Research Article

The European Journal on Artificial Intelligence
1–17

DOI: 10.1177/30504554261453277
journals.sagepub.com/home/eai

Instance Selection Methods in Automated Algorithm Configuration

Marie Anastacio[1,2], Théo Matricon[3] and Holger H. Hoos[1,2,4]

Abstract

Empirical performance evaluation is crucial for algorithm configuration and performance optimization. Prior work showed that comparing the running time of two algorithms can be accelerated by evaluating them on strategically selected instances. We explore this approach in the context of automated algorithm configuration, adapting prior methods to leverage empirical performance models and introducing two active learning-inspired methods. We evaluate these methods on two performance comparison situations arising during configuration, achieving speedups of 5 to 3,000 times over the random instance sampling method of state-of-the-art configurators. We then integrate the best methods into the model-based configurator sequential model-based algorithm configurator (SMAC). In two of five running time optimization scenarios, we nearly double the performance gain of SMAC. An ablation study confirms that instance selection drives this improvement, indicating substantial potential for advancing algorithm configuration.

Keywords

automated algorithm configuration, empirical performance comparison, empirical performance models, running time optimization

Received: May 06, 2025; accepted: May 04, 2026

1 Introduction

Many state-of-the-art solvers for non-deterministic polynomial-time hard (NP-hard) problems, such as Boolean satisfiability (SAT) (Falkner et al., 2015; Xu et al., 2008) or mixed integer programming (MIP) (Hutter et al., 2010; Xu et al., 2011), come with parameters that enable users to adapt the inner workings of the solver to the problem instances they are trying to solve. This process of algorithm configuration has traditionally been conducted manually or through simple search procedures such as random search, which is still the approach used in many applications, despite the ready availability of tools for efficiently automating the configuration process. The question of finding which parameter values should be used for an algorithm to perform well on a given set of problem instances is known as the *automated algorithm configuration (AAC) problem*.

The AAC problem can be formally defined as follows (see, e.g., Hoos, 2012).

Given a target algorithm A with k parameters $p_1, p_2, \ldots, p_k$, and, for each parameter p_j, a domain $\mathcal{D}_j$ of possible values and a default value $D_j \in \mathcal{D}_j$; a configuration space Ω, containing all valid combinations of parameter values of A; a set of problem instances $\mathcal{I}$; and a performance metric m that measures the performance of target algorithm A, configured according to $\omega \in \Omega$, on $\mathcal{I}$; find $\omega^* \in \Omega$ that optimizes (in our work, minimizes) the performance of A on instance set $\mathcal{I}$, according to metric m.

[1]LIACS, Leiden University, Leiden, The Netherlands
[2]AIM, RWTH Aachen University, Aachen, Germany
[3]CNRS, LaBRI, Universtité de Bordeaux, Talence, France
[4]Department of Computer Science, University of British Columbia (UBC), Vancouver, Canada

Corresponding Author:
Marie Anastacio, LIACS, Leiden University, Leiden, The Netherlands; AIM, RWTH Aachen University, Aachen, Germany.
Email: anastacio@aim.rwth-aachen.de

Note that typically the running time of the target algorithm is limited by an upper bound defined by means of a cutoff time T_{cut} for A. This limit has an impact on any kind of modeling of the running time, as it gives rise to a saturation phenomenon on the upper bound. Some configuration approaches comprise methods aimed at alleviating the effects of this. For example, Hutter et al. (2011a) proposed to replace the censored data by imputed data when training the empirical performance model.

Comparing the performance of two configurations of a given algorithm is a key element of procedures for solving the AAC problem, since such comparisons are performed many times during the configuration process. However, in an automated algorithm configurator, the most computationally expensive task is to evaluate the quality of candidate parameter configurations. Executing time-consuming runs of the target algorithm on different problem instances to determine which parameter settings achieve the best performance requires substantial resources, and time is often wasted on less promising configurations as well as on instances that require a long running time to solve, regardless of the configuration utilized. With the increasing focus on sustainability, the computational resources and the environmental impact associated with the use of artificial intelligence methods should be put under scrutiny, providing additional incentives not only to configure algorithms but also to reduce the computational cost of AAC. Several lines of research attempt to tackle this problem, mainly focusing on the idea of discarding configurations that are not sufficiently promising. For anytime algorithms, such as machine learning methods, there has been work on early stopping less promising runs based on learning curves (see, e.g., Domhan et al., 2015; Luo et al., 2019), while adaptive capping mechanisms, such as the ones included in paramILS and irace (Hutter et al., 2009; López-Ibáñez et al., 2016), permit the early stopping of evaluations of configurations unlikely to be competitive with previously evaluated ones. Those lines of research are focused on the idea of discarding configurations deemed insufficiently promising.

On the other hand, in our previous work (Matricon et al., 2021), inspired by the field of active learning, we explored the idea of selecting on which instance to compare two given algorithms and introduced the per-set efficient algorithm selection (PSEAS) problem. This problem appears during AAC, albeit in a slightly different form. Rather than selecting an algorithm, the configurator needs to select a specific configuration of an algorithm among others. Building on research from several areas, we try to identify instances that help discriminate between the compared configurations. We argue that carefully selecting instances and avoiding long evaluations that provide only a limited amount of information allow the configurator to decide faster whether or not it should reject less promising configurations.

The PSEAS problem is solved by using empirical performance data from other algorithms as a prior to estimate which instances have more discrimination potential. The comparison of two configurations of a single algorithm during configuration lacks access to this prior knowledge. However, model-based algorithm configurators, such as the sequential model-based algorithm configurator (SMAC), learn an empirical performance model that can serve as a prior for instance selection. Moreover, access to a model allows us to build on active learning methods, since they also address the question of which instance should be evaluated next, albeit with a slightly different goal. Our contributions are as follows:

- We define two phases of comparison within the configuration process. The first phase takes place on a subset of instances on which configurations have been run before, and the second on instances never seen before.
- We adapt the two best-performing selection methods from our previous work (Matricon et al., 2021) with the performance model used in model-based configurators, add two methods inspired by the active learning literature (Gu et al., 2014), and evaluate them on five benchmarks. Our empirical evaluation shows that, depending on the problem instances and their running time distribution, the decision to stop evaluating a less promising configuration can be reached 5 to 3,000 times faster than with random sampling, the method used in current state-of-the-art configurators.
- We implement within SMAC (Hutter et al., 2011b) a statistical test to discard configurations as soon as there is sufficient statistical evidence for doing so, as well as the selection mechanisms that showed the best result on our initial evaluation.
- We evaluate the resulting configurator on five configuration scenarios from the literature. We find that the method shows much potential, almost doubling over the improvement reached by vanilla SMAC on two out of five scenarios.
- We perform an ablation study to verify that the selection mechanism is indeed responsible for the observed improvements. We find that when used in a stand-alone fashion, the statistical testing mechanism we use degrades the performance of SMAC, confirming that the origin of the improvement lies in the selection, or the combined effect of both new mechanisms.

The remainder of this article is organized as follows: Section 2 explains the comparison phases within an automated algorithm configurator and the methods we adapted, Section 3 describes the setup of our computational experiments, the datasets, and the implementation decision we made, Section 4 shows the obtained result when evaluating the methods

separately on the two previously described phases without integrating them into the full configuration process, and Section 5 shows the results obtained as a result of these enhancements.

2 Comparison of Two Configurations

We want to efficiently compare two configurations of a single algorithm. To do so, we need to gather sufficient statistical evidence while using the least possible amount of computing time.

2.1 Instance Selection

Following the definition of AAC in the Introduction, $\mathcal{I}$ is the finite set of instances and Ω is the set of valid configurations of the algorithm at hand. At any given step, we have partial running time information on $\mathcal{I}_{\text{known}} \subseteq \mathcal{I}$ for configurations in $\Omega_{\text{known}} \subseteq \Omega$, which means that for $\omega \in \Omega_{\text{known}}$, there exists information about the performance of ω on at least one instance of $\mathcal{I}_{\text{known}}$.

When comparing a challenger configuration ω_{ch} to the incumbent (i.e., the best currently known) configuration ω_{inc}, instance selection appears in two forms. In Algorithm 1, a high-level description of how SMAC works, these are found in lines 6 and 14 (highlighted in bold-italics), but the same mechanisms arise in any configurator. The first of these, which we name *phase 1*, corresponds to the PSEAS (Matricon et al., 2021), where we already know the performance of ω_{inc} on a set of instances $\mathcal{I}_{\text{known}}$ and want to determine whether ω_{ch} performs better on this set. The second, which we name *phase 2*, corresponds to a case where we know the performance of both ω_{inc} and ω_{ch} on $\mathcal{I}_{\text{known}}$ and want to evaluate both configurations on additional instances from $\mathcal{I} \setminus \mathcal{I}_{\text{known}}$. This can happen, for example, through steadily increasing the size of $\mathcal{I}_{\text{known}}$ at each iteration of the configuration process, or only when we do not have sufficient information to decide which one is the best. Since we would have already discarded the worst challenger, and considering our goal of lowering the number of evaluations of the target algorithm, we will focus on this second case in the following.

Algorithm 1. Intensification for one challenger (based on SMAC Hutter et al., 2011b)
ω_{inc}: incumbent configuration. ω_{ch}: challenger configuration. $n_{\max}$: maximum number of new instances on which to run ω_{inc}.

1: **if** Insufficient runs for configuration ω_{inc} **then**
2: execute a run of ω_{inc} on an instance not run yet, sampled uniformly at random
3: **end if**
4: $N \leftarrow 1$
5: **while** there are instances on which ω_{inc}, but not ω_{ch}, has been run **do**
6: execute runs of ω_{ch} ***on a subset of N instances on which*** ω_{inc} ***has been run, but not*** ω_{ch}
7: **if** ω_{ch} is worse than ω_{inc} **then**
8: **return** ω_{inc}
9: **else** $N \leftarrow N \cdot 2$
10: **end if**
11: **end while**
12: **while** ω_{inc} and ω_{ch} cannot be distinguished **do**
13: **if** $N_{run} < n_{\max}$ **then**
14: run ω_{inc} and ω_{ch} on ***an instance on which neither has been run previously***
15: $N_{run} \leftarrow N_{run} + 1$
16: **else**
17: **return** ω_{inc}
18: **end if**
19: **end while**
20: **return best of** ω_{ch}, ω_{inc}

In both cases, we seek to iteratively choose an instance $I \in \mathcal{I}_{\text{choose}} \subseteq \mathcal{I}$ and gather performance information on it until we satisfy a stopping condition. Figure 1 gives a high-level overview of the main selection loop described in more detail below.

In phase 1, $\mathcal{I}_{\text{known}}$ is the subset of instances on which we have run our incumbent ω_{inc} so far, and ω_{inc} is the best-performing configuration known to us on $\mathcal{I}_{\text{known}}$. At the first step, we have no performance information regarding ω_{ch}. At each step, we select an instance I from $\mathcal{I}_{\text{choose}}$, run ω_{ch} on I and add it to $\mathcal{I}_{\text{selected}}$. At any step, $\mathcal{I}_{\text{selected}} \subseteq \mathcal{I}_{\text{known}}$ and $\mathcal{I}_{\text{choose}} = \mathcal{I}_{\text{known}} \setminus \mathcal{I}_{\text{selected}}$. During this phase, we want to discard ω_{ch}, given sufficient evidence that it performs worse than ω_{inc}. If ω_{ch} performs as well or better than ω_{inc}, we would need to run it on all instances of $\mathcal{I}_{\text{known}}$ before

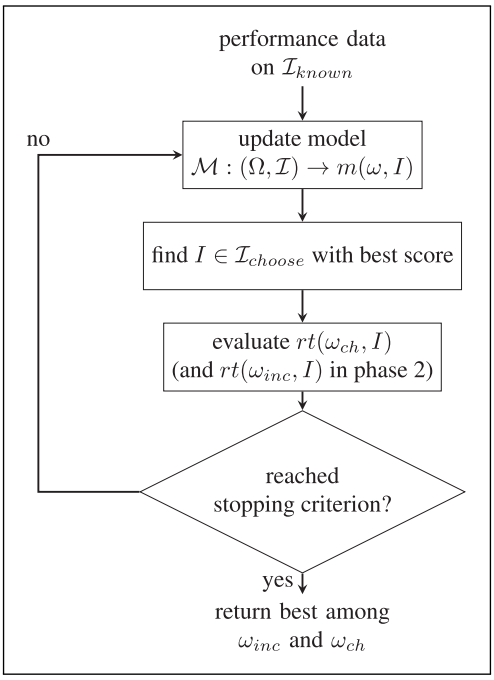

Figure 1. Workflow of our comparison strategy.

applying the second phase or continuing the configuration process, thus we do not discard ω_{inc} early. Moreover, ω_{inc} already showed evidence that it performs better than previously tested challengers and thus comes with stronger evidence of good performance. Thus, our stopping criterion is either to have $\mathcal{I}_{\text{choose}} = \emptyset$, or to be confident that ω_{ch} is worse than ω_{inc}. We consider that, to select instances, we have access to an empirical prediction model (EPM) trained on all pairs $(\omega, I) \in \Omega_{\text{known}} \times \mathcal{I}_{\text{known}}$, such that $m(\omega, I)$ is known, and predicting the performance for any pair of instance and configuration.

In phase 2, we also have a subset $\mathcal{I}_{\text{known}} \subset \mathcal{I}$, but unlike in phase 1, there is no asymmetry between ω_{inc} and ω_{ch}. We know their running time on all instances of $\mathcal{I}_{\text{known}}$ and both can be discarded given sufficient evidence. The goal is to be able to decide which of ω_{inc} and ω_{ch}, whose performance on $\mathcal{I}_{\text{known}}$ cannot be distinguished reliably, actually is to be preferred; to achieve this, we can select instances from $\mathcal{I}_{\text{choose}} = \mathcal{I} \setminus \mathcal{I}_{\text{known}}$ and iteratively add them to $\mathcal{I}_{\text{known}}$. Since no configuration has been run on any of the instances in $\mathcal{I}_{\text{choose}}$, we predict the performance of ω_{inc} and ω_{ch} with a predictive model trained on the performance of the configurations from Ω_{known} on the instances from $\mathcal{I}_{\text{known}}$. To do so, we require instance features, as defined in previous work for a broad range of problems (e.g., for SAT (Xu et al., 2008), MIP (Xu et al., 2011), or traveling salesperson problem (TSP) (Mersmann et al., 2013; Pihera & Musliu, 2014)). In this phase, the stopping criterion is either to be able to clearly separate the performance of ω_{inc} and ω_{ch} on $\mathcal{I}_{\text{known}}$, or to reach a predefined maximum number $n_{\max} \in \mathbb{N}$ of instances added during the process.

2.2 Selection Methods

In prior work (Matricon et al., 2021), we have used selection methods to decide which of two given solvers for an NP-hard problem performs best. To do so, each instance is assigned a score designed to reflect the relevance of choosing that instance both in terms of information obtained and cost incurred. The highest-scoring instance is chosen iteratively until one solver is deemed to have shown better performance than the other. Since we are working with similar types of solvers, we expect that a similar approach would be promising in our situation. We assign scores to instances from $\mathcal{I}_{\text{choose}}$ and select iteratively the highest-scoring instance $I^* \in \text{argmax}_{I \in \mathcal{I}_{\text{choose}}} \text{score}(I)$.

Differing from the context of PSEAS, we have access to an EPM trained on all pairs $(\omega, I) \in \Omega_{\text{known}} \times \mathcal{I}_{\text{known}}$, such that $m(\omega, I)$ is known. Each configuration is represented by a vector of parameter values, and each instance is represented by a vector of features specific to the type of problem at hand. The EPM predicts the performance for any pair of instance and configuration. We use the same type of random forest model as SMAC, previously demonstrated to be most effective for empirical performance prediction (Hutter, Xu et al., 2014).

We adapted two of the best methods tested on PSEAS to support the partial-information context. Note that these methods do not take advantage of the model in phase 1, while in phase 2, they are using the predictions given by the model as if they were ground truth. We did not adapt the information-based method, as it relies on assumptions regarding the performance distribution that could not be made in the current context. For PSEAS, this method relied on distributions estimated on the runs of Ω_{known} on all the instances from $\mathcal{I}$. In our current context, we only have information on $\mathcal{I}_{\text{known}}$, which is influenced heavily by the selection method itself; the estimated distributions would hence be strongly biased. Considering work from the active learning literature applicable to a random forest regression model, we chose to adapt the work of Gu et al., which considers active learning for terrain classification using random forests (Gu et al., 2014). Other works (e.g., Ayerdi & Graña, 2015; Bhosle & Kokare, 2020) have used similar ideas, focusing on the uncertainty of the model, so we also include a measure solely based on uncertainty.

2.2.1 Baseline: Uniform Random Sampling. This is equivalent to assigning every instance the same score, and thus sampling an instance uniformly at random.

2.2.2 Discrimination. This method, originally inspired by the work of Gent et al. (2014), aims to choose the instance that best discriminates between the best and other configurations. We say that a configuration ω is *ρ-dominated* on an instance I, for a given $\rho > 1$, if there exists another configuration ω' such that $m(\omega', I) \leq \rho \cdot m(\omega, I)$. Thus, we define the *discrimination quality* of an instance I, denoted $Q(I)$, as the fraction of known configurations that are ρ-dominated on this instance. The score is then defined as the discrimination quality divided by the mean running time of the instance:

$$\text{score}(I) = \frac{Q(I)}{\text{Mean(I)}}.$$

2.2.3 Variance. This approach is based on the intuition that an instance with high variance is likely to discriminate between two configurations. To also take into account the cost of running this instance, we divide the variance by the mean running time of the instance. Note that, according to the literature (Matricon et al., 2021), the underlying distribution of running times follows a Cauchy distribution and would thus not have a well-defined mean or variance. However, due to running times being bounded by 0 and the cutoff time, it is a truncated Cauchy distribution, which is well-behaved and has a mean and a variance. Our score is thus the relative variance

$$\text{score}(I) = \frac{\text{Var}(I)}{\text{Mean(I)}}.$$

2.2.4 Uncertainty–Diversity–Density. This method is inspired by the work of Gu et al. (2014) from the active learning literature mentioned earlier. We decided to take the core ideas for their classification model and adapt it to our regression model. We named this approach uncertainty–diversity–density (UDD), because it is based on a combination of three scores: uncertainty, diversity, and density. All three scores are scaled and translated to the interval [0; 1] before computing score(I) as

$$\begin{aligned}\text{score}(I) = \text{Uncertainty}(I) \\ + \alpha \cdot \text{Diversity}(I) \\ + \beta \cdot \text{Density}(I).\end{aligned}$$

Uncertainty(I) is the variance of the random forest on running time predictions, for instance I and $\text{Diversity}(I) = -\min_{I' \in \mathcal{I}_{\text{known}}} d(I, I')$, where d is a distance function over instances. Intuitively, the closer I is to instances from $\mathcal{I}_{\text{known}}$, the more unlikely it is to provide additional information. Finally, $\text{Density}(I) = (1/k) \cdot \sum_{I' \in \mathcal{N}_k(I,d)} d(I, I')^2$ where $k \in \mathbb{N}$ is a parameter, d is the same distance function as for diversity, and $\mathcal{N}_k(I, d, \mathcal{I}_{\text{choose}})$ returns the k closest neighbors of I in $\mathcal{I}_{\text{choose}} \setminus \{I\}$ according to d. Intuitively, if an instance I is close to many instances from $\mathcal{I}_{\text{choose}}$, then running I should also provide information about these other instances.

2.2.5 Uncertainty. This corresponds to UDD with α and β set to zero, which is reminiscent of the variance method applied to the predictions of a model instead of the measured performance values.

2.3 Stopping Criterion

At each phase, we need to decide when we consider that sufficient statistical evidence has been gathered. In Algorithm 1, this decision appears in lines 7 and 12. Based on previous works (López-Ibáñez et al., 2016; Matricon et al., 2021), we use a *Wilcoxon matched-pairs signed-rank test* (Conover, 1998) with a significance level of 0.05.

3 Experimental Setup

Our first goal is to include the instance selection methods in a configurator at both phases and to assess the impact of these modifications on the performance. However, directly including them without decomposing the mechanism into smaller, more easily analyzed components would give us little to no information about which of the components show the desired impact.

3.1 Experiments

We divided our evaluation into two main sections, each subdivided into two research questions. First, we evaluated our methods outside the configurator on artificially generated running time data. We conducted experiments to evaluate the performance of the selection methods, separately for phase 1 and phase 2 defined earlier. We designed two sets of experiments aiming to answer the following questions:

1. How does the selection method perform when comparing a new configuration to the incumbent on the subset of instances for which we already collected information throughout the configuration run as seen in phase 1?
2. How does the selection method perform when comparing a new configuration to the incumbent on all instances, selecting instances for which we did not collect information throughout the configuration run as seen in phase 2?

Then, we included the best-performing methods in a state-of-the-art configurator and evaluated their performance. Since SMAC does not include any statistical test to decide when to stop the evaluation of a challenger configuration, we conducted an ablation study to evaluate the impact of the test we newly introduced without instance selection. This allows us to answer the following questions:

1. Do sophisticated instance selection mechanisms allow us to improve over picking instances uniformly at random?
2. How does the introduction of a statistical test during the comparison impact the performance of the configurator—in our case SMAC?

3.2 Datasets

We used configuration scenarios taken from the algorithm configuration library AClib (Hutter, López-Ibáñez et al., 2014) or derived from these. Table 1 provides some details regarding the datasets we used. The number of clusters is computed using the mean-shift (Comaniciu & Meer, 2002) implementation of `scikitlearn` and is used to provide insight into the homogeneity of the respective dataset.

3.2.1 Evaluation Outside of a Configurator. We evaluated our method in two NP-hard problems that have been well-studied in the algorithm configuration literature: SAT and MIP. For each, we chose two widely used datasets from AClib and added a more recent and harder dataset to test the limits of our methods.

For SAT, we used CF and IBM from AClib and generated a new set of cryptography instances based on the work of Nejati and Ganesh (2019) (we used the sha256 encoding, 16 to 60 rounds, and an input size of 2^n with $n \in \mathbb{N}, n \leq 10$). For this last dataset, we set the cutoff time to 5,000 s, such that 70% of the instances can be solved by the default configuration before reaching this time limit. Based on the results of the SAT competition 2020 (Balyo et al., 2020), we decided to configure Kissat (Biere et al., 2020), the best SAT solver currently available.

For MIP, we used RCW2, REG200 from AClib and added a more difficult dataset based on the work of König et al. (2021), which is comprised of challenging neural network verification problems. For this last dataset, we set the cutoff time to 9,000 s, such that 70% of the instances can be solved by the default configuration before reaching this time limit. For these scenarios, we chose CPLEX, since it is well known in the literature and also prominently used in AClib.

3.2.2 Evaluation Inside a Configurator. We chose two TSP scenarios and three MIP scenarios. Because we had to run many different versions of the configurators, we selected well-studied scenarios with a relatively low time budget. *For TSP,* we

Table 1. Characteristics of the Benchmark Instance Sets.

Name	Train Size	Test Size	Features	Clusters
CF	298	301	113	14
IBM	382	302	113	21
Crypto	225	225	103	8
CLS	50	50	148	3
RCW2	495	495	148	6
REG200	999	999	148	2
MIPverify	92	92	206	5
rue-1000-3000	50	250	64	9

used two datasets from AClib (EAX and LKH on rue-1000-3000) due to their short configuration time (see, e.g., Pushak & Hoos, 2020). *For MIP,* we used three datasets from AClib (REG200, CLS and RCW2) well-known from the literature (see, e.g., Hutter et al., 2010). For this scenario, we use a cutoff time of 300 s (following Cáceres et al., 2017), since our method is more suited for situations in which runs might be cut off upon reaching the time limit. Using a cutoff time of 10,000 s results in all runs completing before the cutoff is reached.

3.3 Implementation Details

Our implementations are available on GitHub.[1] We used Python 3.9 with the libraries `ConfigSpace v0.4.20` to define the configuration space and `pyrfr v0.8.0` for the random forest model introduced by Hutter, Xu et al. (2014).

The UDD method requires a distance function d in the instance space; we compute this using the same procedure as Matricon et al. (2021), which finds weights for instance features and computes a weighted feature distance between instances.

Since the discrimination and UDD methods have parameters, we tuned those with a simple grid search on a separate scenario (Kissat with the SWGCP dataset from AClib). For discrimination, we evaluated values in $[1.01; 2]$ with a step size of 0.11 and found that $\rho = 1.12$ performed well on both phases. For UDD, we evaluated values in $[0; 2]$ with a step size of 0.21 for both values independently and found that $\alpha = 0.2$ and $\beta = 1.4$ performed well on both levels.

3.3.1 Evaluation Outside of a Configurator. To carry out our empirical investigation, a dataset of configurations and their associated performance scores were required. To obtain such a set, we generated 100 random configurations uniformly at random for each solver and ran them on all instances of the datasets included in the respective configuration scenarios. This allowed us to collect performance data on many pairs of problem instances and algorithm configurations. We used the same random forest model as in SMAC (Hutter et al., 2011b) as an EPM. We trained this EPM on the previously described performance dataset. To evaluate how efficient our methods will be along a configuration run, we trained the EPM on various amounts of performance data: the number of known configurations is in $[10, 20, 30, 40, 50]$ and the amount of known instances is a fraction of $[0.1, 0.2, 0.3, 0.4, 0.5]$ of the full dataset. This allows us to simulate the growing amount of performance data the EPM is trained on along the configuration run and evaluate how well our selection strategies behave based on it.

3.3.2 Evaluation Inside the Configurator. This evaluation required us to include the selection method in the configurator. As our first evaluation is based on the inner working of SMAC, we included the methods in SMAC3 version 1.1.1. However, in principle, a similar mechanism could be used in any configuration procedure.

We included in SMAC a *Wilcoxon matched-pairs signed-rank test* (Conover, 1998) with a significance level of 0.05 between the runtime of the incumbent ω_{inc} and the challenger ω_{ch} to decide if the challenger can be discarded. Remember that in phase 1, we only discard the challenger in the presence of sufficient evidence that it performs worse than the incumbent and never decide to replace the incumbent based on the test. This means that we take the risk of discarding good configurations in case of error, but would not risk replacing the incumbent with a worse configuration (on known instances).

Due to the large number of statistical tests involved, we need to account for the problem of multiple testing. First, we do not perform tests before running ω_{ch} on at least five instances based on the recommended smallest number of samples for the statistical test to be effective (Conover, 1998). Moreover, we use batches to lower the number of tests performed. For each test, we apply a Bonferroni correction (Dunn, 1961), which means that we divide the significance threshold by the number of tests to be performed (given by the size of $\mathcal{I}_{\text{known}}$ divided by our batch size) before comparing it to our

confidence threshold. Note that if we use a fixed batch size, the larger the number of instances in $\mathcal{I}_{\text{known}}$, the lower the p-value would need to be to reject the null hypothesis. Along the configuration process, more instances are added to $\mathcal{I}_{\text{known}}$, and it would become very unlikely to reject a new incumbent, while the time to compare configurations will become larger due to the number of runs required. This phenomenon would counteract our goal to lower the comparison computation cost. Thus, we decide to set our batch size relatively to the size of $\mathcal{I}_{\text{known}}$. Based on the results of Matricon et al. (2021), we decided to test every 20% of $\mathcal{I}_{\text{known}}$; this corresponds to an amount of instances above which the Wilcoxon test had high accuracy in their reported results for most of the selection methods.

Due to the large computation time required to evaluate every possible combination of methods between phase 1 and phase 2, we had to carefully select a subset of possible experiments; specifically, we only considered the best-performing methods from the first set of experiments at each phase of the configuration. Because our method involves adding a Wilcoxon test to stop comparisons early, we also evaluate its impact separately, to gain further insights into the observed behavior. To compare the performance of two versions of the configurator, we want to look at the expected best performance of the best found configuration. Since a user would typically run the configurator several times and select the configuration found to perform best on the given training data (which corresponds to the so-called standard protocol), we apply the following protocol: we run the configuration eight times with seeds from 1 to 8, repeatedly sample five runs uniformly at random from that set of 8, and identify the best of these according to performance on the training set. We used 1,000 such samples to estimate the probability distribution of the quality of the result produced by each configurator on each configuration scenario. We then compared the medians of these empirical distributions. This is similar to procedures used in the literature (see, e.g., Anastacio & Hoos, 2020; Pushak & Hoos, 2020).

3.4 Execution Environment

The first set of experiments was run on a high-performance compute cluster running CentOS Linux operating system version 8.5. Each node is equipped with two Intel Xeon E5-2683 CPUs with 16 cores and 40 MB cache each, as well as 94 GB RAM. The second set of experiments was run on a high-performance cluster running Rocky Linux operating system version 9.3. Each node is equipped with two AMD EPYC 7543 CPUs with 32 cores and 256 MB of cache each as well as 1 TB of RAM.

4 Evaluation Outside the Configuration Process

This section describes the results obtained from our first set of experiments. The goal of these experiments was to evaluate how well the selection methods perform at both phases described in Section 2, independently of the whole configuration procedure around. We show aggregated results here, but the raw results and scripts to generate more visualizations are available in our Git repository.

4.1 Comparing Configurations on Known Instances

To answer the first question—*How does the selection method perform to compare a new configuration to the incumbent on the subset of instances for which we already collected information throughout the configuration run as seen in phase 1*?—we consider phase 1 (see Section 2.1). We populate $\mathcal{I}_{\text{known}}$ and Ω_{known} with instances and configurations, respectively, selected uniformly at random. We choose $\omega_{\text{inc}} \in \text{argmin}_{\omega \in \Omega_{\text{known}}} m(\omega, \mathcal{I}_{\text{known}})$ and train the random forest model on all the available data. Then, we pick configurations from $\Omega \backslash \Omega_{\text{known}}$ as ω_{ch} and run our iterative process; we refer to this as one run. We stop when we have run all instances of $\mathcal{I}_{\text{selected}}$. After each new instance is added, we report the percentage of time that has been spent up to that point to evaluate $m(\omega_{\text{ch}}, \mathcal{I}_{\text{selected}})$ compared to running it on all instances of $\mathcal{I}_{\text{known}}$, and we perform a *Wilcoxon matched-pairs signed-rank test* (Conover, 1998) with a significance level of 0.05 to decide if the challenger can be discarded. If $m(\omega_{\text{ch}}, \mathcal{I}_{\text{selected}}) > m(\omega_{\text{inc}}, \mathcal{I}_{\text{selected}})$ and the statistical test indicated statistical significance, ω_{ch} is discarded. We compare the resulting decision to the ground truth given by comparing $m(\omega_{\text{ch}}, \mathcal{I}_{\text{known}})$ to $m(\omega_{\text{inc}}, \mathcal{I}_{\text{known}})$ to assess the accuracy of the decision. For a given pair of $(\mathcal{I}_{\text{known}}, \Omega_{\text{known}})$, we performed 10 independent runs, using different pseudo-random number seeds, and report the average over those runs.

Figure 2 shows the collected accuracy over the time spent to make the comparison for two examples. Figure 2(a) is a case in which the discrimination and variance methods are significantly more accurate than the three others at any given time, while UDD and uncertainty show lower accuracy than random sampling. Figure 2(b) is a case in which discrimination and variance methods start with an advantage over random but quickly reach the same accuracy; once again, UDD and uncertainty perform substantially worse.

Figure 3 summarizes the previously described curves by computing the area under the curve (AUC) for all tested amounts of prior data; the higher the AUC, the faster and more accurately the decision can be taken. This visualization

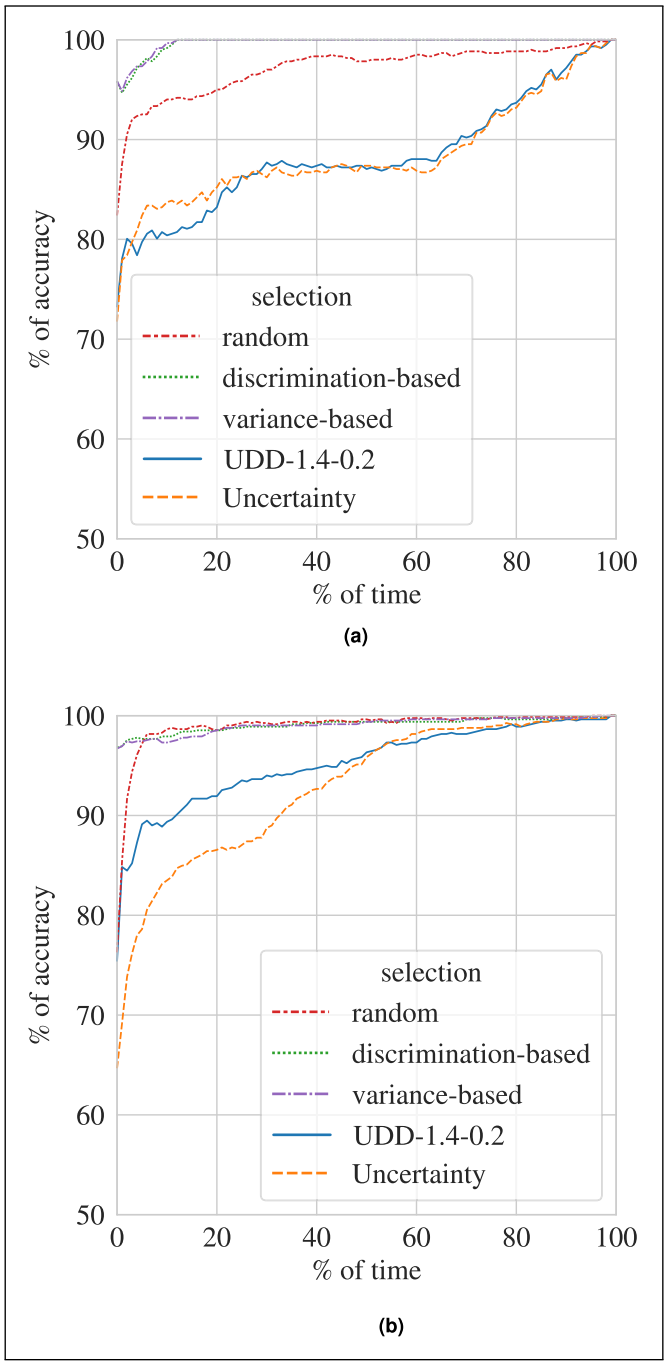

Figure 2. Mean accuracy of the Wilcoxon test ($p < 0.05$) on which among ω_{ch} and ω_{inc} performs best vs. the percentage of time spent on evaluations (100% means that all instances of $\mathcal{I}_{known}$ have been run). (a) Kissat on IBM, with an empirical prediction model trained on the performance of 50 configurations on 50% of the full instance set and (b) CPLEX on RCW2, with an empirical prediction model trained on the performance of 20 configurations on 10% of the full instance set.

allows us to examine how the methods compare and also illustrate the impact of the prior data used on the empirical performance model. In all our scenarios, we can see a clear correlation between the amount of configurations in Ω_{known} and the AUC. This would allow the selection method to become more and more efficient over the course of the configuration run and avoid wasting time in the final steps. On the other hand, adding more instances does not seem to consistently improve performance. This is in line with the expectation that our instance sets are built to be homogeneous; thus adding more instances will be unlikely to substantially improve the model.

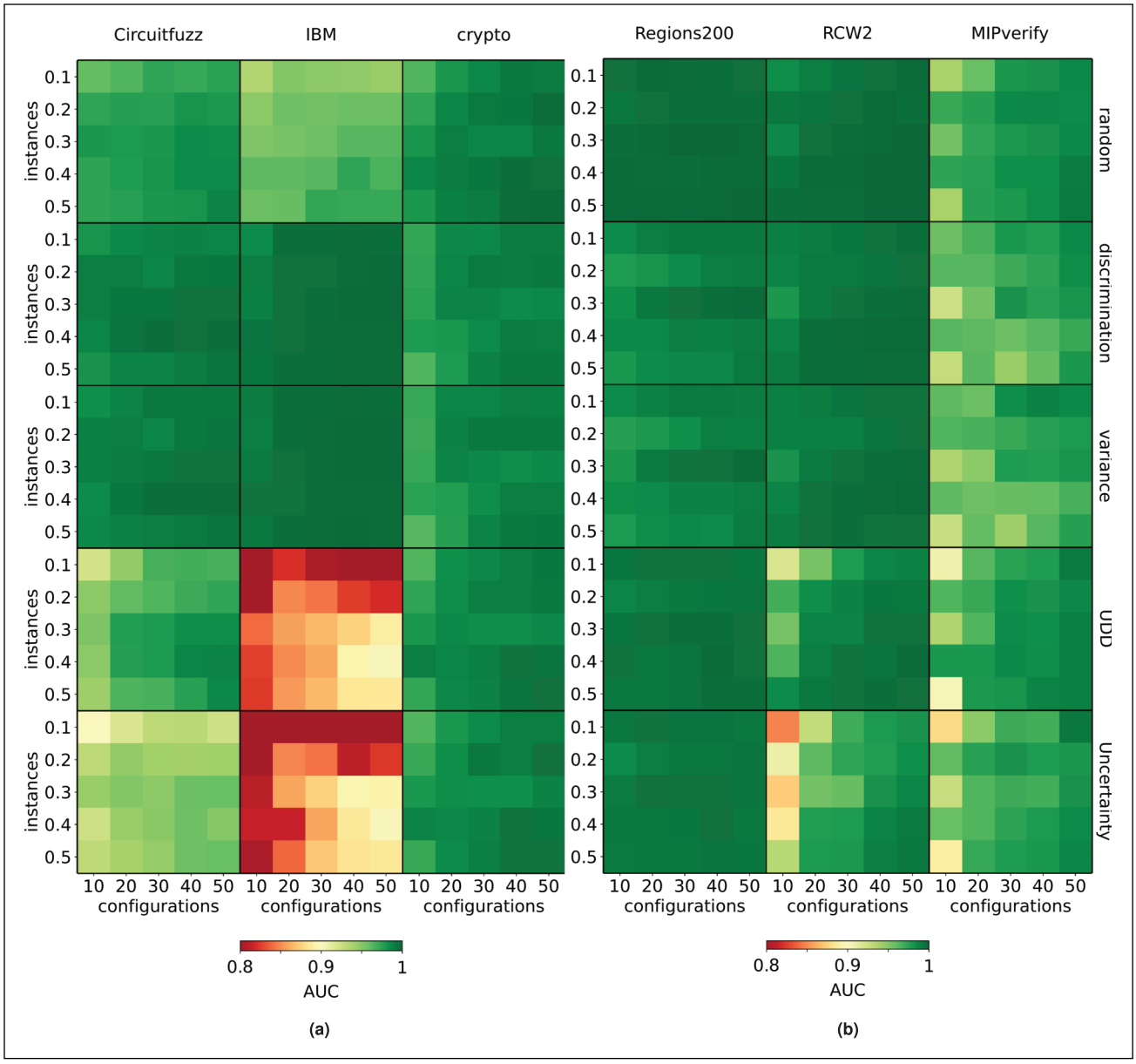

Figure 3. Area under the curve of the mean accuracy of the Wilcoxon test ($p < 0.05$) on which among ω_{ch} and ω_{inc} performs best against the time spent on evaluations. For each instance selection method, on each configuration scenario, we show results for a number of known configurations in [10, 20, 30, 40, 50] and a fraction of known instances in [0.1, 0.2, 0.3, 0.4, 0.5] of the full dataset. (a) Kissat scenarios and (b) CPLEX scenarios.

Regarding the selection methods, randomly sampling instances performs well, but in most cases, the discrimination and variance approaches are superior.

The results on the IBM dataset exacerbate the tendencies observed on others. Compared to datasets where little to no instances remain unsolved within the given cutoff time by a well-chosen configuration, timeouts occur on about a third of the IBM dataset. On the other hand, half of the instances are solvable within a few seconds. This large variation in running times means that selecting the wrong instance can have a dramatic effect on the overall running time and would explain why random sampling does not perform as well on this scenario as on the others. The fact that the UDD and Uncertainty scores do not account for the expected running time on an instance also penalizes them strongly.

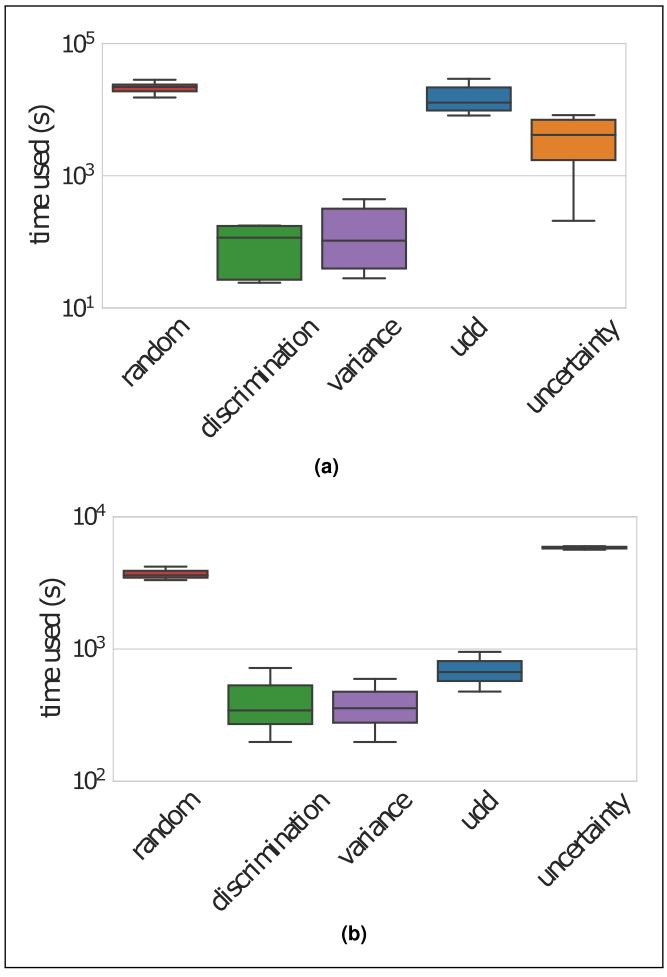

Figure 4. Time used (in s) before deciding that one configuration is better than the other based on a Wilcoxon test ($\alpha = 0.05$) or reaching a maximum of 10 instance selected. (a) Kissat on crypto, with an empirical prediction model trained on the performance of 20 configurations on 30% of the full instance set and (b) CPLEX on RCW2, with an empirical prediction model trained on the performance of 40 configurations on 40% of the full instance set.

4.2 Compare Configurations on Unknown Instances

To answer the second question—*How does the selection method perform to compare a new configuration to the incumbent on all instances, selecting instances for which we did not collect information throughout the configuration run as seen in phase 2*?—we consider phase 2 (see Section 2.1). We populate $\mathcal{I}_{\text{known}}$ and Ω_{known} with instances and configurations, respectively, selected uniformly at random. The random forest model is trained on all the performance data regarding all pairs of instances and configurations from those two sets. We choose $\omega_{\text{inc}} \in \text{argmin}_{\omega \in \Omega_{\text{known}}} m(\omega, \mathcal{I}_{\text{selected}})$ and collect all $\omega_{\text{ch}} \in \Omega \setminus \Omega_{\text{known}}$, such that the performance of ω_{inc} and ω_{ch} cannot be distinguished on the instances of $\mathcal{I}_{\text{known}}$ by a Wilcoxon test with a significance level of 0.05. We then apply our selection methods to select up to $n_{\max} = 10$ instances on which we run both configurations until they can be distinguished using the previous test.

For each method and each considered pair of $(\mathcal{I}_{\text{known}}, \Omega_{\text{known}})$, we gather the time used to decide between the two configurations at hand, that is, the sum of the running times of ω_{inc} and ω_{ch} on $\mathcal{I}_{\text{selected}}$. Figure 4 shows the running times obtained for two example scenarios.

To evaluate the performance of the selection methods, we computed the median time used to run the instances selected by each of the methods for each prior data and reported it in Table 2. The statistical significance of differences in the medians was tested with a permutation test (significance level of 0.05). In most cases, random is outperformed by all other methods, with some exceptions (uncertainty performs worse on RCW2, and random is best on MIPverify). The data shows discrimination and variance outperform the other methods in almost all cases, with variance providing a speedup ranging from a factor of 5.8 up to 3,000 compared to random. We note that the high speedups observed for the IBM dataset are

Table 2. Median Time in Seconds for Each Method Over Every Tested Prior Data, With Lowest Medians Boldfaced (Statistical Significance According to a Permutation Test With $\alpha = 0.05$).

	IBM	Kissat CF	Crypto	REG200	CPLEX RCW2	MIPverify
random	1,557	979.7	21,243	576.8	4,138	**29,470**
discrimination	0.086	143.6	419.3	**96.66**	364.7	44,390
variance	0.776	**95.16**	**372.2**	109.5	**342.0**	41,365
udd	880.9	393.2	13,483	379.7	1,299	**28,845**
uncertainty	**0.033**	330.8	2,361.9	152.7	5,974	39,801

linked to high variance in the running time distribution of the instances, which range from milliseconds to the timeout of 300 s.

4.3 Discussion

The results shown in this section indicate that the best-performing methods to discriminate between two configurations of the same algorithm within a limited amount of time are the ones based on the running time variance on each instance and on their discrimination power. We notice that both methods inspired by the active learning literature are not performing as well. While we wanted to assess these methods on our problem, this was to be expected, since they were designed with a different goal in mind. Indeed, the field of active learning focuses on improving the accuracy of the model, whereas we only use a model to avoid having to run each configuration on each instance. Improving the accuracy of this model can serve our purpose, but it is not our final goal.

We note that the experiments reported here made use of randomly chosen configurations of a given algorithm. As a result, the variation in running times between these configurations is much larger than that expected during an actual configuration run, which focuses on high-performance configurations. While this certainly does not invalidate our results, it implies that we should not expect speedups as large as the ones observed in Table 2 when including our methods inside a configurator.

5 Evaluation Inside a Configurator

As previously shown, applying instance selection and performing a statistical test allows us to spend less time on comparing the performance of two configurations through two expected behaviors: early stopping of the evaluation of less promising configurations and performance comparisons on less time-consuming instances. In this section, we include the instance selection mechanism inside a model-based configurator in order to evaluate if the previously observed results can be translated to the performance of the configurator itself. To do so, we expanded the prominent sequential model-based configurator SMAC. However, since SMAC does not include a statistical test, the two aspects of our methods have to be evaluated separately. First, we evaluate SMAC-IS (SMAC with Instance Selection), a version of SMAC in which we added at both phases of the both parts of the instance selection method, namely a *Wilcoxon matched-pairs signed-rank test* (Conover, 1998) with a significance level $\alpha = 0.05$ to decide if the challenger configuration should be dropped earlier, and an instance selection method to decide on which instance the next run should happen. To compare the performance of SMAC to SMAC-IS, we followed the procedure described in Section 3 and obtained for each scenario and configurator a distribution of best configurations. The following results are based on those distributions.

5.1 Impact of the Instance Selection Methods

To answer our first question, we implemented the instance selection mechanisms inside SMAC at the two phases identified earlier and named this new version SMAC-IS. Table 3 shows the median performance values of the best configurations distribution. We validate the statistical significance of the differences with a Mann–Whitney U test with a significance level $\alpha = 0.05$. Moreover, we show in bold methods that perform better than SMAC, our baseline.

Compared to vanilla SMAC, SMAC-IS improved three of our five scenarios. In particular, the EAX on rue-1000-3000 scenario (Table 3(a)), which showed an improvement by simply adding the statistical test, displays even further improvement with most instance selection methods; at best, from a default performance of 120.82 s, SMAC-IS reaches a median of 65.68 s, while SMAC could only reach 92.93 s. A similarly impressive improvement was achieved for CPLEX on RCW2 (Table 3(d)) on which, from a default value of 115.95 s, SMAC-IS reaches a median of 57.63 s, while SMAC could only reach 83.96 s.

Table 3. Median Performance of SMAC-IS With the Selection Methods Random (Rand), Variance-Based (Var), and Discrimination-Based (Disc) at Both Phases.

		Phase 2		
		Rand	Var	Disc
(a) EAX rue-1000-3000				
phase 1	rand	**89.87**	**71.87**	**72.72**
	var	121.69	**87.14**	95.55
	disc	**89.80**	**87.34**	**65.68**
(b) LKH rue-1000-3000				
phase 1	rand	233.13	228.74	229.48
	var	229.39	229.00	243.04
	disc	228.76	185.62	229.19
(c) CPLEX CLS				
phase 1	rand	1.79	1.73	1.67
	var	1.61	1.66	1.68
	disc	1.66	1.69	1.65
(d) CPLEX RCW2				
phase 1	rand	113.73	113.54	113.94
	var	**57.63**	113.98	114.27
	disc	86.06	114.86	114.48
(e) CPLEX regions 200				
phase 1	rand	3.68	2.95	3.35
	var	3.25	3.77	3.74
	disc	**2.79**	**2.78**	3.05

Note. Boldfaced values are better than those for vanilla SMAC. The lowest median is underlined. All underlined medians are significantly different from others based on a Mann–Whitney *U* test ($\alpha = 0.05$). SMAC-IS = SMAC-Instance Selection.

For CPLEX on REG200 (Table 3(e)), SMAC-IS improves slightly over SMAC, but for CPLEX on CLS (Table 3(c)) it does not, despite being able to find a better configuration than the default. At the other end of the spectrum, for LKH on rue-1000-3000 (Table 3(b)) SMAC-IS returns configurations that perform even worse than the default values in half of the cases.

5.2 Impact of the Statistical Test

To evaluate the impact of the statistical test, we examined the performance of SMAC-IS with random sampling at both phases, which corresponds to vanilla SMAC with a Wilcoxon test to discriminate between the performance of the incumbent and one of the challenger configurations at both phases of the configuration. We dub this variant SMAC-W (SMAC with Wilcoxon test). We show the median of those distributions in Table 4(a). Similarly to the previous results, we validated the statistical significance of the differences using a Mann–Whitney *U* test (with $\alpha = 0.05$) and detected statistical significance for all observed differences.

In all except one scenario, the use of a statistical test for early stopping of the comparison has a negative effect. To further investigate these results, we looked at how many times the challenger replaces the incumbent and on how many instances the configurations are evaluated. These results are shown in Table 4(b). We note that the number of accepted incumbents during a run is significantly lower when using the test. Vanilla SMAC accepts the incumbent up to twice as often than SMAC-W for CPLEX on CLS. Moreover, since incumbents get rejected more quickly, the number of instances on which the configurations are evaluated does not increase as quickly in SMAC-W as in SMAC. We can expect that running on a smaller number of instances prevents the configurator from seeing the full range of instances on which the algorithm should perform well, leading to overfitting. This is especially evident for the LKH scenario, on which the expected performance of SMAC-W is worse than the default on the test set. We also noticed that the only case in which the

Table 4. Comparison of SMAC and SMAC-W, Respectively Without and With a Wilcoxon Test to Decide Whether a Challenger Configuration Should be Kept Longer.

(a) Median PAR10 of the best found configurations. The lowest medians are underlined, all are statistically significantly lower according to a Mann–Whitney *U* test (with $\alpha = 0.05$)

Scenario		Default	SMAC	SMAC-W
CPLEX	CLS	1.72	1.31	1.79
	RCW2	115.97	83.96	113.73
	REG200	6.13	2.84	3.68
EAX	rue-1000-3000	120.82	92.93	89.87
LKH		229.22	157.83	233.13

(b) Mean number of changes in incumbent and number of instances in $\mathcal{I}_{\text{known}}$ at the end of the configuration procedure

		Changes		Instances	
Scenario		SMAC-W	SMAC	SMAC-W	SMAC
CPLEX	CLS	3.0	7.1	50	50
	RCW2	3.1	4.2	495	495
	REG200	4.5	5.6	816	823
EAX	rue-1000-3000	4.9	7.6	332	294
LKH		3.1	3.5	432	685

number of instances seen during configuration is higher for SMAC-W corresponds to the only scenario in which SMAC-W performs better than SMAC. Based on those results, we can answer our research question and state that in most cases, adding a statistical test to SMAC hinders its performance.

5.3 Discussion

Since the instance selection mechanism did not allow us to improve over SMAC on all scenarios, we looked into the characteristics of each scenario to better understand what might allow instance selection to reach its full potential.

When we look at each selection phase separately, there is no clear trend in terms of which method performs best at any of those. One expectation was that for scenarios with a low running time, the overhead induced by our methods would hinder the process, but the instance selection performed slightly better than SMAC on one out of our two scenarios with short running times (CPLEX on CLS and REG200), so this hypothesis does not hold in our experiments. Another expectation was that the homogeneity of the dataset would strongly impact the ability to select the right instances and to accurately decide which configurations to drop. However, the best and worst outcomes were obtained on the same dataset, rue-1000-3000, on which we found nine clusters of instances when applying a simple mean shift algorithm, which is the highest number among our datasets. Moreover, two seemingly homogeneous datasets, namely CLS and REG200, show very different outcomes. However, the number of clusters does not capture how far those clusters are from each other, which would impact the difficulty to select representative instances.

Thus, based on our results, we do not see a clear trend regarding what kinds of scenarios would benefit (or not) from our instance selection mechanism. We note, however, that in two out of five scenarios, we were able to nearly double the improvement obtained by SMAC. This improvement demonstrates that in some scenarios, selecting the instances on which to run the configurations at hand can significantly improve the performance of a general-purpose algorithm configurator.

6 Conclusion

Inspired by the success of instance selection when comparing algorithms (Matricon et al., 2021), we adapted four methods from several fields (Gu et al., 2014; Matricon et al., 2021) that could be applied to select instances in the context of AAC. We identified two steps of AAC procedures at which the selection mechanism could be applied and designed two sets of experiments to assess the performance gains thus obtainable. In the first, we consider a situation in which the performance of an incumbent configuration on a set of instances is known and we want to determine whether the challenger configuration, whose performance is unknown, performs better on this set. In the second, two similarly performing configurations

have to be evaluated on unknown instances. Our results show that in both cases, there is considerable potential in the use of those methods, in particular the ones based on the variability in running time or on discrimination power.

Based on those encouraging results, we included the two best selection mechanisms identified in the first phase of our study at both identified steps of the configuration process within the prominent and state-of-the-art SMAC3 configuration system. On half of the considered scenarios, selecting on which instances to run the first and second phases, on top of performing a Wilcoxon test to decide when to stop the comparison between the current incumbent and a challenger configuration, makes it possible to achieve better performing configurations within the same configuration budget, sometimes reaching major improvement compared to SMAC. However, we not yet found a clear way to decide which instance selection method to apply or which scenarios have the potential to benefit from them. Moreover, we studied the impact of solely adding the Wilcoxon test and found that, in most scenarios, using the test degrades the configuration process of SMAC. We note that on the scenarios we have studied, use of the test lowers the number of accepted challengers, likely discarding well-performing configurations by mistake, and tends to slow down the addition of new instances to the pool of instances on which configurations are evaluated. This second point could potentially lead to a form of over-fitting. Those observations confirm that the selection mechanism is the one leading to the observed improvements.

This work opens the door to a more principled way of deciding on which instances the configurations should be evaluated. While more research is needed to decide which specific method to apply in practice, selecting instances during AAC shows great potential.

ORCID iDs

Marie Anastacio https://orcid.org/0000-0002-4039-2470
Théo Matricon https://orcid.org/0000-0002-5043-3221
Holger H. Hoos https://orcid.org/0000-0003-0629-0099

Author Contributions

Marie Anastacio was leading the project. She wrote code, planned and ran experiments, analysed the results and wrote the paper. Théo Matricon wrote a large part of the code, participated in designing the experiments and analysing the results, and assisted with writing the paper. Holger H. Hoos supervised the project, advised on the design of experiments and methodology, participated in analysing the results and with writing the paper.

Funding

The authors acknowledge support through an Alexander von Humboldt Professorship in Artificial Intelligence held by Holger H. Hoos.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Note

1. Implementation of the first set of experiments: `github.com/Theomat/MPSEAS` and of the second set of experiments: `github.com/ADA-research/SMACIS`

References

Anastacio, M., & Hoos, H. H. (2020). Model-based algorithm configuration with default-guided probabilistic sampling. In *Proceedings of Parallel Problem Solving From Nature—PPSN XVI, Part I* (Lecture Notes in Computer Science, Vol. 12269, pp. 95–110). https://doi.org/10.1007/978-3-030-58112-1_7

Ayerdi, B., & Graña, M. (2015). Random forest active learning for retinal image segmentation. In *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015* (Advances in Intelligent Systems and Computing, Vol. 403, pp. 213–221). https://doi.org/10.1007/978-3-319-26227-7_20

Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., & Suda, M. (2020). *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*. Department of Computer Science Report Series B. University of Helsinki.

Bhosle, N. P., & Kokare, M. (2020). Random forest-based active learning for content-based image retrieval. *International Journal of Intelligent Information and Database Systems*, *13*(1), 72–88. https://doi.org/10.1504/IJIIDS.2020.108223

Biere, A., Fazekas, K., Fleury, M., & Heisinger, M. (2020). CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020. In *Proceedings of SAT Competition 2020—Solver and Benchmark Descriptions* (Department of Computer Science Report Series B, Vol. B-2020-1, pp. 51–53). University of Helsinki.

Cáceres, L. P., López-Ibánez, M., Hoos, H., & Stützle, T. (2017). An experimental study of adaptive capping in irace. In *Proceedings of the 11th International Conference on Learning and Intelligent Optimization (LION 11)* (Lecture Notes in Computer Science, Vol. 10556, pp. 235–250). https://doi.org/10.1007/978-3-319-69404-7_17

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence TPAMI*, *24*(5), 603–619. https://doi.org/10.1109/34.1000236

Conover, W. (1998). *Practical nonparametric statistics* (Wiley Series in Probability and Statistics, Vol. 350).

Domhan, T., Springenberg, J. T., & Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015* (pp. 3460–3468). https://ijcai.org/Abstract/15/487.

Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, *56*(293), 52–64. https://doi.org/10.1080/01621459.1961.10482090

Falkner, S., Lindauer, M., & Hutter, F. (2015). SpySMAC: Automated configuration and performance analysis of SAT solvers. In *Proceedings of Theory and Applications of Satisfiability Testing—SAT 2015* (Lecture Notes in Computer Science, Vol. 9340, pp. 215–222). https://doi.org/10.1007/978-3-319-24318-4_16

Gent, I. P., Hussain, B. S., Jefferson, C., Kotthoff, L., Miguel, I., Nightingale, G. F., & Nightingale, P. (2014). Discriminating instance generation for automated constraint model selection. In B. O'Sullivan (Ed.) *Proceedings of Principles and Practice of Constraint Programming—20th International Conference, CP 2014* (Lecture Notes in Computer Science, Vol. 8656, pp. 356–365). https://doi.org/10.1007/978-3-319-10428-7_27

Gu, Y., Zydek, D., & Jin, Z. (2014). Active learning based on random forest and its application to terrain classification. In *Progress in Systems Engineering—Proceedings of the Twenty-Third International Conference on Systems Engineering*, *ICSEng 2014* (Advances in Intelligent Systems and Computing, Vol. 366, pp. 273–278). https://doi.org/10.1007/978-3-319-08422-0_41

Hoos, H. H. (2012). Automated algorithm configuration and parameter tuning. In *Autonomous search* (pp. 37–71). https://doi.org/10.1007/978-3-642-21434-9_3.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2011a). Bayesian optimization with censored response data. In *Workshop on Bayesian optimization, sequential experimental design, and bandits, in conjunction with NIPS*.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In *Proceedings of Integration of AI and or Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010* (Lecture Notes in Computer Science, Vol. 6140, pp. 186–202). https://doi.org/10.1007/978-3-642-13520-0_23

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011b). Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello (Ed.), *Learning and Intelligent Optimization—5th International Conference, LION 5, Selected Papers* (Lecture Notes in Computer Science, Vol. 6683, pp. 507–523). https://doi.org/10.1007/978-3-642-25566-3_40

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research JAIR*, *36*, 267–306. https://doi.org/10.1613/jair.2861

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2014). AClib: A benchmark library for algorithm configuration. In *Learning and Intelligent Optimization—8th International Conference, Lion 8, Revised Selected Papers* (Lecture Notes in Computer Science, Vol. 8426, pp. 36–40). https://doi.org/10.1007/978-3-319-09584-4_4

Hutter, F., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, *206*, 79–111. https://doi.org/10.1016/j.artint.2013.10.003

König, M., Hoos, H. H., & van Rijn, J. N. (2021). Speeding up neural network verification via automated algorithm configuration. In *Workshop on security and safety in machine learning systems, in conjunction with to the international conference on learning representations ICLR*.

Luo, C., Hoos, H. H., Cai, S., Lin, Q., Zhang, H., & Zhang, D. (2019). Local search with efficient automatic configuration for minimum vertex cover. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI 2019)* (pp. 1297–1304). https://doi.org/10.24963/ijcai.2019/180

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58. https://doi.org/10.1016/j.orp.2016.09.002

Matricon, T., Anastacio, M., Fijalkow, N., Simon, L., & Hoos, H. H. (2021). Statistical comparison of algorithm performance through instance selection. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, (LIPIcs, Vol. 210, pp. 43:1–43:21). Dagstuhl: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.CP.2021.43

Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., & Neumann, F. (2013). A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence*, *69*(2), 151–182. https://doi.org/10.1007/s10472-013-9341-2

Nejati, S., & Ganesh, V. (2019). CDCL(Crypto) SAT solvers for cryptanalysis. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, CASCON 2019* (pp. 311–316). https://dl.acm.org/doi/10.5555/3370272.3370307

Pihera, J., & Musliu, N. (2014). Application of machine learning to algorithm selection for TSP. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence* (pp. 47–54). https://doi.org/10.1109/ICTAI.2014.18

Pushak, Y., & Hoos, H. H. (2020). Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In *GECCO '20: Genetic and Evolutionary Computation Conference 2020* (pp. 245–253). https://doi.org/10.1145/3377930.3390211

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research JAIR*, *32*, 565–606. https://doi.org/10.1613/jair.2490

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proceedings of the Knowledge Representation and Automated Reasoning RCRA Workshop, in Conjunction with the International Joint Conference on Artificial Intelligence*, *IJCAI 2011* (pp. 16–30).