Proceedings of Machine Learning Research 191:11–23, 2022 ECMLPKDD Workshop on Meta-Knowledge Transfer

Instance selection for configuration performance comparison

Marie Anastacio
Leiden Institute of Advances Computer Science, Leiden University, The Netherlands

Théo Matricon
Univ. Bordeaux, CNRS, LaBRI, UMR 5800, F-33400, Talence, France

Holger H. Hoos
RWTH Aachen University, Aachen, Germany
Leiden Institute of Advances Computer Science, Leiden University, The Netherlands
University of British Columbia, Vancouver, Canada

Editors: P. Brazdil, J. N. van Rijn, H. Gouk and F. Mohr

Abstract

Comparing the performance of two configurations of a given algorithm plays a critical role in algorithm configuration and performance optimisation, be it automated or manual, and requires substantial computational resources. Time is often wasted on less promising configurations but also on instances that require a long running time regardless of the configuration. Prior work has shown that by running an algorithm on carefully selected instances, the time required to accurately decide the better of two given algorithms can be significantly reduced. In this work, we explore ways to apply a similar selection process to compare two configurations of the same algorithm. We adapted four selection methods from the literature to work with the performance model used in model-based configurators and evaluated them on six benchmarks. Our experimental evaluation shows that, depending on the problem instances and their running time distribution, a decision can be reached 5 to 3000 times faster than with random sampling, the method used in current state-of-the-art configurators.

Keywords: instance selection, running time optimisation, algorithm configuration

1. Introduction

The automatic configuration of algorithms is an active research topic that produced impressive results and showed great success in terms of performance improvements in solvers for prominent and challenging AI problems, such as Boolean Satisfiability (SAT) (Xu et al., 2008; Falkner et al., 2015) or Mixed-integer programming (MIP) (Xu et al., 2011; Hutter et al., 2010). Applications to machine learning have enabled major progress in the area of automated machine learning (AutoML) (Hutter et al., 2019). However, as there is an increasing focus on sustainability, the computational resources and the environmental impact associated with the use of AI methods should be put under scrutiny, providing additional incentives to configure algorithms but also to reduce the computational cost of automated configuration.

In particular, the most expensive part of configuration is to run the target algorithm with various parameter values on different problem instances to evaluate performance and to determine which parameter settings achieve the best performance. For anytime algorithms, such as machine learning methods, there has been work on early stopping less promising runs

based on the learning curve – *e.g.*, Domhan et al. (2015); Luo et al. (2019), while adaptive capping such as the one included in irace (López-Ibáñez et al., 2016) allows to early stop the evaluation when a configuration is already deemed to not be competitive with the current incumbent. Those lines of research are focused on the idea of discarding configurations that are not sufficiently promising.

In this work, we focus on the instance space and on techniques for identifying instances that help discriminate between the compared configurations. Building on previous work by Matricon et al. (2021) that compared the performance of algorithms, we argue that carefully selecting instances and avoiding long evaluations that provide only a limited amount of information makes it possible to reach a decision faster. Despite similarities with active learning, this problem has a different objective, and it is thus not possible to apply existing methods directly. With four methods adapted from the literature, we evaluate the potential of an instance selection mechanism to compare configurations of a single algorithm. Our experiments shows a speed-up of 5 up to 3000 times to take decisions depending on the instances and their running time distribution.

We first introduce the algorithm configuration problem (Section 2), followed by the methods and how we adapted them in Section 3. Section 4 presents the experiments we conducted, Section 5 describes their results and we conclude in Section 6.

2. Background

This work aims at smartly selecting instances when comparing the performance of different configurations of a given algorithm. Let us first introduce the algorithm configuration problem and existing methods related to the selection of instances.

2.1. Algorithm configuration

The algorithm configuration problem is defined as follows (see, *e.g.*, Hoos (2012)): Given a target algorithm A; a configuration space $\mathcal{C}$ containing all valid combinations of parameter values for A; a set of problem instances $\mathcal{I}$; and a performance metric m that measures the performance of the target algorithm A on an instance of $\mathcal{I}$ following a configuration in $\mathcal{C}$; find $c^* \in \mathcal{C}$ that optimises the performance of A on $\mathcal{I}$ according to m. Each parameter has a domain of possible values that can be of different types: categorical, ordinal or numerical. Categorical parameters have an unordered finite set of possible values and are often used to select between several heuristic components or mechanisms. Ordinal parameters have an ordered finite set of possible values. Numerical parameters are real- or integer-valued and are often used to calibrate heuristic mechanisms or components. Parameters can also conditionally depend on each other, so that one is active only when another takes a specific value; as a simple example, consider a Boolean parameter that activates a mechanism, whose behaviour is then adjusted using a numerical parameter.

Methods to tackle this problem are called *configurators* and typically include two key components: one generates configurations from $\mathcal{C}$ to be evaluated and the other compares those configurations based on the metric m. Because the most time-consuming part of this process is typically the performance evaluation of the target algorithm, much work has been focusing on ways to generate a good configuration, *e.g.*, using a surrogate performance

model (Hutter et al., 2011; Ansótegui et al., 2015), a model of known good configurations (López-Ibáñez et al., 2016; Anastacio and Hoos, 2020) or pruning methods (Pushak and Hoos, 2020). Rather than looking at the generation of new configurations, here, we consider the choice of instances on which the algorithm is run, with the goal of lowering the amount of time spent on solving these. In this work, we limit our scope to a model-based approach to support our selection methods.

2.2. Instance selection

Choosing the most relevant instance among a set is not a new problem and has been tackled by active learning (Sun and Wang, 2010) and in the context of comparing the performance of algorithms on a given dataset.

Performance comparison. We build on the work of Matricon et al. (2021), which compares the running time of two algorithms on a given set of instances. Similarly, we aim to decide which is the fastest among two challengers – in their case, two algorithms and in ours two configuration of a single algorithm. This is done by selecting the instances providing the most information per expected unit of time spent running the algorithm. However, they based their selection heuristics on a dataset of algorithms runs and distribution assumptions that can not be used in a configurator. A configurator also needs to select among instances for which it has no performance information yet. Thus, we adapt their methods to rely on a performance model, such as those already used in several configurators (Hutter et al., 2011; Ansótegui et al., 2015).

Active learning. The active learning problem seeks to choose an instance among a set on which the model should be trained on next. The idea is that a relevant instance should have a high impact on the model (e.g., increasing its accuracy or reducing its variance). It is closely related to our problem, but differs as the chosen instance should also lead to low running times, which is not a common objective in active learning. Considering work applicable to a random forest (RF) regression model, we chose to adapt the work of Gu et al. (2015) which considers active learning for terrain classification using random forests. Other works (*e.g.*, Bhosle and Kokare (2020); Ayerdi and Graña (2016)) used similar ideas, focusing on the uncertainty of the model.

3. Instance selection for algorithm configuration

The per-set efficient algorithm selection problem (PSEAS) as defined in the work of Matricon et al. (2021) appears during algorithm configuration in a slightly different form. Rather than selecting an algorithm, the configurator needs to select a specific configuration of an algorithm among other configurations. To allow the selection of an instance, we need prior information. In a configurator, this information comes in the form of prior runs of other configurations on the instances and instance features. The latter is used in particular by model-based configurators.

3.1. Instance selection

Following the definition of automated algorithm configuration in Section 2.1, $\mathcal{I}$ is the finite set of instances and $\mathcal{C}$ is the set of valid configurations of the algorithm at hand. At a given step, we have partial running time information on $\mathcal{I}_{known} \subseteq \mathcal{I}$ for configurations in $\mathcal{C}_{known} \subseteq \mathcal{C}$, which means that for $C \in \mathcal{C}_{known}$, there exists information about the performance of C on at least one instance of $\mathcal{I}_{known}$.

When comparing a challenger configuration C_{ch} to the current best configuration C_{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 10 (coloured purple), but the same mechanism arises in any configurator. The first of these, which we name phase 1, corresponds to the problem studied in Matricon et al. (2021), where we already know the performance of C_{inc} on a set of instances $\mathcal{I}_{known}$ and want to determine whether C_{ch} performs better on this set. The second, which we name phase 2, corresponds to a case where we know the performance of both C_{inc} and C_{ch} on $\mathcal{I}_{known}$, but do not have sufficient information to decide which one is the best and thus want to evaluate both configurations on additional instances from $\mathcal{I} \setminus \mathcal{I}_{known}$.

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

```
if Not enough runs for configuration C_inc then
 |  Execute a run of C_inc on an instance not run sampled uniformly at random
end
N ← 1
while there are instances on which C_inc was run but not C_ch do
 |  Run C_ch on a subset of N instances on which C_inc was run but not C_ch
 |  if challenger is worse then
 |   |  return C_inc
 |  else
 |   |  multiply N by 2
 |  end
end
N_run ← 0
while C_inc and C_ch cannot be distinguished do
 |  if N_run < n_max then
 |   |  Run C_inc and C_ch on an instance on which none were run before
 |   |  N_run = N_run + 1
 |  else
 |   |  return C_inc
 |  end
end
return Best between C_inc and C_ch
```

In both cases, we seek to choose an instance $I \in \mathcal{I}_{choose} \subseteq \mathcal{I}$ and gather performance information on it iteratively until we satisfy a stopping condition.

In phase 1, $\mathcal{I}_{known}$ is the subset of instances on which we have run C_{inc} so far, and C_{inc} is the best performing configuration known to us on $\mathcal{I}_{known}$. At each step, we will select an instance on which to run C_{ch} and add it to the set of instances on which C_{ch}

has been run, noted $\mathcal{I}_{selected} \subseteq \mathcal{I}_{known}$. At any step, the set we can select instances from is $\mathcal{I}_{choose} = \mathcal{I}_{known} \setminus \mathcal{I}_{selected}$. During this phase, we want to discard C_{ch}, given sufficient evidence that it performs worse than C_{inc}, but not the other way around. Thus, our stopping criteria are to be confident that C_{ch} is worse than C_{inc} or to have $\mathcal{I}_{choose} = \emptyset$. We consider that to select instances we have access to a prediction model trained on the valid pairs of $\mathcal{I}_{known}$, $\mathcal{C}_{known}$, which enables the prediction of running times on unknown pairs of $\mathcal{I}$, $\mathcal{C}$.

In phase 2, we also have a subset $\mathcal{I}_{known} \subset \mathcal{I}$. The goal is to be able to decide which is better between C_{inc} and C_{ch}, whose performance on $\mathcal{I}_{known}$ cannot be distinguished reliably; to achieve this we can select instances from $\mathcal{I}_{choose} = \mathcal{I} \setminus \mathcal{I}_{known}$ and iteratively add them to $\mathcal{I}_{known}$. Unlike in phase 1, there is no asymmetry between C_{inc} and C_{ch}. Both can be discarded given enough evidence. Since no configurations has been run on any of the instances in $\mathcal{I}_{choose}$, we predict the performance of C_{inc} and C_{ch} with a predictive model trained on the performance of the configurations from $\mathcal{C}_{known}$ on the instances from $\mathcal{I}_{known}$. To do so, we require instance features, as defined in previous work for a broad range of problems – *e.g.* SAT (Xu et al., 2008), MIP (Xu et al., 2011)). We stop when we can clearly separate the performance of C_{inc} and C_{ch} on $\mathcal{I}_{known}$, or when we have added $n_{max} \in \mathbb{N}$ instances in total during the process.

3.2. Methods

Following Matricon et al. (2021), we assign scores to instances and choose iteratively an instance $I^* \in \text{argmax}_{I \in \mathcal{I}_{choose}} score(I)$ with the highest score. The intuition is that the score should reflect the relevance of choosing that instance both in terms of information obtained and cost incurred. We adapted two of their methods 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 their information-based method, as it relies on assumptions regarding the performance distribution that could not be made in our context.

3.2.1. Baseline: Uniform random sampling

This is equivalent to assigning every instance the same score, and thus sampling an instance uniformly at random.

3.2.2. Discrimination.

This method, originally inspired by the work of Gent et al. (2014), tries to choose the instance that most discriminate between the best and other configurations. Let $\rho > 1$; we say that a configuration C is *ρ-dominated* on an instance I if there exists another configuration C' such that $m(C', I) \leq \rho \cdot m(C, I)$. 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 divided by the mean running time of the instance. The score is directly $score(I) = Q(I)$.

3.2.3. Variance

The intuition is that an instance with high variance is likely to discriminate between two configurations. But we must also take into account the cost of running this instance, this

is why we divide the variance by the mean running time of the instance. Our score is thus $score(I) = \frac{Var(I)}{Mean(I)}$.

3.2.4. Uncertainty-Diversity-Density (UDD)

This method is inspired by the work of Gu et al. (2015) 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 it UDD because it is a combination of three scores: uncertainty, diversity and density. Thus we can write the final score as $score(I) = Uncertainty(I) + \alpha Diversity(I) + \beta Density(I)$, all three scores are scaled and translated to be in in $[0; 1]$ before computing $score(I)$.

- $Uncertainty(I)$ is the random forest's variance on running time prediction for instance I.
- $Diversity(I) = -min_{I' \in \mathcal{I}_{known}} \mathcal{D}(I, I')$, where $\mathcal{D}$ is a distance function over instances. Intuitively, the closer we are to instances in $\mathcal{I}_{known}$ the more likely that this instance provides little to no additional information.
- $Density(I) = \frac{1}{k} \sum_{I' \in \mathcal{N}_k(I,\mathcal{D})} \mathcal{D}(I, I')^2$ where $k \in \mathbb{N}$ is a parameter, $\mathcal{D}$ is a distance function over instances and $\mathcal{N}_k(I, \mathcal{D}, \mathcal{I}_{choose})$ returns the k closest neighbours of I in $\mathcal{I}_{choose} \setminus \{I\}$ according to $\mathcal{D}$. Intuitively, if an instance I has a lot of close instances then this is a relevant instance since running I should provide information about these other instances.

3.2.5. Uncertainty.

It is UDD with $\alpha = \beta = 0$, which is reminiscent of the variance method but for a model prediction.

4. Empirical evaluation

We designed and conducted experiments to answer the following questions:
Q1 - 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?
Q2 - 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?

4.1. Datasets

We used 6 configuration scenarios either from the Algorithm configuration library AClib (Hutter et al., 2014) or built based on it. Half of those are Boolean satisfiability (SAT) datasets and the other half are mixed integer programming (MIP) dataset.

For SAT, we used two datasets from AClib (Circuitfuzz, IBM) and generated a new set of instances based on the work of Nejati and Ganesh (2019). The later represents problems from cryptography as SAT instances (using the sha256 encoding, from 16 to 60 rounds, and an input size varying from 2^1 to 2^10). Based on the results of the SAT competition 2020, we decided to configure the current best SAT solver Kissat (Balyo et al., 2020) as it is highly configurable and similar to CadiCal (Biere et al., 2020) which is known to be configurable (Pushak and Hoos, 2020).

For MIP, we use 2 datasets from AClib (RCW2, Regions200) and added a more difficult dataset based on the work of König et al. (2021) which represents neural network verification problems. We use cplex as it is well known in the litterature.

4.2. Implementations details

Our implementation is available on GitHub (see supplementary materials).

The UDD method requires a distance function in the instance space, we use the same procedure as in 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 them with a simple grid search on a separate scenario (Kissat solver with the SWGCP dataset from AClib). For discrimination, we evaluated values in $[1.01; 2]$ with a step of 0.11 and found that $\rho = 1.12$ performed well on both levels. For UDD, we evaluated values in $[0; 2]$ with a step of 0.21 for both values independently and found that $\alpha = 0.2$ and $\beta = 1.4$ performed well on both levels.

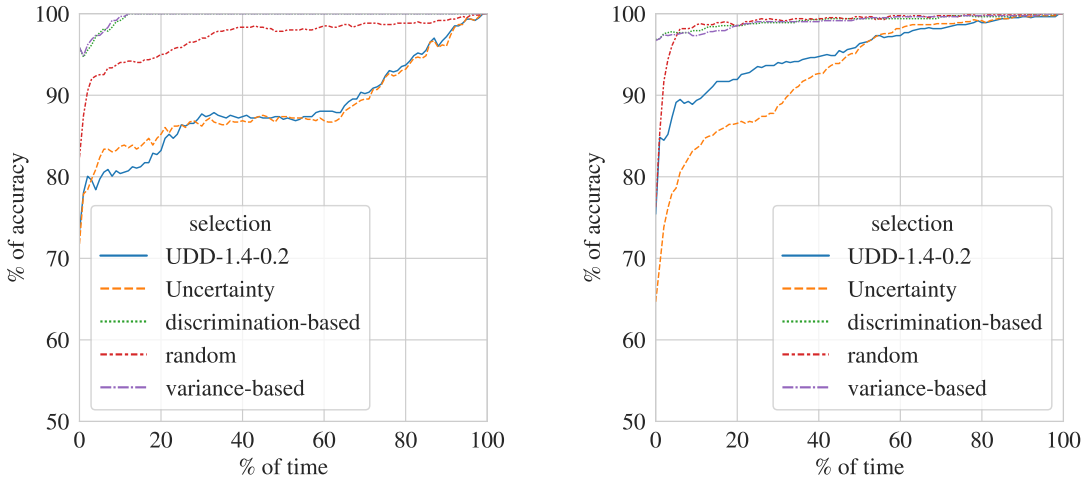
We generated 100 random configurations for each solver and ran them on all instances of their respective datasets. This allowed us to collect the performance of each pair of instance and configuration. We used the same Random Forest Model as in SMAC (Hutter et al., 2011) as a performance prediction model. We train it on a prior data consisting of the performances of every pair or known configuration and instances. Our experiments are run with various amounts of prior data: the number of known configurations is in $[10, 20, 30, 40, 50]$ and the amount of known instances is a portion containing $[0.1, 0.2, 0.3, 0.4, 0.5]$ of the full dataset. This allows us to evaluate how efficient the methods will be along a configuration run.

5. Results

To evaluate the performance of the selection methods in the two phases and to answer our research questions, we designed two sets of experiments. We show aggregated results here but the raw results and scripts to generate more visualisations are available on our git.

5.1. Compare configurations on known instances

To answer the first question, we place ourselves in phase 1 (see Section 3.1). We randomly select a fraction $p_{\mathcal{I}}$ of instances as $\mathcal{I}_{known}$ and a fraction $p_{\mathcal{C}}$ of configurations as $\mathcal{C}_{known}$. We choose $C_{inc} \in \text{argmax}_{C \in \mathcal{C}_{known}} m(C, \mathcal{I}_{known})$ and train the random forest model on all the available data. Then we pick configurations from $\mathcal{C} \setminus \mathcal{C}_{known}$ as C_{ch} and run our iterative process, this is a run. We stop when we have run all instances of $\mathcal{I}_{selected}$. After each new instance is added, we report the percentage of time that has been spent until now to



(*a*) Kissat on IBM
50% instances, 50 configurations

(*b*) Cplex on RCW2
10% instances, 20 configurations

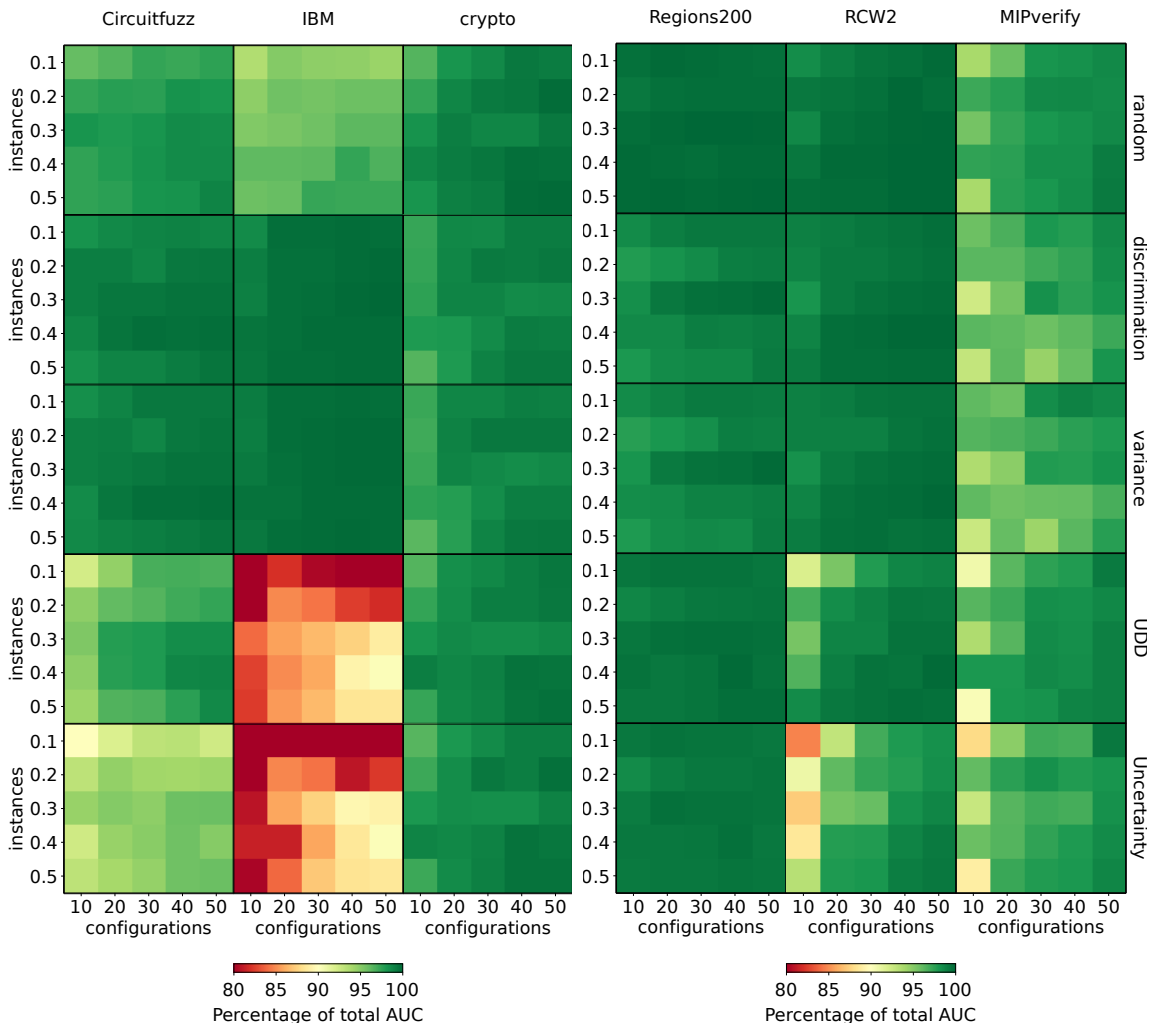
Figure 1: Mean accuracy of the Wilcoxon test (p=0.05) on which among C_{ch} and C_{inc} performs best along the percentage of time spent on evaluations (100% means that all instances of $\mathcal{I}_{known}$ have been run)

evaluate $m(C_{ch}, \mathcal{I}_{selected})$ compared to running it on all instances of $\mathcal{I}_{known}$ and perform a *Wilcoxon matched-pairs signed-ranks test* (Conover, 1998) with a p-value of 0.05 to decide if the challenger can be discarded. We compare the outcome of the test to the ground truth to collect an accuracy. For a given pair $(p_{\mathcal{I}}, p_{\mathcal{C}})$ we run 10 seeds and report the average.

Figure 1 shows the collected accuracy over the time spent to make the comparison for two examples. Figure 1(*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 have a lower accuracy than random sampling. Figure 1(*b*) is a case in which discrimination and variance methods start with an advantage over random but are quickly on par with it. UDD and uncertainty are again largely sub-optimal.

Figure 2 synthesises the above described curves by computing the area under the curve (AUC) of each of them. The higher it is, the faster and more accurately the decision can be taken. This visualisation allows us to see how the methods compare but also the impact of the prior data given to the performance model. In all our scenarios we can see a clear correlation between the amount of known configuration and the AUC. This would allow the selection method to become more and more efficient along the configuration run and avoid wasting time in the last steps of a configuration run. On the other hand, adding more instances does not seem to significantly improve the performance. This is in line with the expectation that our instance sets aim at being homogeneous, thus adding more instances is not helping the model much.

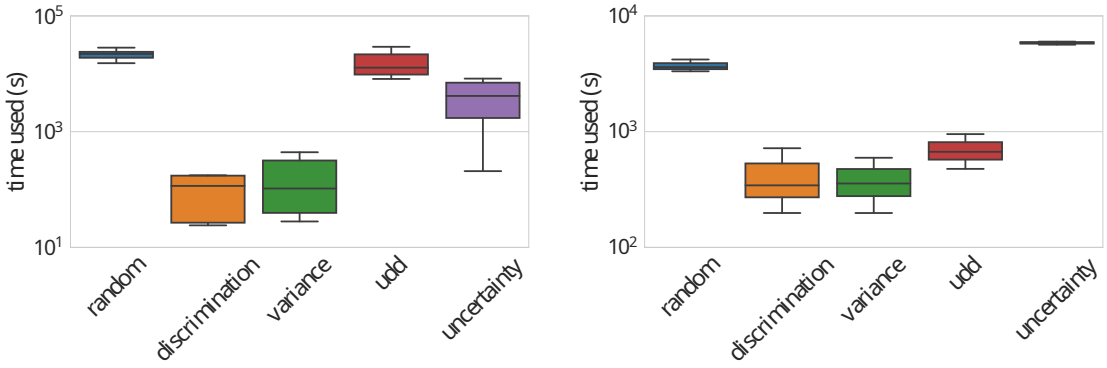
Regarding the selection methods, randomly sampling instances performs well but in most cases the discrimination and variance approaches do better.



(*a*) Kissat scenarios (*b*) Cplex scenarios

Figure 2: Area under the curve of the mean accuracy of the Wilcoxon test (p=0.05) on which among C_{ch} and C_{inc} performs best along the time spent on evaluations

The IBM dataset is unusual in this context, in that the UDD and uncertainty methods perform notably worse than random sampling. This could be explained by the large variation in the running time required to solve those instances. There are many instances which require long running times, but also many that are solved within a second. This means that selecting the wrong instance can have a dramatic effect on overall running time. This would explain why random sampling does not perform as well on this scenario as on the others, but also why adding more instances in the prior data improves the performance of our selection methods for this scenario.



(*a*) Kissat – crypto
30% instances, 20 configurations

(*b*) cplex – RCW2
40% instances, 40 configurations

Figure 3: Time used (in seconds) before taking a decision based on a Wilcoxon test (p=0.05) or reaching a maximum of 10 instance selected

Table 1: Median time in seconds for each method over every tested prior data

		kissat			cplex	
	ibm	cf	crypto	reg200	rcw2	MIPverify
random	1557	979.7	21243	576.8	4138	**29470**
discrimination	0.086	143.6	419.3	**96.66**	364.7	44390
variance	0.776	**95.16**	**372.2**	109.5	**342.0**	41365
udd	880.9	393.2	13483	379.7	1299	**28845**
uncertainty	**0.033**	330.8	2361.9	152.7	5974	39801

5.2. Compare configurations on unknown instances

To answer the second question, we place ourselves in phase 2 (see Section 3.1). We randomly select a fraction $p_{\mathcal{I}}$ of instances as $\mathcal{I}_{known}$ and a fraction $p_{\mathcal{C}}$ of configurations as $\mathcal{C}_{known}$. The random forest model is trained on all the available data. We choose $C_{inc} \in \text{argmax}_{c \in \mathcal{C}_{known}} m(c, \mathcal{I}_{selected})$ and we run this process for all $C_{ch} \in \mathcal{C} \setminus \mathcal{C}_{known}$ such that they cannot be told apart by a Wilcoxon test with a p value less than 0.05. We then select up to $n_{max} = 10$ instances on which we run both configurations until they can be told apart using the previous test.

For each selection method and each considered prior data, we gather the time used to decide between the two configurations at hand, *i.e.* the sum of the running times of C_{inc} and C_{ch} on $\mathcal{I}_{selected}$. Figure 3 shows the running times obtained for two examples. Most cases show that random is outperformed by all methods, with some exceptions in which the uncertainty or UDD methods are outperformed.

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 1. The data shows discrimination and variance outperform the other methods in almost all cases, with variance providing a speedup ranging from a 5.8 up to 3000 times speedup for variance compared to random. We note that such a high speedup for the IBM dataset is linked to a high variance in the running time distribution of the instances, which range from milliseconds to the timeout of 300 seconds.

6. Conclusion and future work

Based on the idea that selecting instances smartly could allow quicker comparison between configurations of an algorithm, we adapted four methods from several fields (Matricon et al., 2021; Gu et al., 2015) that could be applied to select instances. We identified two steps of the configuration problem in which such methods could be applied and designed two sets of experiments to asses their potential. 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 unknown challenger configuration 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. This encourages us to pursue future work to include those methods in a model-based algorithm configuration procedure.

References

Marie Anastacio and Holger H. Hoos. Model-based algorithm configuration with default-guided probabilistic sampling. In Thomas Bäck, Mike Preuss, André H. Deutz, Hao Wang, Carola Doerr, Michael T. M. Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020*, volume 12269 of *Lecture Notes in Computer Science*, pages 95–110. Springer, 2020. doi: 10.1007/978-3-030-58112-1_7.

C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney. Model-based genetic algorithms for algorithm configuration. In *Proc. IJCAI 2015*, pages 733–739, 2015.

Borja Ayerdi and Manuel Graña. Random forest active learning for retinal image segmentation. In Robert Burduk, Konrad Jackowski, Marek Kurzyński, Michał Woźniak, and Andrzej Żołnierek, editors, *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015*, pages 213–221, Cham, 2016. Springer International Publishing. ISBN 978-3-319-26227-7.

Tomáš Balyo, Nils Froleyks, Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland, 2020.

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

Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

William Jay Conover. *Practical nonparametric statistics*, volume 350. John Wiley & Sons, 1998. URL https://www.math.ttu.edu/~wconover/book.html.

T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Qiang Yang and Michael J. Wooldridge, editors, *IJCAI 2015*, pages 3460–3468. AAAI Press, 2015.

Stefan Falkner, Marius Lindauer, and Frank Hutter. Spysmac: Automated configuration and performance analysis of sat solvers. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 215–222, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24318-4.

Ian P. Gent, Bilal Syed Hussain, Christopher Jefferson, Lars Kotthoff, Ian Miguel, Glenna F. Nightingale, and Peter Nightingale. Discriminating instance generation for automated constraint model selection. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming*, pages 356–365. Springer International Publishing, 2014. doi: 10.1007/978-3-319-10428-7_27.

Yingjie Gu, Dawid Zydek, and Zhong Jin. Active learning based on random forest and its application to terrain classification. In Henry Selvaraj, Dawid Zydek, and Grzegorz Chmaj, editors, *Progress in Systems Engineering*, pages 273–278, Cham, 2015. Springer International Publishing. ISBN 978-3-319-08422-0.

Holger H. Hoos. Automated algorithm configuration and parameter tuning. In Youssef Hamadi, Eric Monfroy, and Frédéric Saubion, editors, *Autonomous Search*, pages 37–71. Springer, 2012. doi: 10.1007/978-3-642-21434-9_3.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. LION 5*, pages 507–523, 2011.

F. Hutter, M. López-Ibáñez, C. Fawcett, M. T. Lindauer, H. H. Hoos, K. Leyton-Brown, and T. Stützle. Aclib: A benchmark library for algorithm configuration. In *Proc. LION 8*, volume 8426 of *LNCS*, pages 36–40, 2014.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 186–202. Springer, 2010.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

Matthias König, Holger H Hoos, and Jan N van Rijn. Speeding up neural network verification via automated algorithm configuration. In *ICLR Workshop on Security and Safety in Machine Learning Systems*, 2021.

C. Luo, H. H. Hoos, S. Cai, Q. Lin, H. Zhang, and D. Zhang. Local search with efficient automatic configuration for minimum vertex cover. In *IJCAI-19*, pages 1297–1304. International Joint Conferences on Artificial Intelligence Organization, 2019.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. ISSN 2214-7160. doi: 10.1016/j.orp.2016.09.002.

Théo Matricon, Marie Anastacio, Nathanaël Fijalkow, Laurent Simon, and Holger H. Hoos. Statistical comparison of algorithm performance through instance selection. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:21, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-211-2. doi: 10.4230/LIPIcs.CP.2021.43.

Saeed Nejati and Vijay Ganesh. Cdcl(crypto) SAT solvers for cryptanalysis. In Tima Pakfetrat, Guy-Vincent Jourdan, Kostas Kontogiannis, and Robert F. Enenkel, editors, *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, CASCON 2019, Markham, Ontario, Canada, November 4-6, 2019*, pages 311–316. ACM, 2019. doi: 10.5555/3370272.3370307.

Yasha Pushak and Holger H. Hoos. Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 245–253, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371285. doi: 10.1145/3377930.3390211.

Li-Li Sun and Xi-Zhao Wang. A survey on active learning strategy. In *2010 International Conference on Machine Learning and Cybernetics*, volume 1, pages 161–166, 2010. doi: 10.1109/ICMLC.2010.5581075.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pages 16–30, 2011.