal{R} \leftarrow \emptyset$
3 while not stop ($\mathcal{M}$) do                                      // cf. Section 3.3
4     $e \leftarrow$ selectNextInstance ($\mathcal{M}$)                         // cf. Section 3.2
5     $t \leftarrow$ runExperiment ($\hat{a}$, $e$)                 // Runs $\hat{a}$ on $e$ with time-out $\tau$
6     $\mathcal{R} \leftarrow \mathcal{R} \cup \{(e, t)\}$
7     updateModel ($\mathcal{M}$, $\mathcal{R}$)                               // cf. Section 3.1
8 $s_{\hat{a}} \leftarrow$ predictScore($\mathcal{M}$)                                 // cf. Section 3.1
9 return ($s_{\hat{a}}, \mathcal{R}$)
```

3.1 Solver Model

Let SAT solvers $\mathcal{A}$, benchmark instances $\mathcal{I}$, and runtimes $r : \mathcal{A} \times \mathcal{I} \rightarrow [0, \tau]$ be given. We denote the new solver to be ranked by $\hat{a} \notin A$ and define $\hat{A} := A \cup \{\hat{a}\}$ to denote all solvers, including the new solver. Model initialization (Line 1) creates the prediction model and transforms runtimes, as discussed in the subsequent section. No model training is needed at this point since we do not have any labeled data yet. Later, model $\mathcal{M}$ provides a runtime-label prediction function $f : \hat{\mathcal{A}} \times \mathcal{I} \rightarrow \mathbb{R}$ that powers the decisions within our framework. In particular, the model update (Line 7) uses all acquired runtimes to train f with a standard supervised machine-learning methodology. Training features are instance features and known runtimes r. After applying the necessary transformations described in the subsequent section, we fit the prediction model using the transformed observations $\mathcal{R}$ as ground-truth labels. Note that we build a new prediction model in each iteration since running experiments (Line 5) dominates the runtime of Algorithm 1 by magnitudes. Finally, we predict the score of the new solver $\hat{a}$ in Line 8 with the prediction function f.

Runtime Transformation. We transform the runtimes into discrete runtime labels on a per-instance basis. For each instance $e \in \mathcal{I}$, we use a clustering algorithm to assign the runtimes in $\{r(a, e) \mid a \in A\}$ to one of k clusters $C_1, \dots, C_k$ such that the fastest runtimes for instance e are in cluster C_1 and the slowest are in cluster C_{k-1}. Timeouts τ always form a separate cluster C_k. The runtime transformation function $\gamma_k : \mathcal{A} \times \mathcal{I} \rightarrow \{1, \dots, k\}$ is then specified as follows:

$$\gamma_k(a, e) = j \;\Leftrightarrow\; r(a, e) \in C_j$$

Given an instance $e \in \mathcal{I}$, a solver $a \in A$ belongs to the $\gamma_k(a, e)$-fastest solvers on instance e. In preliminary experiments, we achieved higher accuracy in predicting such discrete-runtime labels than in predicting raw runtimes. Research on portfolio solvers has also shown that discretization works well in practice [3,25].

Ranking Solvers. To determine solver ranks, we use the transformed runtimes $\gamma_k(a, e)$ in the adapted scoring function $s_k : A \rightarrow [1, 2 \cdot k]$ as follows:

$$s_k(a) := \frac{1}{|\mathcal{I}|} \sum_{e \in \mathcal{I}} \gamma'_k(a, e) \qquad \gamma'_k(a, e) := \begin{cases} 2 \cdot \gamma_k(a, e) & \text{if } \gamma_k(a, e) = k \\ \gamma_k(a, e) & \text{otherwise} \end{cases} \tag{1}$$

I.e., we apply PAR-2 scoring, which is commonly used in SAT competitions [7], on the discrete labels. The scoring function s_k induces a ranking among solvers.

3.2 Instance Selection

Selecting an instance based on the model is a core functionality of our framework (cf. Algorithm 1, Line 4). In this section, we introduce our sampling strategies, which use the model's label-prediction function f and are inspired by existing work within the realms of active learning [29]. We implement an uncertainty and an information-gain sampling strategy. These methods require the model's predictions to include probabilities for the k discrete runtime labels induced by the clustering described in the preceding section. Let $f' : \hat{\mathcal{A}} \times \mathcal{I} \rightarrow [0, 1]^k$ denote this modified prediction function.

Uncertainty Sampling. The uncertainty sampling strategy simply selects the instance closest to the model's decision boundary. The set $\tilde{\mathcal{I}} \subseteq \mathcal{I}$ denotes the instances that have already been sampled.

$$\operatorname*{arg\,min}_{e \in \mathcal{I} \setminus \tilde{\mathcal{I}}} \left| \frac{1}{k} - \max_{n \in \{1, \ldots, k\}} f'(\hat{a}, e)_n \right|$$

Information-Gain Sampling. The information-gain sampling strategy selects the instance with the highest expected entropy reduction regarding the runtime labels of the instance. To be more specific, we select the instance $e \in \mathcal{I} \setminus \tilde{\mathcal{I}}$ that maximizes $IG(e)$, which is specified as follows:

$$\mathrm{IG}(e) := \mathrm{H}(e) - \sum_{n=1}^{k} f'(\hat{a}, e)_n \, \hat{\mathrm{H}}_n(e)$$

Here, $\mathrm{H}(e)$ denotes the entropy of the runtime labels $\gamma(a, e)$ over all $a \in \mathcal{A}$ and $\mathrm{H}(e, n)$ denotes the entropy of these labels plus n as the runtime label for $\hat{a}$. The term $\hat{\mathrm{H}}_n(e)$ is computed for every possible runtime label $n \in \{1, \ldots, k\}$. By maximizing information gain, we select instances that identify solvers with similar behavior.

3.3 Stopping Criteria

In this section, we present two dynamic stopping criteria, the Wilcoxon and the ranking stopping criterion (cf. Algorithm 1, Line 3).

Wilcoxon Stopping Criterion. The Wilcoxon stopping criterion stops the active-learning process when we are confident enough that the predicted runtime labels of the new solver are sufficiently different from the labels of the existing solvers. This criterion is loosely inspired by Matricon et. al. [20]. We use the average p-value $W_{\hat{a}}$ of a Wilcoxon signed-rank test $w(S, P)$ of the two runtime label distributions $S = \{\gamma(a, e) \mid e \in \mathcal{I}\}$ for an existing solver a and $P = \{f(\hat{a}, e) \mid e \in \mathcal{I}\}$ for the new solver $\hat{a}$:

$$W_{\hat{a}} := \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \mathrm{w}(S, P)$$

To improve the stability of this criterion, we use an exponential moving average to smooth out outliers and stop as soon as $W_{\exp}^{(i)}$ drops below a fixed threshold:

$$\begin{aligned} W_{\exp}^{(0)} &:= 1 \\ W_{\exp}^{(i)} &:= \beta W_{\hat{a}} + (1 - \beta)\, W_{\exp}^{(i-1)} \end{aligned}$$

Ranking Stopping Criterion. The ranking stopping criterion is less sophisticated in comparison. It stops the active-learning process if the ranking induced by the model's predictions (Equation 1) remains unchanged within the last l iterations. Note that the concrete values of the predicted score $s_{\hat{a}}$ might still change. We are solemnly interested in the induced ranking in this case.

4 Experimental Design

Given all the previously presented instantiations for Algorithm 1, this section briefly outlines our experimental design, including our evaluation framework, used data sets, hyper-parameter choices, and implementation details.

4.1 Evaluation Framework

As already stated in the introductory section, this work addresses the *New-Solver Problem* (cf. Definition 1). We create a prediction model $\mathcal{M}$ as described in Section 3.1 that provides us with a scoring function s_k.

To evaluate a concrete instantiation of Algorithm 1, i.e., a concrete choice for all the sub-routines, we perform cross-validation on our set of solvers. Algorithm 2 shows this. That means that each solver plays the role of the new solver once (Line 2). Note that the *new* solver in each iteration is excluded from the set of solvers $\mathcal{A}$ to avoid data leakage (Line 3). After running our active-learning framework for a solver $\hat{a}$ in Line 4, we compute the value of both our optimization goals: First and foremost, we want to provide the engineer of new SAT solvers with an accurate ranking. We define the *ranking accuracy* $O_{\text{acc}} \in [0, 1]$ (higher is better) by the fraction of pairs $(\hat{a}, a)$ for all $a \in \mathcal{A}$ that are decided correctly by the given ranking regarding the PAR-2 scores par_2 (Lines 5-8). Second, we also

Algorithm 2: Evaluation Framework

```
   Input: Solvers $\mathcal{A}$, Instances $\mathcal{I}$, Runtimes $r : \mathcal{A} \times \mathcal{I} \to [0, \tau]$
   Output: Average Rank Accuracy $\bar{O}_{\text{acc}}$, Average Fraction of Runtime $\bar{O}_{\text{rt}}$
 1 $O \leftarrow \emptyset$
 2 for $\hat{a} \in \mathcal{A}$ do
 3     $\mathcal{A}' \leftarrow \mathcal{A} \setminus \{\hat{a}\}$
 4     $(s_{\hat{a}}, \mathcal{R}) \leftarrow \text{runALAlgorithm}(\mathcal{A}', \mathcal{I}, r, \hat{a})$        // Refer to Algorithm 1
       // Determine Rank Accuracy
 5     $O_{\text{acc}} \leftarrow 0$
 6     for $a \in \mathcal{A}$ do
 7         if $\big(s_k(a) - s_{\hat{a}}\big) \cdot \big(\text{par}_2(a) - \text{par}_2(\hat{a})\big) > 0$ then
 8             $O_{\text{acc}} \leftarrow O_{\text{acc}} + \frac{1}{|\mathcal{A}|}$
       // Determine Runtime Fraction
 9     $r \leftarrow \sum_{e \in \mathcal{I}} r(\hat{a}, e)$
10     $O_{\text{rt}} \leftarrow 0$
11     for $e \in \mathcal{I}$ do
12         if $\exists t, (e, t) \in \mathcal{R}$ then
13             $O_{\text{rt}} \leftarrow O_{\text{rt}} + \frac{t}{r}$
14     $O \leftarrow O \cup \big\{(O_{\text{acc}}, O_{\text{rt}})\big\}$
15 $\big(\bar{O}_{\text{acc}}, \bar{O}_{\text{rt}}\big) \leftarrow \text{average}(O)$
16 return $\big(\bar{O}_{\text{acc}}, \bar{O}_{\text{rt}}\big)$
```

want to optimize for runtime. The *fraction of runtime* that the algorithm needs to arrive at its conclusion is denoted by $O_{\text{rt}} \in [0, 1]$ (lower is better). This metric puts the runtime summed over the sampled instances in relation to the runtime summed over all instances in the dataset (Lines 9-13). Finally, we compute averages of the output metrics in Line 15 after we have collected all cross-validation results in Line 14. Overall, we want to find an approach that maximizes

$$O_\delta := \delta O_{\text{acc}} + (1 - \delta)\,(1 - O_{\text{rt}}) \quad , \tag{2}$$

whereby $\delta \in [0, 1]$ allows for linear weighting between the two optimization goals O_{acc} and O_{rt}. Plotting the approaches that maximize O_δ for all $\delta \in [0, 1]$ on a O_{rt}-O_{acc}-diagram provides us with a Pareto front of the best approaches for different optimization-goal weightings.

4.2 Data

In our experiments, we work with the dataset of the SAT Competition 2022 Anniversary Track [2]. The dataset consists of 5355 instances with respective runtime data of 28 sequential SAT solvers. We also use a database of 56 instance features[3] provided in the GBD Benchmark Database (GBD) by Iser et al. [16]. They comprise instance size features and node distribution statistics for several graph representations of SAT instances, among others, and are primarily inspired by the SATzilla 2012 features [35]. All features are numeric and free of missing values. We drop 10 out of 56 features because of zero variance. Overall, prediction models have access to 46 instances features and 27 runtime features, i.e., excluding the current new solver $\hat{a}$. Additionally, we retrieve instance-family information[4] to evaluate the composition of our sampled benchmarks. Instance families comprise instances derived from the same application domain, e.g., planning, cryptography, etc., and are a valuable tool for analyzing solver performance and portfolios.

For hyper-parameter tuning, we randomly sample 10 % of the complete set of 5355 instances with stratification regarding the instance's family. All instance families that are too *small*, i.e., 10 % of them corresponds to less than one instance, are put into one meta-family for stratification. This *tuning dataset* allows for a more extensive exploration of hyper-parameter space.

4.3 Hyper-parameters

Given Algorithm 1, there are several possible instantiations for the three phases, i.e., *ranking*, *selection*, and *stopping*. Also, there are different choices for the runtime-label prediction model and runtime discretization. The experiment configurations are given below.

Ranking. Regarding *ranking* (cf. Section 3.1), we experiment with the following approaches, including our used hyper-parameter values:

- Observed PAR-2 ranking of already sampled instances
- Predicted runtime-label ranking
  - History size: consider the latest 1, 10, 20, 30, or 40 predictions within a voting approach for stability. The latest x predictions for each instance vote on the instance's winning label.
  - Fallback threshold: if the difference of scores between the new solver $\hat{a}$ and another solver drops below 0.01, 0.05, or 0.1, use the partially observed PAR-2 ranking as a tie-breaker.

Selection. For *selection* (cf. Section 3.2), we experiment with the following methods, including our used hyper-parameter values. Since the potential runtime of experiments is by magnitudes larger than the model's update time, we only consider incrementing our benchmark by one instance at a time rather than using batches, which is also proposed in current active-learning advances [30,32]. A drawback of this is the lack of parallel execution of runtime experiment.

[3] https://gbd.iti.kit.edu/getdatabase/base_db
[4] https://gbd.iti.kit.edu/getdatabase/meta_db

- Random sampling
- Uncertainty sampling
  - Fallback threshold: Use random sampling for the first 0 %, 5 %, 10 %, 15 %, or 20 % of instances to explore the instance space.
  - Runtime scaling: Whether to normalize uncertainty scores per instance by the average runtime of solvers on it or use the absolute values.
- Information-gain sampling
  - Fallback threshold: Use random sampling for the first 0 %, 5 %, 10 %, 15 %, or 20 % of instances to explore the instance space.
  - Runtime scaling: Whether to normalize information-gain scores per instance by the average runtime of solvers on it or use the absolute values.

Stopping. For *stopping* decisions (cf. Section 3.3), we experiment with the following criteria, including our used hyper-parameter values:

- Subset-size stopping criterion, using 10 % or 20 % of instances
- Ranking stopping criterion
  - Minimum amount: Sample at least 2 %, 8 %, 10 %, or 12 % of instances before applying the criterion.
  - Convergence duration: Stop if the predicted ranking stays the same for a number of sampled instances equal to 1 % or 2 % of all instances.
- Wilcoxon stopping criterion
  - Minimum amount: Sample at least 2 %, 8 %, 10 %, or 12 % of instances before applying the criterion.
  - Average of p-values to drop below: 5 %.
  - Exponential-moving average: Incorporate previous significance values by using an EMA with $\beta = 0.1$ or $\beta = 0.7$.

Prediction model. We only use one model configuration for runtime-label prediction in our experiments since an exhaustive grid search would be infeasible. In preliminary experiments, we compared various model types from *scikit-learn* [27]. In particular, we conducted nested cross-validation, including hyper-parameter tuning, and used Matthews Correlation Coefficient [11,21] to assess the performance for predicting runtime labels. Our final choice is a stacking ensemble of a quadratic-discriminant analysis and a random forest, using a decision tree to combine its ensemble members.

Runtime discretization. To define prediction targets, i.e., discretized runtime labels, we use hierarchical clustering with $k = 3$ and a log-single-link criterion, which produced the most *useful* labels in preliminary experiments. We denote this adapted solver scoring function with s_3. Other clustering approaches that we have tried include hierarchical clustering with mean-, median- and complete-link criteria, as well as k-means and spectral clustering. In our chosen hierarchical procedure, each non-time-out runtime starts in a separate interval. We then gradually merge intervals whose single-link logarithmic distance is the smallest until the desired number of partitions is reached.

To obtain *useful* labels, we need to ensure that discretized labels still discriminate solvers and align with the actual PAR-2 ranking. We analyzed the ranking induced by s_3 in preliminary experiments with the SAT Competition 2022 Anniversary Track [2]. According to a Wilcoxon-signed-rank test with $\alpha = 0.05$, 87.83 % of solver pairs have significantly different scores after discretization, only a slight drop compared to 89.95 % before discretization. Further, our ranking approach correctly decides for almost all (about 97.45 %; $\sigma = 3.68\,\%$) solver pairs which solver is faster. In particular, the Spearman correlation of s_3 and PAR-2 ranking is about 0.988, which is very close to the optimal value of 1 [5]. All these results show that discretized runtimes are suitable for our framework.

4.4 Implementation Details

For reproducibility, our source code and data are available on GitHub[5]. Our code is implemented in Python using *scikit-learn* [27] for making predictions and *gbd-tools* [16] for SAT-instance retrieval.

5 Evaluation

In this section, we evaluate our active-learning framework. First, we analyze and tune the different components of our framework on the tuning dataset. Next, we evaluate the best configurations with the full dataset. Finally, we analyze the importance of different instance families to our framework.

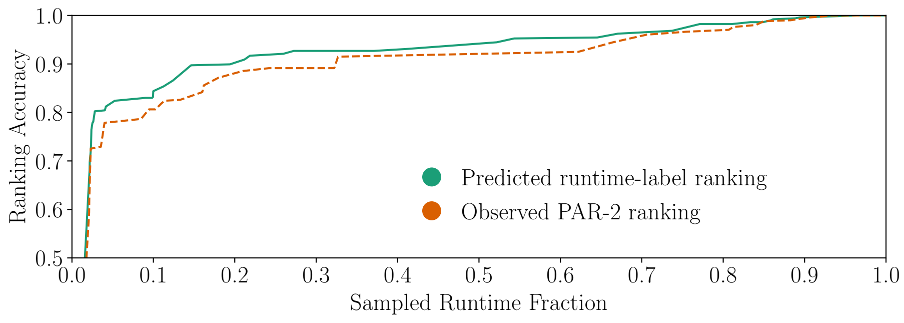
5.1 Hyper-Parameter Analysis

Our experiments follow the evaluation framework introduced in Section 4.1. Fig. 1 shows the performance of the approaches from Section 4.3 on O_{rt}-O_{acc}-diagrams for the hyper-parameter-tuning dataset. Evaluating a particular configuration with Algorithm 2 returns a point (O_{rt}, O_{acc}). We do not show intermediate results of the active-learning procedure but only the final results after stopping. The plotted lines represent the best performing configurations (convex hull) per top-level hyper-parameter choice, i.e., per ranking approach (Fig. 1a), selection approach (Fig. 1b), and stopping criterion (Fig. 1c).

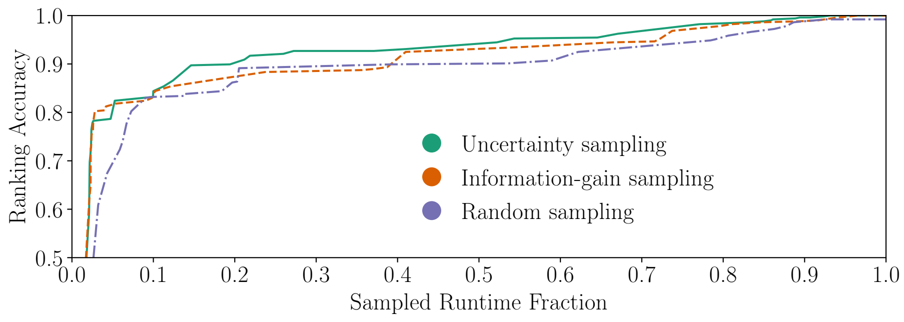
Regarding ranking approaches (Fig. 1a), using the s_3-induced runtime-label ranking consistently outperforms the partially observed PAR-2 ranking for each possible value of the trade-off parameter δ. This outcome is expected since selection decisions are not random. For example, we might sample more instances of one family if it benefits discrimination of solvers. While the partially observed PAR-2 score is skewed, the prediction model can account for this.

Regarding the selection approaches (Fig. 1b), uncertainty sampling performs best in most cases. However, information-gain sampling is beneficial if runtime is

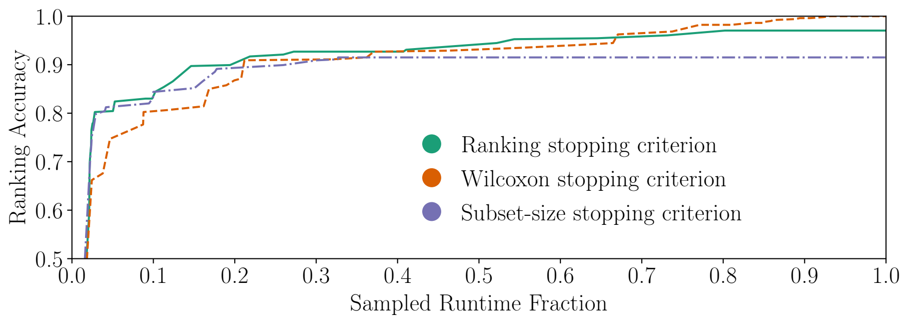
[5] temporary, anonymized version for review: xxx



(a) Ranking approaches

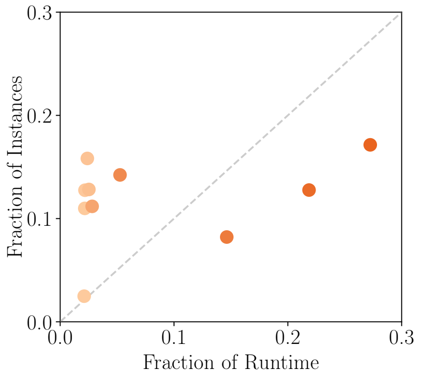


(b) Selection approaches

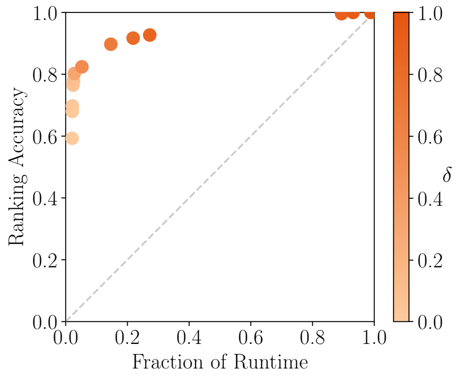


(c) Stopping criteria

Fig. 1: O_{rt}-O_{acc}-diagrams comparing different hyper-parameter instantiations of our active-learning framework. The x-axis shows the ratio of total solver runtime on the sampled instances relative to all instances. The y-axis shows the ranking accuracy (cf. Section 4.1). Each line entails the front of Pareto-optimal configurations for the respective hyper-parameter instantiation.



(a) Runtime vs. Instances



(b) Runtime vs. Accuracy

Fig. 2: Scatter plot comparing different instantiations of δ for our active-learning strategy on the hyper-parameter-tuning dataset. The x-axis shows the fraction of runtime O_{rt} of the sample, while the y-axes show the fraction of instances sampled and ranking accuracy, respectively. The color indicates the weighting between different optimization goals $\delta \in [0, 1]$. The larger δ, the more we favor accuracy over runtime.

strongly favored (small δ; runtime fraction less than 5 %). This result aligns with our expectations: Information-gain sampling selects instances that maximize the expected reduction in entropy. This means we sample instances revealing similarities between solvers rather than differences, which helps to build a confident model fast. However, the method lacks the possibility of selecting helpful instances for distinguishing solvers later. Random sampling performs reasonably well but is outperformed by uncertainty sampling in all cases, showing the benefit of actively selecting instances based on a prediction model.

Regarding the stopping criterion (Fig. 1c), the ranking stopping criterion performs the most consistently well. If accuracy is strongly favored (very high δ), the Wilcoxon stopping criterion performs better. The subset-size stopping criterion performs reasonably well but does not improve beyond a certain accuracy because of sampling a fixed subset of instances.

Figure 2a shows an interesting consequence of weighting our optimization goals: If we, on the one hand, desire to get a *rough* estimate of a solver's performance fast (low δ), approaches favor selecting many *easy* instances. In particular, the fraction of sampled instances is larger than the fraction of runtime. By having many observations, it is easier to build a model. If we, on the other hand, desire to get a *good* estimate of a solver's performance in a moderate amount of time (high δ), approaches favor selecting fewer *difficult* instances. In particular, the fraction of instances is smaller than the fraction of runtime.

Furthermore, Figure 2b reveals which values make the most sense for δ. The range $\delta \in [0.2, 0.8]$, thereby, corresponds to the points with a runtime fraction

Table 2: Performance comparison of the best-performing active-learning approaches (*AL*), random sampling of the same runtime fraction with 1000 repetitions (*Random*) and statically selecting the instances that are most frequently sampled by active-learning approaches (*Most Freq.*)

(a) Best-performing AL approach for $\delta \in [0.2, 0.7]$

	AL	Random	Most Freq.
Sampled Runtime Fraction (%)	5.41	5.43	5.44
Sampled Instance Fraction (%)	26.53	5.43	27.75
Ranking Accuracy (%)	90.48	88.54	81.08

(b) Best-performing AL approach for $\delta \in (0.7, 0.8]$

	AL	Random	Most Freq.
Sampled Runtime Fraction (%)	10.35	10.37	10.37
Sampled Instance Fraction (%)	5.24	10.37	36.96
Ranking Accuracy (%)	92.33	91.61	84.52

between 0.03 and 0.22 We consider this region to be most promising, analogous to the *elbow* method in cluster analysis [17].

5.2 Full-Dataset Evaluation

Having selected the most promising hyper-parameters, we run our active-learning experiments on the complete Anniversary Track dataset (5355 instances). The aforementioned range $\delta \in [0.2, 0.8]$ is covered by only two configurations. The best-performing approach for $\delta \in [0.2, 0.7]$ uses the predicted runtime-label ranking, information-gain sampling, and ranking stopping criterion. It can predict a new solver's PAR-2 ranking with about 90.48 % accuracy (O_{acc}), while only needing 5.41 % of the full evaluation time (O_{rt}). The best-performing approach for $\delta \in (0.7, 0.8]$ uses the predicted runtime-label ranking, uncertainty sampling, and ranking stopping criterion. It can predict a new solver's PAR-2 ranking with about 92.33 % accuracy (O_{acc}), while only needing 10.35 % of the full evaluation time (O_{rt}).

Table 2 shows how both active-learning approaches compare against static baselines. The first column shows the two aforementioned AL configurations. We compare it against a random baseline and a static benchmark set consisting of the most frequently sampled instances by active learning. The random baseline (*random*) samples instances until roughly the same fraction of runtime is in the benchmark set. We repeat this 1000 times and report average results. The benchmark set of the most frequently AL-sampled instances (*Most Freq.*) uses the average sampling frequencies over all solvers using all Pareto-optimal active-learning approaches.

Our AL approaches perform slightly better compared to random sampling. However, differences are not significant regarding a Wilcoxon signed-rank test

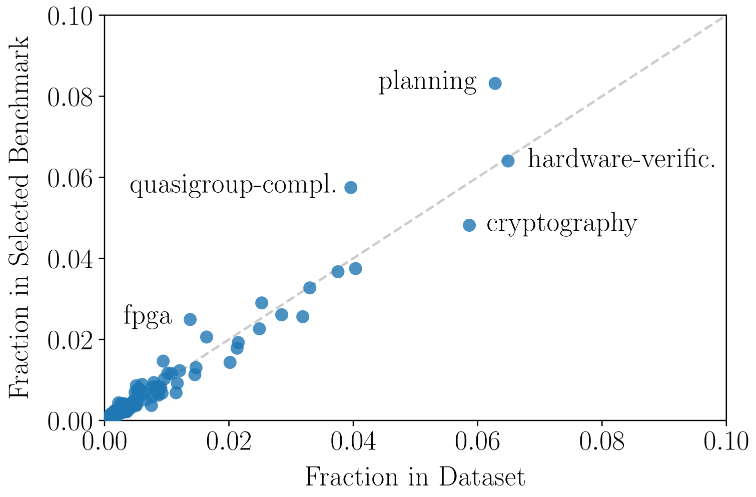


Fig. 3: Scatter plot showing the *importance* of different instance families to our framework. The x-axis shows the frequency of instance families in the full Anniversary Track dataset. The y-axis shows the average frequency of instance families in the samples selected by active learning. The dashed line represents families that occur with the same frequency in both, the dataset and samples.

with $\alpha = 0.05$. Note that the performance difference between random and non-random sampling changes for different fractions of sampled runtime (cf. Figure 1b). The static benchmark using the most frequently AL-sampled instances performs poorly, though, in comparison to active learning and random sampling. This is somewhat expected since it does not reflect the right balance of instance families: Families whose instances are uniform-randomly selected by AL appear less often in this benchmark than families where some instances are sampled more often than others. While the active-learning results are less strong on the full dataset than on the smaller tuning dataset, it still shows the benefit of making benchmark selection dependent on the solvers to distinguish.

5.3 Instance-Family Importance

Selection decisions of our approach also reveal the importance of different instance families to our framework. Figure 3 shows the occurrence of instance families within the dataset and the benchmarks created by active learning. We use the best-performing configurations for all $\delta \in [0, 1]$ and examine the selection decisions by the active-learning approach on the SAT Competition 2022 Anniversary Track dataset [2]. While most families appear with the same fraction in the dataset and the sampled benchmarks, a few outliers need further discussion. Problem instances of the families *fpga*, *quasigroup-completion*, and *planning* are especially helpful to our framework in distinguishing solvers. Instances of these families are selected over-proportionally in comparison to the full dataset. In contrast, instances of the largest family, i.e., *hardware-verification*, roughly appear with the same fraction in the dataset and the sampled benchmarks. Finally, instances of the family *cryptography* are less important in distinguishing solvers than their vast weight in the dataset suggests. A possible explanation is that these instances are very similar, such that a small fraction of them is sufficient to estimate a solver's performance on all of them.

6 Conclusion

In this work, we have discussed possible solutions to the *New-Solver Problem*: Given a new solver, we want to find its ranking amidst its competitors. Our approach provides accurate ranking predictions while needing significantly less runtime than a full evaluation on a given benchmark set. In particular, we can determine a new solver's PAR-2 ranking with about 92 % accuracy while only needing 10 % of the full-evaluation time. Our framework produces good results working with discrete runtime labels rather than regressing actual runtimes. We have evaluated several ranking algorithms, instance-selection approaches, and stopping criteria within our sequential active-learning process. We also took a brief look at which instance families are the most prevalent in selection decisions.

6.1 Future Work

Future work may compare further ranking algorithms, instance-selection approaches, and stopping criteria. Furthermore, it is possible to formulate runtime discretization as an optimization problem, directly selecting the most discriminatory discretization technique rather than making an empirical comparison.

A major shortcoming of our current approach is the lack of parallelization, selecting instances one at a time. Benchmarking on a computing cluster with n cores benefits from having batches of n instances. However, bigger batch sizes n impede *active learning*. Also, it is unclear how to synchronize the model update and instance selection without wasting too much runtime.

On a more general note, one can apply our evaluation framework to arbitrary $\mathcal{NP}$-complete problems, as all discussed active-learning methods are problem-agnostic. Especially the runtime-prediction model would need some tweaking for this purpose, though. Those problems share most of the relevant properties of SAT solving, i.e., there are established instance features, a complete benchmark run takes quite some time, and solver development traditionally requires expert knowledge to hand-select instances.

References

1. Balint, A., Belov, A., Järvisalo, M., Sinz, C.: Overview and analysis of the SAT Challenge 2012 solver competition. Artif. Intell. **223**, 120–155 (2015). https://doi.org/10.1016/j.artint.2015.01.002

2. Balyo, T., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.): Proceedings of SAT Competition 2022: Solver and Benchmark Descriptions. Department of Computer Science, University of Helsinki (2022), http://hdl.handle.net/10138/347211
3. 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
4. Dang, N., Akgün, Ö., Espasa, J., Miguel, I., Nightingale, P.: A framework for generating informative benchmark instances. In: Proc. CP. pp. 18:1–18:18 (2022). https://doi.org/10.4230/LIPIcs.CP.2022.18
5. De Winter, J.C.F., Gosling, S.D., Potter, J.: Comparing the pearson and spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. Psychol. Methods **21**(3), 273–290 (2016). https://doi.org/10.1037/met0000079
6. Dehghani, M., Tay, Y., Gritsenko, A.A., Zhao, Z., Houlsby, N., Diaz, F., Metzler, D., Vinyals, O.: The benchmark lottery. arXiv:2107.07002 [cs.LG] (2021), https://arxiv.org/abs/2107.07002
7. 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
8. Garzón, I., Mesejo, P., Giráldez-Cru, J.: On the performance of deep generative models of realistic SAT instances. In: Proc. SAT. pp. 3:1–3:19 (2022). https://doi.org/10.4230/LIPIcs.SAT.2022.3
9. Gelder, A.V.: Careful ranking of multiple solvers with timeouts and ties. In: Proc. SAT. pp. 317–328 (2011). https://doi.org/10.1007/978-3-642-21581-0_25
10. Golbandi, N., Koren, Y., Lempel, R.: Adaptive bootstrapping of recommender systems using decision trees. In: Proc. WSDM. pp. 595–604 (2011). https://doi.org/10.1145/1935826.1935910
11. 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
12. Harpale, A., Yang, Y.: Personalized active learning for collaborative filtering. In: Proc. SIGIR. pp. 91–98 (2008). https://doi.org/10.1145/1390334.1390352
13. Hoos, H.H., Hutter, F., Leyton-Brown, K.: Automated configuration and selection of SAT solvers. In: Handbook of Satisfiability, chap. 12, pp. 481–507. IOS Press, 2 edn. (2021). https://doi.org/10.3233/FAIA200995
14. Hoos, H.H., Kaufmann, B., Schaub, T., Schneider, M.: Robust benchmark set selection for boolean constraint solvers. In: Proc. LION. pp. 138–152 (2013). https://doi.org/10.1007/978-3-642-44973-4_16
15. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. LION. pp. 507–523 (2011). https://doi.org/10.1007/978-3-642-25566-3_40
16. Iser, M., Sinz, C.: A problem meta-data library for research in SAT. In: Proc. PoS. pp. 144–152 (2018). https://doi.org/10.29007/gdbb
17. Kodinariya, T.M., Makwana, P.R.: Review on determining number of cluster in k-means clustering. Int. J. Adv. Res. Comput. Sci. Manage. Stud. **1**(6), 90–95 (2013)
18. Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009). https://doi.org/10.1109/MC.2009.263
19. Manthey, N., Möhle, S.: Better evaluations by analyzing benchmark structure. In: Proc. PoS (2016), http://www.pragmaticsofsat.org/2016/reg/POS-16_paper_4.pdf

20. Matricon, T., Anastacio, M., Fijalkow, N., Simon, L., Hoos, H.H.: Statistical comparison of algorithm performance through instance selection. In: Proc. CP. pp. 43:1–43:21 (2021). https://doi.org/10.4230/LIPIcs.CP.2021.43
21. 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
22. Mısır, M.: Data sampling through collaborative filtering for algorithm selection. In: Proc. IEEE CEC. pp. 2494–2501 (2017). https://doi.org/10.1109/CEC.2017.7969608
23. Mısır, M.: Benchmark set reduction for cheap empirical algorithmic studies. In: Proc. IEEE CEC. pp. 871–877 (2021). https://doi.org/10.1109/CEC45853.2021.9505012
24. Mısır, M., Sebag, M.: ALORS: An algorithm recommender system. Artif. Intell. **244**, 291–314 (2017). https://doi.org/10.1016/j.artint.2016.12.001
25. Ngoko, Y., Cérin, C., Trystram, D.: Solving SAT in a distributed cloud: A portfolio approach. Int. J. Appl. Math. Comput. Sci. **29**(2), 261–274 (2019). https://doi.org/10.2478/amcs-2019-0019
26. Nießl, C., Herrmann, M., Wiedemann, C., Casalicchio, G., Boulesteix, A.: Over-optimism in benchmark studies and the multiplicity of design and analysis options when interpreting their results. WIREs Data Min. Knowl. Discov. **12**(2) (2022). https://doi.org/10.1002/widm.1441
27. 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**(85), 2825–2830 (2011), http://jmlr.org/papers/v12/pedregosa11a.html
28. Rubens, N., Elahi, M., Sugiyama, M., Kaplan, D.: Active learning in recommender systems. In: Recommender Systems Handbook, chap. 24, pp. 809–846. Springer, 2 edn. (2015). https://doi.org/10.1007/978-1-4899-7637-6_24
29. Settles, B.: Active learning literature survey. Tech. rep., University of Wisconsin-Madison, Department of Computer Sciences (2009), http://digital.library.wisc.edu/1793/60660
30. Sinha, S., Ebrahimi, S., Darrell, T.: Variational adversarial active learning. In: Proc. ICCV. pp. 5971–5980 (2019). https://doi.org/10.1109/ICCV.2019.00607
31. Stützle, T., López-Ibáñez, M., Pérez-Cáceres, L.: Automated algorithm configuration and design. In: Proc. GECCO. pp. 997–1019 (2022). https://doi.org/10.1145/3520304.3533663
32. Tran, T., Do, T., Reid, I.D., Carneiro, G.: Bayesian generative active deep learning. In: Proc. ICML. pp. 6295–6304 (2019), http://proceedings.mlr.press/v97/tran19a.html
33. Volpato, R., Song, G.: Active learning to optimise time-expensive algorithm selection. arXiv:1909.03261 [cs.LG] (2019), https://arxiv.org/abs/1909.03261
34. 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
35. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Features for SAT. Tech. rep., University of British Columbia (2012), https://www.cs.ubc.ca/labs/beta/Projects/SATzilla/Report_SAT_features.pdf