ECONSTOR
Make Your Publications Visible.

A Service of

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Weiß, Dimitri; Schede, Elias; Tierney, Kevin

Article — Published Version

Selector: Ensemble-Based Automated Algorithm Configuration

Journal of Heuristics

Provided in Cooperation with:
Springer Nature

Suggested Citation: Weiß, Dimitri; Schede, Elias; Tierney, Kevin (2025) : Selector: Ensemble-Based Automated Algorithm Configuration, Journal of Heuristics, ISSN 1572-9397, Springer US, New York, NY, Vol. 31, Iss. 3,
https://doi.org/10.1007/s10732-025-09561-6

This Version is available at:
https://hdl.handle.net/10419/330421

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

https://creativecommons.org/licenses/by/4.0/

WWW.ECONSTOR.EU

Mitglied der Leibniz-Gemeinschaft

Journal of Heuristics (2025) 31:28
https://doi.org/10.1007/s10732-025-09561-6

Selector: Ensemble-Based Automated Algorithm Configuration

Dimitri Weiß[1] · **Elias Schede**[1] · **Kevin Tierney**[1]

Received: 25 July 2024 / Revised: 14 February 2025 / Accepted: 30 May 2025 /
Published online: 26 July 2025
© The Author(s) 2025

Abstract
Solvers contain parameters that influence their performance and these must be set by the user to ensure that high-quality solutions are generated, or optimal solutions are found quickly. Manually setting these parameters is tedious and error-prone, since search spaces may be large or even infinite. Existing approaches to automate the task of algorithm configuration (AC) make use of a single machine learning model that is trained on previous runtime data and used to create or evaluate promising new configurations. We combine a variety of successful models from different AC approaches into an ensemble that proposes new configurations. To this end, each model in the ensemble suggests configurations and a hyper-configurable selection algorithm chooses a subset of configurations to match the amount of computational resources available. We call this approach SELECTOR, and we examine its performance against the state-of-the-art AC methods PyDGGA and SMAC, respectively. The new configurator will be made available as an open source software package.

Keywords Algorithm configuration · Ensemble optimization · SAT · MILP · CVRP · TSP · MAX-SAT

1 Introduction

The configuration of a solver's parameters has a drastic impact on its performance. Finding good parameters can make the difference between solving a problem instance to optimality or not solving it at all. Furthermore, parameters can greatly affect solver

✉ Dimitri Weiß
dimitri.weiss@uni-bielefeld.de

Elias Schede
elias.schede@uni-bielefeld.de

Kevin Tierney
kevin.tierney@uni-bielefeld.de

[1] Decision and Operation Technologies, Bielefeld University, Universitaetsstrasse 25, Bielefeld 33615, NRW, Germany

runtime, sometimes by orders of magnitude (Hutter et al. 2009). However, manually identifying high quality parameter configurations is extremely challenging, demanding domain knowledge and a significant time investment. Thus, finding such parameters is better left to automated approaches.

Offline automated algorithm configuration (AC) has emerged as a highly effective approach for determining parameters to solvers. A variety of offline AC methods have been developed, such as SMAC (Lindauer et al. 2022), GGA++ (Ansótegui et al. 2015)/OAT (Kemminer et al. 2024), irace (López-Ibáñez et al. 2016) as well as DEED (Indu and Velayutham 2024), a specialized configurator of differential evolution ensembles. These configurators employ varying strategies, from racing, which is the evaluation of configurations runtimes in tournaments, to machine learning (ML)-based models with different assumptions to learn about and discern configurations that yield high-quality performance. The efficacy of AC approaches has been demonstrated across a diverse array of optimization problems in Hutter et al. (2014).

We propose a new offline AC method that takes advantage of the wide variety of AC methods available and combines them into an ensemble. The no free lunch theorem for optimization (Wolpert and Macready 1997) tells us that there is no single optimization algorithm that performs best across all classes of problems, and the setting of AC is no exception. With this in mind, we integrate components of state-of-the-art AC methods in which these multiple AC components are queried and their suggestions are processed to find high-quality configurations. This general approach has been shown to be effective for solving few-shot black box optimization problems (Ansótegui et al. 2021).

Our new method, called SELECTOR, queries multiple AC methods that suggest configurations to solve a given set of problem instances. Since there are generally more suggestions than there are computational resources to run them, SELECTOR then reduces the set of candidates for evaluation using a learned model. These candidates are executed on the current subset of instances and the obtained feedback is provided to the AC methods in the ensemble. The contributions in this work are as follows:

1. We implement the first ensemble-based algorithm configuration approach.
2. We examine two different methods for selection from the pool of suggestions, namely a bandit-based method and an iterative scoring technique.
3. We conduct an empirical assessment of the performance gain of SELECTOR on an array of different AC scenarios.

2 Related work

In this section, we provide an overview of AC and related work. Then, we outline work on ensembles in related fields.

2.1 Algorithm configuration

Algorithm configuration can be applied in several settings, such as offline and real-time (Schede et al. 2022). SELECTOR is an offline AC method, but includes components

from the realtime AC setting, thus, in the following, we define AC for both settings and outline related work.

2.1.1 Offline AC

In the offline setting, the objective of AC is to identify a single high-quality configuration for a specified target algorithm for a specific dataset of problem instances. Following the formal notation introduced by Schede et al. (2022), we are given a set of problem instances $\mathcal{I}' \in \mathcal{I}$, with $\mathcal{I}$ representing the space of problem instances over which the probability distribution $\mathcal{P}$ is defined, single instances i, and a parameterized algorithm $\mathcal{A}$. A problem instance i can be described by an optional feature vector $\boldsymbol{f}_i \in \mathbb{R}^d$ with features $f_{i,1}, \ldots f_{i,d}$. The target algorithm $\mathcal{A}$ has parameters $p_1, \ldots, p_k$ open for configuration where each parameter p_i has a domain Θ_i. The search space of feasible parameter combinations (respecting conditional parameter settings) is then defined as $\Theta \subseteq \Theta_1 \times ... \times \Theta_k$. The primary goal in offline AC is to find a configuration $\boldsymbol{\theta}^* \in \Theta$ for $\mathcal{A}$ that minimizes a performance metric. This metric, defined as a cost function from the cost function space $\mathcal{C}$, is denoted as $c : \mathcal{I} \times \Theta \to \mathbb{R}$. The metric can represent a variety of aspects of the performance of $\mathcal{A}$, such as the runtime required to achieve a specified solution quality (referred to as *runtime configuration*) or the solution quality obtained within a given time limit (referred to as *solution quality configuration*). We note that in this work we focus exclusively on a single objective function. Consequently, the objective is to find

$$\boldsymbol{\theta}^* \in \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} \int_{\mathcal{I}} c(i, \boldsymbol{\theta}) d\mathcal{P}(i).$$

In practice, a proxy problem is solved, since the distribution $\mathcal{P}$ over $\mathcal{I}$ is not known. To approximate $\boldsymbol{\theta}^*$, an aggregation function $m : \mathcal{C} \times 2^{\mathcal{I}} \times \Theta \to \mathbb{R}$ is minimized on a set of training instances $\mathcal{I}_{\text{train}} \subseteq \mathcal{I}$. Typically, m is an arithmetic mean or a similar function computed by applying the approximation $\hat{\boldsymbol{\theta}}$ over $\mathcal{I}_{\text{train}}$. Consequently, the objective becomes

$$\hat{\boldsymbol{\theta}} \in \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}}\, m(c, \mathcal{I}_{\text{train}}, \boldsymbol{\theta}).$$

Various strategies have successfully addressed offline AC, employing diverse search paradigms such as evolutionary algorithms, local search, and model-based learning. The configurator SMAC (Hutter et al. 2011) operates on sequential model-based optimization principles, utilizing a Bayesian optimization paradigm to propose high-quality configurations. The genetic algorithm-based configurator GGA (Ansótegui et al. 2009) uses a racing mechanism and a specialized, tree-based crossover operator specifically designed to capture parameter dependencies, facilitating the generation of new configurations. GGA is extended in Ansótegui et al. (2015) with the incorporation of a random forest surrogate to aid in the generation of new configurations. The method PyDGGA (Ansótegui et al. 2021) extends GGA to distribute algorithm runs effectively across multiple machines on a high-peformance cluster. The .NET-based

Optano Algorithm Tuner (OAT) (OPTANO: OPTANO Algorithm Tuner Documentation 2022; Kemminer et al. 2024) provides a GGA-based .NET interface for AC. The irace configurator (López-Ibáñez et al. 2016) employs a distribution-based approach to sample configurations, evaluating them through racing. Following each race, it updates the sampling distribution based on the best-performing configurations, iterating this process to enhance configuration qualities until a predetermined threshold is achieved. Finally, the golden parameter search (GPS) (Pushak and Hoos 2020) method capitalizes on the structure of parameter space landscapes, assuming weak interaction among parameters and a unimodal response to changing parameter values. We refer the reader to Schede et al. (2022) for an extended overview and classification of existing AC techniques.

2.1.2 Realtime AC

In contrast to the previously defined offline AC setting in which the problem instance set is known in advance, in the *realtime* AC (RAC) setting, problem instances arrive one at a time and must be immediately solved. Consequently, at each time step t, a corresponding instance i_t must be solved. The underlying distribution of the problem instance set $\mathcal{I}$ is assumed to not be fixed, rendering learning a single offline configuration in advance inadequate. For a given instance i_t, the objective of RAC is to identify a configuration $\theta(i_t)$ that optimizes $m(i_t, \theta(i_t))$ within the set of possible configurations $\Theta(i_t)$, using information gathered from solving instances $i_1, \ldots, i_{t-1}$. Methods for solving the RAC setting usually maintain a pool of multiple configurations and select a subset of configurations to try on each new instance (Fitzgerald et al. 2014, 2015; El Mesaoudi-Paul et al. 2020). This selection of a few configurations from a larger pool is similar to what SELECTOR must do in each iteration, and we thus include the current state-of-the-art RAC method CPPL (El Mesaoudi-Paul et al. 2020; Weiss and Tierney 2022) in our portfolio and experiment with it for the selection task.

2.2 Ensemble approaches in areas related to AC

Ensembles of methods have a wide range of success in areas related to AC, such as algorithm selection (AS) and Bayesian optimization (BO). Furthermore, ensembles have a long history in the machine learning community (ML) (Polikar and Polikar 2006). We briefly highlight relevant work to SELECTOR in each of these areas.

2.2.1 Algorithm selection

Given a portfolio of multiple (potentially pre-configured) solvers for solving a particular class of problem instances, the goal of algorithm selection (AS) is to learn a policy using the solvers to solve unseen instances either as quickly as possible or to the best solution quality possible (as in AC), see, e.g., Bischl et al. (2016) and Kerschke et al. (2019). Algorithm selection techniques can also be used in dynamic settings, e.g., in Di Liberto et al. (2016), where they are used to select between heuristics within a mixed-integer programming solver. There are ensemble-based algorithm selection

approaches as well. The algorithm selector MachSMT (Scott et al. 2021) employs adaptive boosting to select SMT solvers based on pairwise ranking comparisons and empirical hardness models of the solvers. Furthermore, ensembles of algorithm selectors are shown to outperform single algorithm selectors by Tornede et al. (2023).

2.2.2 Bayesian optimization

SELECTOR makes use of Bayesian optimizers, such as SMAC, and is based on a Bayesian optimization ensemble technique (Ansótegui et al. 2021). BO approaches are highly effective at black-box optimization tasks, e.g., in the NeurIPS 2020 black-box optimization challenge (Turner et al. 2021). The winning method, heteroscedastic evolutionary Bayesian optimisation (HEBO) (Cowen-Rivers et al. 2020), includes a multi-objective acquisition ensemble algorithm. Moreover, ensemble strategies dominate the competition, such that the top 10 methods in the competition are comprised primarily of ensembles.

The hyperparameterized parallel few-shot optimization (HPFSO) approach (Ansótegui et al. 2021) also tackles the above setting. In this approach, an ensemble of eight candidate solution suggesters generates solutions to the optimization problem and features of the suggested solutions are computed. Based on the features, a selection-mechanism reduces the set of suggested solutions to a number of candidate solutions to be evaluated in parallel. The results are then provided to all ensemble members. This approach achieves a performance improvement over state-of-the-art optimization methods.

3 Selector

We now describe SELECTOR, an ensemble-based offline algorithm configurator. SELECTOR uses a parallel tournament-based evaluation scheme, meaning multiple configurations are run in parallel (as in GGA/GGA++) in multiple tournaments, see Figure 1. The AC methods (*suggesters*) propose up to n configurations each, which are passed into a selection mechanism that will be described in Subsection 3.4.1. The selection mechanism reduces the number of configurations from $m \times n$ to ℓ, the number of configurations required for a tournament. It uses features which are partly generated by suggesters and describe the suggested configurations. The configurations in the tournament are then paired with instances and executed. The obtained feedback is provided to the suggesters, although not all methods contain models requiring feedback. In the runtime configuration setting, the tournament can be stopped once one of the configurations has finished all instances. We further note that SELECTOR operates multiple tournaments in parallel; as soon as one tournament is finished, suggesters are updated and a new tournament is started.

We include two types of configuration suggesters in the ensemble: non-model-based and model based. The former generate configurations either randomly or according to some rules (e.g., an extended GGA crossover from Ansótegui et al. (2009)), whereas the latter contain trainable models that adapt to the feedback from the parallel runs during the configuration process, e.g., SMAC (Lindauer et al. 2022) and

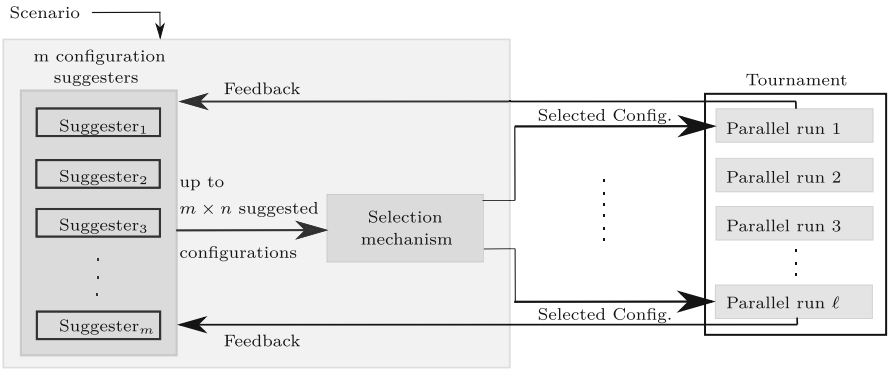

Fig. 1 Configuration suggestion and selection for evaluation.

Table 1 Classification of suggesters in model based and non-model based methods

	Suggester	Reference
	Default	-
Non-model based	Random	-
	Latin hypercube sampling	Timgates42 (2020)
	GGA graph crossover	Extension of Ansótegui et al. (2009)
Model based	SMAC	Lindauer et al. (2022)
	GGA++	Ansótegui et al. (2015)
	CPPL	El Mesaoudi-Paul et al. (2020)

GGA++ (Ansótegui et al. 2015). Table 1 gives an overview of the suggesters. In the following, we describe the suggesters included in our approach in more detail.

3.1 Non-model-based suggesters

The non-model-based suggesters are based on the ideas presented by Ansótegui et al. (2021) for black-box optimization, but extended for AC. We include four non-model-based suggesters in total, ranging from random configuration generation to sophisticated crossover techniques. We note that these suggesters play an important role in the technique, ensuring diversity in the configurations generated for each tournament and preventing premature convergence.

3.1.1 Default parameter suggester

The *default parameter* suggester is included within the ensemble to ensure the AC approach has a chance to yield a configuration with a performance at least as good as the default parameterization of the target algorithm. This is a common procedure and typically implemented in AC methods (e.g., in ParamILS Hutter et al. (2009) and GGA Ansótegui et al. (2009)) to enforce a baseline for minimum performance. Since

the set of configurations to be evaluated is selected from the suggested configurations, the default parameter configuration has to be injected from a dedicated configuration suggester to enforce its periodical evaluation and comparison to the quality of the other current configurations.

3.1.2 Random suggester

We generate configurations uniformly at random to encourage a diverse exploration of the search space. Note that values for parameters with numerically large bounds can be sampled from the logarithm space if desired by the user. Using the log space is common in AC methods when dealing with parameters with large ranges (e.g., in Lindauer et al. (2022)).

3.1.3 Latin hypercube sampling (LHS) suggester

An LHS configuration suggester is included as an alternative to uniform random sampling. We implement the LHS suggester using the scikit-optimize Python package (Timgates42 2020). The LHS suggester takes into account previously suggested configurations to avoid re-sampling configurations it already suggested to thus spread out the sampling of the parameter space.

3.1.4 GGA graph crossover suggester

The GGA parameter tree crossover mechanism introduced in Ansótegui et al. (2009) offers a way of combining two configurations together in a random process to create new configurations. The parameter tree mechanism assumes that the target algorithm's parameters have relations/dependencies that form a tree structure. However, not all parameters follow this assumption and several AC methods such as SMAC and irace support richer dependency structures. Thus, we extend the original GGA crossover mechanism to support a *parameter graph* instead of a tree. This graph allows us to easily account for forbidden and conditional parameter value combinations according to the parameter configuration space (PCS) format as introduced by Hutter et al. (2014). The graph crossover mechanism is described in detail in Appendix A.

3.2 Model-based suggesters

We include three model-based suggesters in our approach. These suggesters train their models based on the runtimes or solution qualities obtained by running the target algorithm. Thus, these approaches build models that improve as the configuration process progresses. Through our setup, they further benefit from the suggestions from other ensemble members, allowing them to see a diverse set of configurations. We note that our suggesters do not include all functionalities of the original algorithms, such as capping mechanisms, etc.

3.2.1 SMAC suggester

The SMAC method (Hutter et al. 2011) is based on BO with a random forest model with a local search-based acquisition function. We implement the SMAC suggester using the SMAC3 library (Lindauer et al. 2022). We provide the SMAC suggester with instance features and all runtime information that SELECTOR has gathered.

3.2.2 GGA++ suggester

The GGA++ suggester is a random forest model introduced in Ansótegui et al. (2015). It is trained using evaluation results gathered by SELECTOR. To generate configuration suggestions, a large batch of configurations is generated with the GGA graph crossover suggester. The model is then used to predict the performance of the configurations and returns the subset with the best performance.

3.2.3 CPPL suggester

The CPPL suggester is based on a contextual preselection bandit approach introduced by El Mesaoudi-Paul et al. (2020). The bandit model is trained using instance features and evaluation results from all previous tournaments. To generate suggestions, a batch of configurations larger than the requested number of suggestions is generated with an approach similar to the GGA++ suggester, but using a different crossover scheme. The generated configurations are assessed by CPPL in context of the problem instance features of the next instance set to be run. Note that the original setting for this method is RAC where only one problem instance is solved in tournaments. The best ranked configurations are picked as suggestions.

3.3 Streamlining the ensemble

While using multiple models offers the opportunity to benefit from the strengths of multiple different types of models, SELECTOR must also deal with the disadvantage of needing to train and evaluate multiple models many times over the course of its execution. To counteract the overhead, we target several computationally expensive components of SELECTOR and propose improvements.

3.3.1 Model size of the GGA++ suggester

The main computational time consumption of the GGA++ suggester is twofold: feature generation and model training. The time consumption in feature generation can be reduced by setting the number of estimators to 10, which is the default configuration for the number of estimators in the SMAC functionalities. This also reduces the time consumption through model training. We further reduce the overhead from model training by reducing the training data. For that, we introduce a parameter which dictates after which fraction of AC runtime to start reducing the history of configuration-performance pairs that are used for retraining of the GGA++ model. We

set the parameter to 15%. The data is reduced by deleting the same number of oldest entries as is added after a tournament.

3.3.2 Model size and suggestion function of the SMAC suggester

As in the approach with the GGA++ suggester, we adjust the SMAC functions used in our implementation to reduce the recorded data used for retraining of the model. The reduction is performed in the same manner, starting from the same fraction of allocated runtime dictated by the same parameter. Further, the SMAC suggester shows a high computational time consumption in the suggestion of configurations. This is on one hand due to the growing record of configuration-performance pairs, and on the other hand due to the local search function searching for more configuration suggestions than requested. The reason for this is that the SMAC configurator handles the number of suggestions dynamically, which is not desired in our approach. A simple adjustment of the included functionalities reduces the overhead from the SMAC suggester further.

3.3.3 Model training and configuration suggesting

Updating the models with results of the tournaments and acquiring configuration suggestions consumes considerable compute time. This overhead can partly be reduced by adjusting the use of available cores to increase the efficiency of the model functions. Model training and generation of configuration suggestions occurs in between tournaments. During this period, as many cores are available as configurations are run within a tournament in parallel. To efficiently utilize the number of available cores in between tournaments, we set SciPy and NumPy modules used in all three implementations of the model-based suggesters to use a number of cores equal to the number of configurations within a tournament. Importantly, this measure adjusts both modules to not use more than the set number of cores for the model-based suggester computations, which would lead to inefficiencies otherwise. Additionally, we set the random forest regressor of the GGA++ suggester to use a number of cores as mentioned above.

3.4 Iterative scoring selection mechanism

The goal of the iterative scoring mechanism is to select a diverse set of high-quality configurations using a learned, linear model. The mechanism works by scoring each configuration using a linear regression with weights that are learned offline (see, e.g., the hyper-configurable approach in Ansótegui et al. (2017)) and features that describe each configuration.

3.4.1 Selection mechanism

The selection is performed using the iterative scoring selection mechanism (ISSM) proposed in the HPFSO approach, as described in Algorithm 1. We select one configuration from the pool of suggested configurations in each iteration in line 4. Based on the current set of already selected configurations, a complete selection is simulated

J times (line 6). The simulation starts with an empty set of selected configurations and the first configuration is selected based on the score computed excluding features that relate configurations in the current selection to each other. For each iteration of line 8, the diversity features in the set of feature vectors F are updated based on the current selection S_n (line 9). At this stage of the simulation and for its duration, configurations added to S_n are regarded to be part of the set of configurations the diversity feature computation is based on. Due to this set of configurations changing during a simulation, diversity features need to be recomputed. The new feature values are then min-max normalized. A new feature vector $F(m)$ is generated with respect to the current set of selected configurations S_n, in the current simulation step in line 11. The feature vector f^m is then used to compute a score for the candidate configuration in line 12. The vector f^m represents the features that are based on the set of configurations in the current simulation step and w^T are feature weights defined before the AC process. Configurations are sampled for each simulation step based on the distribution s_c formed by scores s (line 15). The final set of selected configurations is determined by the frequency of appearance of the configuration in the complete selection sets of all simulations (line 20).

Algorithm 1: Iterative scoring selection mechanism (ISSM), based on Ansótegui et al. (2021)

1 **Input** : F : set of feature vectors for M candidates; w: feature weights; N: # of final candidates, J : # simulations
2 **Output** : S : Indices of selected candidates
3 **Initialize** $S \leftarrow \{\}$, $R \leftarrow \{1, 2, \ldots, M\}$
4 **while** $|S| < N$ **do**
5 $Q \leftarrow$ vector of length M of zeros
6 **for** j in $1, 2, \ldots, J$ **do**
7 $S_n \leftarrow S$
8 **while** $|S_n| < N$ **do**
9 Update and normalize diversity features in F w.r.t. S_n
10 **for** $m \in R \setminus S_n$ **do**
11 $f^m \leftarrow F(m)$
12 $s(f^m) \leftarrow \frac{1}{1+e^{w^T \cdot f^m}}$
13 **end**
14 $s_m \leftarrow s(f^m) / \sum_{m=1}^{M} s(f^m)\ \forall m \in R \setminus S_n$
15 Sample k from $R \setminus S_n$ with distribution $\{s_m\}$
16 $S_n \leftarrow S_n \cup \{k\}$
17 $Q[k] \leftarrow Q[k] + 1$
18 **end**
19 **end**
20 $S \leftarrow S \cup \{\text{argmax}\{Q\}\}$
21 **end**

3.4.2 Features

We compute features for each suggested configuration based on the ISSM approach (Ansótegui et al. 2021). The goal of the features is to provide a description of each configuration to our selection mechanism, both in terms of how good or bad the configuration is on its own, and how similar it is to other configurations. Table 2 provides an overview of the features. Except for feature 6, features 1 to 14 are also used in ISSM. Feature 6 is based on feature 3, and lets the model-based suggesters estimate the quality improvement of a configuration and is put in relation to the best performance recorded so far. In contrast, feature 3 uses the absolute value estimated by the suggesters. Features 15 to 18 are derived from features 3 to 6 and are introduced with our approach.

We divide the features into four categories: static, dynamic, diversity, and agreement. The static features give general information about the configurations and the state of the AC process. Note that it is possible to introduce a bias for configurations generated by specific model-based suggesters by setting the weights accordingly for feature 1. The dynamic features are statistical quantities derived from each model-based suggester and, as such, depend on the current states of the models. The diversity features evaluate the difference of the configurations to be selected to each other and to configurations already seen. These features are derived from the evaluation history of the AC process, except for feature 13, which sets the configurations in the pool of suggestions into relation to each other. The size of the evaluation history is controlled during the AC process by reducing it to k entries in intervals dictated by the parameter τ. The agreement features are generated by multiplying values of the dynamic features of each pair of model-based suggesters, e.g., feature 15a is computed by multiplying features 3a and 3b, thus indicating to what extent the suggesters "agree" with their assessment of a configuration. All feature values are normalized between zero and one. We include three model-based suggesters in the approach, hence each dynamic feature is computed from each model and each agreement feature is computed for each model-based suggester pair, resulting in a total of 12 dynamic and 12 agreement features.

3.4.3 Faster feature computation

Generating the previously described features consumes a high amount of runtime, particularly for diversity and dynamic features. Diversity features are computed based on the history of all configurations to have been evaluated, which increases the computation time with further progress into the AC process. Thus, we introduce a parameter τ that dictates in which interval the configuration record is reduced in the history. Furthermore, instead of simply deleting data, we reduce the record to k configurations identified by using the k-medoids algorithm introduced in Park and Jun (2009). The number of remaining configurations, k, is set to the number of parallel tournaments times the number of configurations in a tournament.

Runtime consumption by the computation of dynamic features is proportional to the runtime of the model-based suggesters. The highest overhead is generated by the GGA++ and SMAC suggesters, which we tackle separately, as described in Section 3.3.

Table 2 Extended selection features generated for configurations based on features described by Ansótegui et al. (2021)

Feature type	Specific feature
Static	1. One-hot encoding of suggester used
	2. Current tournament number / maximum tournament number
Dynamic	3(a-c). Expected quality based on {SMAC, GGA++, CPPL}
	4(a-c). Probability of quality improvement based on {SMAC, GGA++, CPPL}
	5(a-c). Uncertainty of quality improvement based on{SMAC, GGA++, CPPL}
	6(a-c). Expected quality improvement based on {SMAC, GGA++, CPPL}
Diversity	7. Percent of relatives in history(τ, k)
	8. Average quality of relatives in history(τ, k)
	9. Best performance of relatives in history(τ, k)
	10. STD of performance of relatives in history(τ, k)
	11. Diff. predicted/actual performance of relatives in history(τ, k)
	12. Avg. distance to configurations in history(τ, k)
	13. Avg. distance to suggested configuration
	14. Avg. distance to relatives in history(τ, k)
Agreement	15(a-c). Multiplied expected qualities from {SMAC×GGA++, GGA++×CPPL, CPPL×SMAC}
	16(a-c). Multiplied probabilities of quality improvement from {SMAC×GGA++, GGA++×CPPL, CPPL×SMAC}
	17(a-c). Multiplied uncertainties of quality improvement from {SMAC×GGA++, GGA++×CPPL, CPPL×SMAC}
	18(a-c). Multiplied expected quality improvements from {SMAC×GGA++, GGA++×CPPL, CPPL×SMAC}

3.4.4 CPPL as a selection mechanism

The CPPL method as an algorithm configurator is already equipped with all functionalities necessary to perform the AC process. This includes choosing candidate configurations from a pool of configurations based on the current state of the bandit model and problem instance features of the next instance set for evaluation. The implementation of the CPPL suggester already includes all necessary functionalities and does not need to be adjusted in any major way in order for it to be used as a selection mechanism.

The main difference between the CPPL suggester and CPPL as a selection mechanism is the input data. The CPPL suggester processes configurations parameter values as features. In preliminary experiments, we compare the quality of the selection of configurations by CPPL with configurations on the one hand described by parameter values and on the other hand described by the features in Table 2, which are also used for the ISSM selection mechanism described in 3.4.1. The quality of the selection of configuration using configuration features improves over the partially one-hot-encoded parameter values as input.

CPPL as a selection mechanism is implemented as follows. After each tournament, the selection mechanism receives evaluation results and features of the configurations, as well as the problem instances the results were generated with. In the next step, configurations are generated by the suggesters and presented to the CPPL selection mechanism as a pool to select from. For this, the problem instance features of the set of instances selected for the next tournament are included. Based on this input, a set of configurations is selected by CPPL and evaluated in the next tournament.

3.4.5 Configuring the configurator

Just like the algorithms they configure, AC approaches have parameters that affect their performance. A key difference between SELECTOR and state-of-the-art configurators is that SELECTOR's parameters have a much larger impact on its performance. In particular, the weights w^T must be configured for the ISSM to work at all. The bandit model of CPPL can be adjusted by a number of hyperparameters as well. The hyperparameters affect all functionalities of the CPPL suggester as well as the selection mechanism, which can be utilized as SELECTOR's selection mechanism. We develop a mechanism for configuring these parameters that we describe in Section 4.2.2, which is also applied for a fair comparison of both selection mechanisms.

4 Experimental Results

We assess performance of SELECTOR on a benchmark of 14 AC scenarios on four different optimization problem types and six different target algorithms. All experiments use compute nodes with AMD Milan 7763 processors with 128 cores each running at 2.45 GHz. We allow SELECTOR to use 128 cores on a single machine to address the following research questions:

Table 3 Scenarios considered in the experiments

Scenario	Metric	Domain	Target algorithm	Dataset
1	Runtime	SAT	CaDiCaL	Fuzz
2				BMC
3				LABS
4				Limited
5			Glucose	Fuzz
6				BMC
7				LABS
8				Limited
9		MIP	CPLEX	Regions
10				RCW
11				CVRP15
12	Solution Quality	MaxSAT	D-SAT2	MaxSAT
13		TSP	ACOTSP	tsp-rue-1000-3000
14			ACOTSP-VAR	tsp-rue-3000

RQ1: Given a set of configurations, can either CPPL or our ISSM identify the best configurations in the set?
RQ2: What is the contribution of the ensemble members towards the configuration found?
RQ3: Does SELECTOR find better configurations (either in terms of runtime or solution quality) compared to the state-of-the-art AC methods?

4.1 Datasets and target algorithms

We examine SELECTOR on a wide range of scenarios that are first partitioned into two categories based on the configuration objective of the scenario, namely runtime and solution quality. In the runtime optimization setting, we use the SAT solvers CaDiCaL (Han 2020) and Glucose (Audemard and Simon 2009, 2012), as well as the mixed-integer linear programming (MILP) solver CPLEX (IBM: IBM ILOG CPLEX Optimization Studio 2016). In the solution quality optimization scenario, we use the MaxSAT solver D-SAT2 (Ansótegui et al. 2017) and two ant colony optimization (ACO) approaches for the traveling salesperson problem (TSP): ACOTSP (Stuetzle 2004) and ACOTSP-VAR (López-Ibáñez and Stützle 2014). Table 3 gives an overview of the scenarios.

We pair the following datasets with the target algorithms. Both SAT solvers, CaDiCaL and Glucose, are configured on the instance sets Circuit Fuzz (Fuzz), Low Autocorrelation Binary Sequence (LABS), Bounded Model Checking (BMC) and a downsized set of BMC (Limited) which are all part of AClib (Hutter et al. 2014). These instance sets are filtered to only include instances that are solved within 30 seconds or more with the default parameter settings of the respective target algorithms, except for the Limited set. Limited contains instances of BMC that are solvable by

the default between 5 and 30 seconds.The MILP solver CPLEX is configured with the instance sets "Regions", which models combinatorial auction problems with 200 regions, an instance set of Red-Cockaded Woodpecker problems (RCW) derived from the ACLib, and capacitated vehicle routing problems with 15 customers (CVRP15), which are generated using the same distribution as the instances used in the experiments by Kool et al. (2019). These instances are also filtered to include only instances that need at least 30 seconds to be solved with the default parameterization of CPLEX. The time limit in all the described scenarios is 300 seconds, except for the Limited dataset, which is given a maximum of 30 seconds of runtime.

For the solution quality setting, we configure the MaxSAT solver D-SAT2 on MaxSAT instances generated with an implementation of the procedure described by Escamocher et al. (2019). The TSP instances tsp-rue-1000-3000 and tsp-rue-3000 used in configuration of the ACOTSP and ACOTSP-VAR algorithms, respectively, are also derived from the AClib. The runtime cutoff for D-SAT2 is set to 30 seconds, and the cutoff is 20 seconds in the TSP scenarios.

4.2 RQ1: Identifying high-quality configurations

To assess if it is possible to effectively select from the suggestions of our ensemble, we conduct two experiments to compare the configured proposed selection mechanisms and fully configure the chosen selection mechanism. For both experiments, we generate an offline dataset in which the suggesters propose configurations, and instead of selecting only a subset to evaluate, we run all of them (still respecting the timeout) to have information about how they perform. We record data for all scenarios except for the Limited scenarios, since these instances are already contained in the BMC scenarios. With this data, we can assess how effectively our proposed selection mechanisms can identify high-quality configurations, which is also used to assess the quality of a hyperparameter configuration of the selection mechanisms.

4.2.1 Choosing a selection mechanism

We evaluate the performance of both potential selection mechanisms: CPPL and ISSM. To allow for a fair comparison, the parameters of both selection mechanisms are configured first. This experiment consists of the following steps.

1. Generate offline data of tournaments for all scenarios.
2. Run hyperparameter configuration of both selection mechanisms with the generated offline data.
3. Evaluate the hyperparameter configured selection mechanisms on offline data.

In step 1, we run SELECTOR on a scenario and log all the suggestions of the suggesters. We then take all configurations suggested and run them until a timeout as described in 4.1. We can thus examine our two selection approaches and see whether they are able to identify the best suggestions or not. The offline data is then split into three folds by scenarios. We use all three possible permutations of training and test set combinations and perform a 3-fold cross-validation. This results in one hyperparameter configuration of a selection mechanism for each combination. A limitation of this

Table 4 Comparison of selection mechanisms, with test folds highlighted in gray. All scenarios have a minimization objective, and the results are aggregated in averages of the folds

Folds	ISSM				CPPL			
	Basel.	Conf. 1	Conf. 2	Conf. 3	Basel.	Conf. 1	Conf. 2	Conf. 3
Fold 1	0.34	0.16	**0.15**	**0.15**	0.36	0.33	0.36	0.32
Fold 2	0.38	**0.18**	0.20	**0.18**	0.33	0.26	0.34	0.37
Fold 3	0.29	0.15	**0.13**	0.15	0.29	0.32	0.36	0.35

approach is that we can only examine the performance of the selection approaches in a single iteration and not over an entire configuration run. We nonetheless believe this experiment gives us valid insights into which selection mechanism will work best in SELECTOR.

In step 2, the offline data gathered in step 1 is used to configure the hyperparameters of both selection mechanisms with the OAT (OPTANO: OPTANO Algorithm Tuner Documentation 2022) configurator. Each tournament of an offline run (i.e., all suggesters suggest configurations) is treated as a problem instance by OAT. The performance of the selection mechanisms is measured as follows. All tournaments in the offline data of a scenario are considered in the sequence they were recorded. The selection mechanism selects configurations from the pool of potential configurations. Since we record the complete performance data, the true ranking of the configurations of the complete tournament is known. The rankings are squared to achieve a higher penalty on selecting worse rankings and normalized between zero and one, where a lower value represents better rankings. These values are then aggregated in each tournament. At the end of running a scenario, the average of the rankings over all tournaments is given as feedback to OAT. OAT is run for 500 generations, and we conduct three configuration runs for each selection mechanism, capturing all permutations of the three folds. This results in one individual hyperparameter setting per permutation of the folds.

In step 3, we evaluate the performance of the resulting configurations by running the configured selection mechanisms on the offline data three times and averaging the results. We consider the performance on all scenarios, however, the performance on test folds is in the focus of this experiment. The results on the test folds yield insights about the generalizability of the selection mechanism to unseen scenarios. Thus, this experimental setting is particularly challenging, as the test scenarios can differ greatly from the training scenarios.

Table 4 evaluates ISSM against CPPL. Additionally, a baseline performance of the selection mechanisms is included. For ISSM, it is determined by running the iterative scoring selection mechanism with all values set to one. This has the same effect as removing the weights from the scoring mechanism. For the baseline of the CPPL selection mechanism, we run it with a hyperparameter configuration as implemented in our ensemble. This hyperparameter setting is not adjusted to the use case of a selection mechanism in SELECTOR. The configurations in the offline data are ranked by their recorded performance in each tournament, squared and normalized between

Table 5 Results of hyperparameter configuration trained and evaluated on full offline data compared to best results from Table 4

Folds	Baseline	Best in Table 4	Full data result
Fold 1	0.35	**0.19**	0.21
Fold 2	0.35	**0.18**	0.19
Fold 3	0.31	**0.14**	**0.14**
Percentage improvement		-49.5	-46.5

zero and one. The average of the ranks of the configurations selected by the respective selection mechanism for each tournament in the offline data is presented as the result in Table 4. We aggregate the evaluation results of the individual scenarios in averages of the folds for conciseness.

Although CPPL has a slight edge in the baseline performance, ISSM and CPPL do not differ significantly. However, the configured ISSM has improved performance in all folds with all configurations, whereas the CPPL selection mechanism only improves over the baseline in fold 1 with configurations 1 and 2 and in fold 2 with configuration 1. Note that the CPPL mechanism improves performance with configuration 3 on a test fold, and ISSM improves performance on all test folds, which suggests generalizability of our hyperparameter configuration approach.

All best performances on the different folds are achieved by the configured ISSM. Moreover, ISSM outperforms the configured CPPL selection mechanism on each fold in every permutation of the folds. We conclude that ISSM is better suited for our approach. Thus, we continue with it as the selection mechanism in SELECTOR.

4.2.2 Configuring Selection Weights

Having chosen ISSM as our selection mechanism, we must now configure it on the complete offline data. We use the offline data generated in step 1 in Subsection 4.2.1. The hyperparameter configuration is performed in the same way as in step 2 in Section 4.2.1. However, instead of splitting the offline data into three folds, we use the complete data for training as well as evaluation. To obtain a view at the quality of the resulting hyperparameter configuration, we compare the evaluation results to results with the default hyperparameter configuration as well as the best results obtained with any of the three hyperparameter configurations in Section 4.2.1. The comparison is given in Table 5 where results are computed as described for Table 4.

The performance results of ISSM with the hyperparameter configuration obtained on the full offline data does not improve over the best performances in Table 4. However, the small difference and the tie in fold 3 indicate a high quality of the hyperparameter configuration. This is underlined by only 3 percentage points difference in percentage improvement. Thus, we use the hyperparameter configuration resulting from the configuration on the full offline data in our implementation and continue the experiments with it.

4.3 RQ2: Contribution of ensemble members

To illustrate the contributions of the ensemble members, we evaluate how often configurations generated by a suggester are selected and how often they win a tournament. The data used for the evaluation consists of logs of AC runs with SELECTOR in all scenarios, as illustrated in Section 4.4. The results are aggregated in Table 6. Note that these results highly depend on the hyperparameter configuration resulting from the procedure described in Section 4.2.2 and a different distribution is possible with a different hyperparameter configuration.

Table 6 illustrates a clear difference between runtime (1 - 11) and solution quality scenarios (12 - 14). In the solution quality scenarios, the selection highly concentrates on configurations suggested by either SMAC or GGA++, whereas in runtime minimization scenarios suggestions are more balanced across the ensemble. Furthermore, configurations suggested by the GGA graph crossover suggester are only selected in the solution quality scenarios. This indicates that the contribution of these configurations is negligible in the runtime scenarios according to the selection mechanism based on the hyperparameter configuration generated in Subsection 4.2.2.

Configurations suggested by LHS and CPPL are regularly chosen in the runtime scenarios, but never win any tournament. This indicates that each tournament includes a configuration of some other suggester that outperforms configurations originating from LHS and CPPL. Let us emphasize, however, that these configurations can still significantly contribute to the AC process, e.g., in the updating of the models of the other selection mechanisms. An explanation for the underperformance of configurations generated by the CPPL suggester is that we are using it in a setting that it was not developed for. A core assumption of the realtime AC setting is that problem instances are to be solved in a sequence, which is not the case in offline AC. CPPL suggests high quality configurations for a single problem instance, such that the incorporation of features of multiple problem instances may dilute the quality of the configuration.

A surprising result is that on the CaDiCaL LABS scenario nearly 100% of the winning configurations are generated by the random suggester. Note that SELECTOR does not improve over the default CaDiCaL configuration in this scenario in the experiment in Subsection 4.4. This scenario is dominated by the SMAC method, as can be seen in Table 7 in Section 4.4. However, configurations suggested by the SMAC suggester represent only 1.89% of all configurations selected in this scenario. We thus posit that there remains significant opportunity within the area of algorithm configuration to propose new configuration suggestion approaches.

4.4 RQ3: Comparison to SMAC and PyDGGA

We compare SELECTOR to the state-of-the-art algorithm configurators SMAC and PyDGGA. All AC runs are provided all 128 CPU cores for 48 hours in each scenario. The configurations found by the AC methods, as well as the default configurations of the target algorithms, are then evaluated on the test sets of the respective scenarios. Each configuration found is evaluated five times. The performance of the default configurations function as a baseline for both our approach and the state-of-the-art

Table 6 Percentages of configurations selected to run in tournaments (S) and winning tournaments (W) by origin. Scenarios are numbered according to Table 3

Scen.	Default		Random		LHS		G. cross.		SMAC		GGA++		CPPL	
	S	W	S	W	S	W	S	W	S	W	S	W	S	W
1	1.69	64.25	20.09	14.55	19.49	0.00	0.00	0.00	2.09	0.00	37.63	21.18	18.97	0.00
2	1.68	73.76	21.28	10.20	19.34	0.00	0.00	0.00	1.57	0.00	37.52	16.16	18.60	0.00
3	1.92	0.09	20.76	99.89	19.35	0.00	0.00	0.00	1.89	0.00	37.18	0.02	18.89	0.00
4	0.51	18.06	20.04	0.62	19.74	0.00	0.00	0.00	1.18	0.00	39.23	81.36	19.28	0.00
5	0.65	0.01	19.05	1.76	19.68	0.00	0.00	0.00	3.31	0.31	38.74	97.97	18.55	0.00
6	0.61	6.63	17.47	83.24	19.57	0.00	0.00	0.00	4.99	5.61	38.55	4.59	18.79	0.00
7	1.14	0.00	17.04	14.91	19.54	0.00	0.00	0.00	5.56	8.52	37.95	76.65	18.75	0.00
8	0.70	0.07	18.63	0.01	19.64	0.00	0.00	0.00	3.73	0.63	38.63	99.34	18.65	0.00
9	0.99	0.00	18.84	1.33	19.62	0.00	0.00	0.00	6.56	3.19	38.28	95.01	15.70	0.00
10	1.20	33.29	18.91	0.00	19.40	0.00	0.00	0.00	6.84	1.54	37.63	65.27	16.02	0.00
11	0.81	0.03	16.11	1.35	19.51	0.00	0.00	0.00	8.32	37.73	38.24	60.97	17.00	0.00
12	6.22	0.00	1.65	5.11	0.00	0.00	0.38	0.00	85.33	89.67	6.41	5.28	0.00	0.00
13	5.96	0.00	1.51	0.44	1.47	0.00	4.95	7.92	11.47	12.75	72.02	78.89	2.59	0.00
14	10.09	0.32	0.90	0.00	0.11	0.00	0.20	0.00	41.90	26.93	44.89	72.77	1.87	0.00

methods. The results are illustrated in Table 7 in seconds for the runtime scenarios and as output of the target function value in Table 8 for the solution quality scenarios.

Overall, SELECTOR outperforms SMAC and PyDGGA. In 10 out of 14 scenarios the configurations found by SELECTOR outperform or tie the competition. This is the case in six scenarios for SMAC and seven scenarios for PyDGGA. Note that each AC method dominates in exactly two scenarios and ties with at least another method in the remaining scenarios. A reduction in the number of timeouts is achieved by all three AC methods. In terms of timeouts, PyDGGA and SELECTOR tie, although the number of timeouts is reduced to different degrees in different scenarios. Generally, the configurations resulting from AC runs of the different methods are rarely the same. However, there are two exceptions in our experiment, namely the configurations in the Limited BMC scenarios for CaDiCaL and Glucose, which are the default configurations. The slight deviations of the performances are due to randomness.

We compute the percentage improvement of the performance with Equation (1), accounts for negative objective function values.

$$nc_i = sgn(\text{perf}_{\text{AC}_i}) \frac{\text{perf}_{\text{AC}_i} - \text{perf}_{\text{default}}}{-\text{perf}_{\text{default}}} \tag{1}$$

Taking into account the baseline performances of each scenario and averaging these values, SELECTOR clearly outperforms both AC methods in the runtime scenario, see Table 7. Compared to this, the average percentage improvement in performance achieved by SELECTOR in the solution quality scenarios is lower and on par with the performance of SMAC, see Table 8. However, combining the percentage improvements of performance of both runtime and solution quality scenarios results in percentage improvements of performance of -17.89% for SELECTOR, -12.86% for SMAC and -8.79% for PyDGGA versus the default configurations. We provide further details on these experiments in Appendix C. We thus conclude that SELECTOR's ensemble offers significant performance benefits over state-of-the-art AC techniques.

5 Conclusion and future work

We introduced an ensemble-based AC approach comprised of several configuration suggesters, including key functionalities of three model-based AC methods. In this approach, configurations to be evaluated need to be selected from the pool of suggested configurations. Hence, we examined the suitability of two candidate selection mechanisms by performing hyperparameter configuration first and implementing the better suited method. At the same time, we demonstrated the generalizability of the hyperparameter configuration results by applying it on three folds of offline data. Another hyperparameter configuration run on the full offline data set yielded the hyperparameter configuration we then implemented in our approach. Running SELECTOR in actual AC scenarios and comparing the results with performances of the state-of-the-art methods SMAC and PyDGGA, as well as the default target algorithm configurations performance as a baseline, we come to the conclusion that the ensemble-based approach improves AC performance. The assumption of an ensemble-based approach

Table 7 Results on the test set for runtime scenarios. Average runtime and numbers of timeouts from five runs

Domain	TA	Dataset	Runtime				Number timeouts			
			Def.	SMAC	PyDGGA	Sel.	Def.	SMAC	PyDGGA	Sel.
SAT	CaDiCaL	Fuzz	101.0	78.35	98.51	**74.41**	27.6	**17.2**	26.4	**17.4**
		BMC	87.92	92.58	104.58	**80.76**	17.6	18.0	21.2	**16.4**
		LABS	233.17	**218.14**	234.12	233.07	91.2	**82.8**	87.8	92.0
		Limited	**9.89**	**10.93**	15.39	**9.61**	**0.0**	**0.0**	4.6	**0.0**
	Glucose	Fuzz	84.31	65.98	**45.19**	52.25	14.8	11.2	**6.0**	**6.0**
		BMC	70.17	71.53	**68.40**	**67.37**	**19.4**	21.0	**19.8**	**19.4**
		LABS	216.47	215.95	**207.24**	**206.31**	81.8	81.8	**76.0**	81.6
		Limited	**10.34**	**10.53**	**10.26**	**10.52**	**0.0**	0.8	1.0	**0.0**
MIP	CPLEX	Regions	15.65	4.95	**3.93**	**3.68**	**0.0**	**0.0**	**0.0**	**0.0**
		RCW	41.18	**32.93**	43.26	**34.04**	20.0	4.8	**0.2**	**0.0**
		CVRP15	136.40	110.84	**84.25**	91.59	150.4	122.2	**81.8**	98.4
Average			91.5	82.97	83.19	**78.51**	38.43	32.97	**29.52**	**30.1**
Average percentage improvement (%)				-12.58	-8.13	**-18.99**				

Table 8 Results on the test set for quality scenarios. Average quality from five runs

Algorithm	Dataset	Default	SMAC	PyDGGA	Selector
D-sat2	MAXSAT	3.47	**2.74**	3.0	**2.74**
ACOTSP	tsp-rue-1000-3000	3.990×10^7	$\mathbf{3.207}\times10^7$	3.250×10^7	3.220×10^7
ACOTSP-VAR	tsp-rue-3000	-0.879	-0.888	**-0.892**	**-0.890**
Average percentage improvement (%)			**-13.89**	-11.18	**-13.86**

increasing adaptiveness to different scenarios and thus exploiting the advantages of each component in the ensemble proved to work as expected. This conclusion is also reinforced by the data of contributions of configurations originating from different suggesters to the AC process.

Our approach has some key limitations. With the introduction of an ensemble, the complexity of the AC method increases. The algorithm configurator is composed of many interdependent components and dynamic interactions that are adjusted in combination. This complicates customization if it is required in certain application cases. Additionally, the presence of three models leads to a disproportionate increase of overhead compared to methods with a single model. While we managed to introduce methods to alleviate some of the overhead, it will grow quicker than with other methods configuring target algorithms with large parameter spaces or in the presence of a high amount of problem instance features. Further, the hyperparameter configuration of SELECTOR is expensive and we cannot guarantee that the parameters we found in this work are effective for all possible scenarios. This invokes possible additional costs that can be untenable in certain use cases.

In future work, an exchange or extension of configuration suggesters can lead to further improvement of the AC method's performance and a higher adaptability of the approach to scenarios we did not consider in our experiments. Further, the hyperparameter configuration approach used to adjust the weights of the selection mechanism can be replaced or adjusted, thus yield an improved hyperparameter configuration. Introducing additional features used by the selection mechanism, which possibly are of higher order, may better capture the solution space of the AC problem. Besides that, an investigation of alternative selection mechanisms, e.g. other bandit approaches, optimizers or machine learning methods, offers an interesting area of research.

Appendix A GGA graph crossover

The GGA graph crossover suggester is derived from the GGA parameter tree crossover mechanism described by Ansótegui et al. (2009). In the following, we explain the method underlying the GGA graph crossover suggester. We first describe the algorithm and illustrate it in an example after that.

A.1 GGA graph crossover algorithm

The GGA graph crossover method is described in Algorithm 2. The input for this method is the parameter definition p, the graph structure G, which is precomputed based on p and the parent configurations C and N. The parameter definition p contains parameter names and their value spaces, as well as the constraints of forbidden and conditional values. The nodes of G represent parameters and the edges either connect the parameters in the order they are defined in p or by forbidden value pairs or conditional parameter value rules. The output of the method is a new configuration.

The algorithm initializes with empty dictionaries in line 3. They are used to record the labels of the parent configurations C and N assigned to nodes, parameter values

assigned to parameters and the paths associated with nodes, respectively. In lines 4 to 6 the root node of the graph is extracted and set as the start point of the main loop in which labels are assigned to nodes (line 7). This loop runs as long as the set S contains nodes to be processed.

The label assignment is dictated by the labels of the parent nodes. If there is only one parent node, its label is assigned to the child node. If there are multiple parent nodes, the label is assigned uniformly at random but weighted with the ratio of parent labels (line 20). Additionally, the label is flipped to the other parent configuration with a probability of γ (lines 23 and 24).

The node labeling is processed in the order given by G and the paths to the nodes are recorded. To avoid cycling in the graph, a simple condition in line 12 functions as a stopping criterion. If there is a record for the child node in the paths and the parent node is already part of the record, the child node is not added to S.

After all nodes are assigned a label, the configuration is set to values according to the parent labels in lines 29 to 29. Following this, the parameter values are mutated with a probability δ (line 37) and violations of the forbidden and conditional parameter value rules according to p are corrected (line 38). Finally, the configuration setting is processed into the correct format and returned.

Algorithm 2: GGA graph crossover, extending the crossover in Ansótegui et al. (2009)

```
Input  : Graph structure G, configurations C, N, parameter space p
Output: Configuration
Initialize labels, values, paths ← {}, {}, {}
nodes ← keys of G, curNode ← nodes[0], S ← curNode
Set labels[curNode] randomly
paths[curNode] ← [curNode]
while S ≠ ∅ do
    curNode ← pick(S), S ← S \ {curNode}
    for child in G[curNode] do
        if curNode ! = child then
            if child in paths then
                if curNode not in paths[child] then
                    paths[child] ← paths[child] + [curNode]
                    Set labels of child w.r.t. paths and all parent labels
                    S ← S ∪ {child}
                end
            end
            else
                paths[child] ← paths[curNode] + [child]
                Set labels of child w.r.t. paths and all parent labels
                S ← S ∪ {child}
            end
            if rand() mod 100 < γ then
                Flip labels of child
            end
        end
    end
end
for parameter in p.names do
    if labels[parameter] is labeled as C then
        values[parameter] ← C[parameter]
    end
    else
        values[parameter] ← N[parameter]
    end
end
Perform mutation on values with probability δ
Resolve forbidden and conditional value discrepancies according to p
return configuration(values)
```

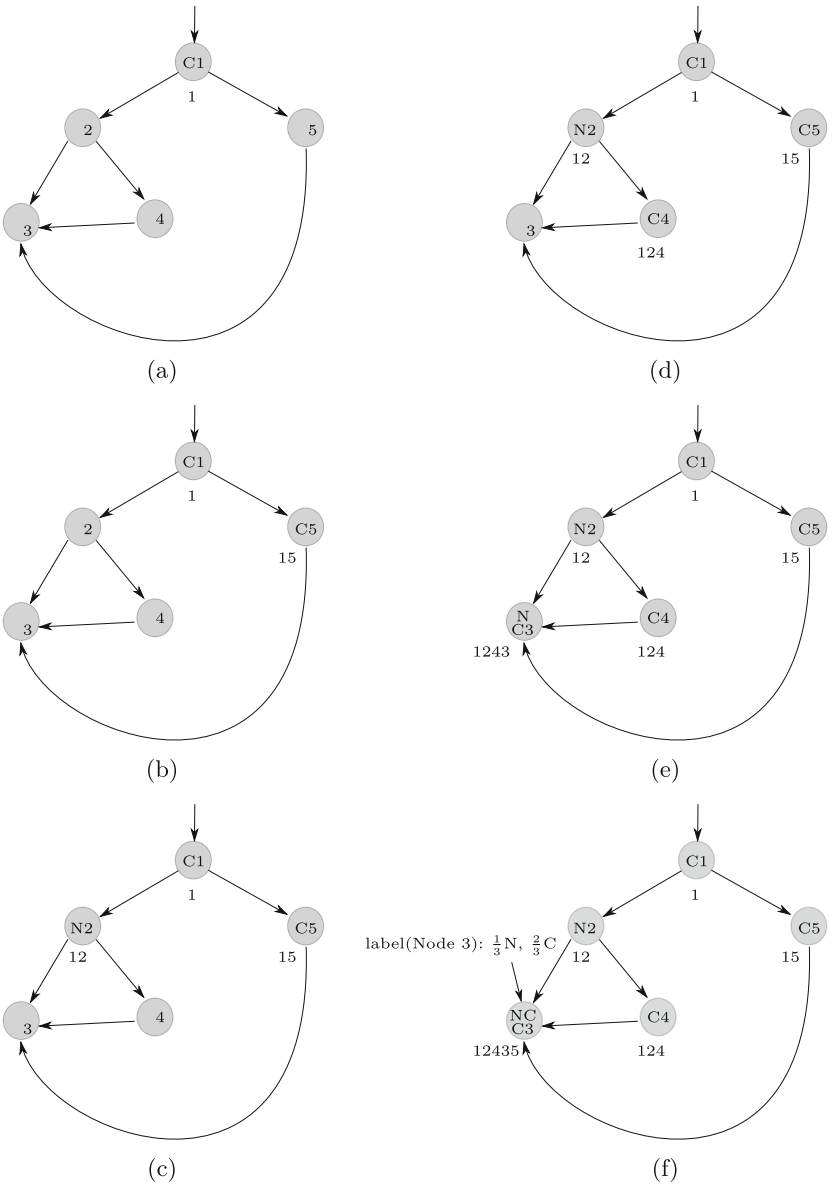

Fig. 2 GGA graph crossover label assignment example.

A.2 GGA graph crossover label assignment example

The label assignment procedure in the GGA graph crossover suggester is exemplified in Figure 2. For simplicity, we use numbers as parameter names. The labels C and N coincide with assignments of values from parent configurations C or N. The terms C and N are borrowed for consistency from the tree-based GGA procedure in which they

refer to the *competitive* and *non-competitive* populations of a genetic algorithm, but we note that we do not maintain such populations in SELECTOR. In this example, the root node is randomly assigned the label C, as illustrated in Subfigure 2a. At the same time, a recording of the path is made, i.e. 1 is recorded because node 1 is traversed.

In Subfigure 2b, node 5 is chosen to be processed next and is now labeled C5, as it will take the parameter from the C parent configuration. Note that choosing node 2 first would lead to the same outcome regarding the recorded paths. Node 5 is assigned the same label as node 1. In Subfigure 2c, the next child node of node 1 is processed. By a small probability, the label of node 2 is flipped as described in line 24 in Algorithm 2. In Subfigure 2c it is processed and the label is flipped.

Finally, node 3 is reached in Subfigure 2e and the recorded path for node 3 includes both parent nodes. The last connection is processed in Subfigure 2f. At this point, all parent nodes are included in the path recorded for node 3. Two of the parent nodes have the label C and one parent node has the label N. The label assignment for node 3 is still randomly generated but weighted according to the ratio of the parent nodes.

Appendix B Additional information on datasets

Table 9 shows the distribution of problem instances in the runtime scenario. Note that problem instance in the Limited scenarios for CaDiCaL and Glucose are also present in the respective BMC scenarios. Table 10 shows the distribution of problem instances in the quality scenarios. The ratio of numbers of problem instances in the training and test sets varies between the scenarios. This constitutes a further difference in challenges for AC methods between the scenarios.

Table 9 Number of instances for runtime scenarios

Domain	Algorithm	Dataset	#Instances train set	#Instances test set
SAT	CaDiCaL	Fuzz	97	129
		BMC	227	84
		LABS	131	129
		Limited	91	42
	Glucose	Fuzz	85	104
		BMC	282	127
		LABS	122	130
		Limited	128	78
MIP	CPLEX	Regions	281	258
		RCW	305	304
		CVRP15	226	534

Table 10 Number of instances for quality scenarios

Algorithm	Dataset	#Instances train set	#Instances test set
D-sat2	MAXSAT	137	864
ACOTSP	tsp-rue-1000-3000	50	250
ACOTSP-VAR	tsp-rue-3000	50	50

Appendix C Additional results on train set

We evaluate the methods compared in Subsection 4.4 on the training sets as well. The configurations resulting from AC runs with the different methods were evaluated five times. The results show a similar trend. While SELECTOR outperforms the state-of-the-art or ties with another method in a lower rate, a clear dominance by SELECTOR in average percentage improvement can be seen in Tables 11 and 12. In addition, on the training sets our proposed method outperforms the state-of-the-art in the reduction of timeouts.

Table 11 Results on the training set for runtime scenarios. Average runtime and numbers of timeouts from five runs

Domain	TA	Dataset	Runtime				Number timeouts			
			Def.	SMAC	PyDGGA	Selector	Def.	SMAC	PyDGGA	Selector
SAT	CaDiCaL	Fuzz	108.27	82.85	113.93	**72.68**	21.8	13.2	23.6	**10.0**
		BMC	147.24	152.15	171.23	**144.69**	**84.8**	88.0	106.8	**83.6**
		LABS	220.24	**205.58**	216.61	219.58	89.6	**80.4**	85.6	88.6
		Limited	**18.03**	**17.49**	21.55	**18.81**	**0.0**	2.0	11.4	**0.0**
	Glucose	Fuzz	81.74	53.52	**49.02**	56.62	11.0	**6.0**	**6.0**	8.0
		BMC	136.80	140.72	**127.34**	132.43	96.2	96.8	**87.0**	93.4
		LABS	225.79	**216.63**	219.90	219.59	83.2	81.2	**79.0**	81.2
		Limited	16.52	**13.34**	**12.11**	14.50	**0.0**	1.4	1.4	**0.0**
MIP	CPLEX	Regions	16.64	4.99	**3.96**	**3.88**	**0.0**	**0.0**	**0.0**	**0.0**
		RCW	41.45	**32.59**	43.42	**34.56**	4.8	0.4	0.2	**0.0**
		CVRP15	132.64	107.33	**78.30**	86.53	63.0	49.2	**30.6**	35.8
Average			103.21	93.38	96.12	**91.26**	41.3	38.05	39.23	**36.41**
Average percentage improvement (%)				-13.64	-10.31	**-15.67**				

Table 12 Results on the training set for quality scenarios. Average quality from five runs

Algorithm	Dataset	Default	SMAC	PyDGGA	Selector
D-sat2	MAXSAT	3.43	**2.70**	2.92	**2.69**
ACOTSP	tsp-rue-1000-3000	3.982×10^7	$\mathbf{3.209}\times10^7$	3.248×10^7	3.219×10^7
ACOTSP-VAR	tsp-rue-3000	-0.876	-0.882	**-0.891**	**-0.890**
Average percentage improvement (%)			-13.79	-11.67	**-14.11**

Table 13 Percentage improvements for individual scenarios for AC methods on the test and training sets in percent

Algorithm	Scenario	Test sets			Training sets		
		SMAC	PyDGGA	Selector	SMAC	PyDGGA	Selector
CaDiCaL	Fuzz	-22.42	-2.46	**-26.32**	-23.47	5.22	**-32.87**
	BMC	5.30	18.94	**-8.14**	3.33	16.29	**-1.73**
	LABS	**-6.44**	0.40	-0.04	**-6.65**	-1.64	-0.29
	Limited	10.51	55.61	**-2.83**	**-2.99**	19.52	4.32
Glucose	Fuzz	-21.74	**-46.40**	-38.02	-34.52	**-40.02**	-30.73
	BMC	1.93	-2.52	**-3.99**	2.86	**-6.91**	-3.19
	Labs	-0.24	-4.26	**-4.69**	**-4.05**	-2.60	-2.74
	Limited	1.83	**-0.77**	1.74	-19.24	**-26.69**	-12.22
CPLEX	Regions	-68.37	-74.88	**-76.48**	-24.84	-40.36	**-41.56**
	RCW	**-20.03**	5.05	-17.33	**-21.37**	4.75	-16.62
	CVRP15	-18.73	**-38.23**	-32.85	-19.08	**-40.96**	-34.76
D-sat2	MAXSAT	**-21.03**	-13.54	**-21.03**	**-21.28**	-14.86	**-21.57**
ACOTSP	tsp-rue-1000-3000	**-19.62**	-18.54	-19.29	**-19.41**	-18.43	**-19.16**
ACOTSP-VAR	tsp-rue-3000	-1.02	**-1.47**	-1.25	-0.68	**-1.71**	**-1.59**

The percentage improvements in the individual scenarios on the training and test sets are illustrated in Table 13. A correlation between the performances on training and test sets is noticeable in the individual scenarios. However, there is no direct translation of the dominance of a method in a scenario from training to test set. While SELECTOR ties with SMAC in the number of dominated scenarios on the training sets, it outperforms SMAC on the test sets. This indicates that our ensemble-based method has a lower chance to overfit on the training set.

Acknowledgements The authors are supported in part by the project *Maschinelle Intelligenz für die Optimierung von Wertschöpfungsnetzwerken (MOVE)* (Grant No. 005-2001-0042) of the "it's OWL" funding of the Ministry of Economics, Innovation, Digitalization and Energy of the German state of North Rhine-Westphalia. This work was partially supported by the research training group "Dataninja" (Trustworthy AI for Seamless Problem Solving: Next Generation Intelligence Joins Robust Data Analysis) funded by the German federal state of North Rhine-Westphalia. The authors would also like to thank the Paderborn Center for Parallel Computation (PC^2) for the use of the Noctua clusters.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflicts of Interest/Competing Interests The research leading to these results received funding from the project *Maschinelle Intelligenz für die Optimierung von Wertschöpfungsnetzwerken (MOVE)* under Grant Agreement No. 005-2001-0042. This work was also partially supported by the research training group "Dataninja" (Trustworthy AI for Seamless Problem Solving: Next Generation Intelligence Joins Robust Data Analysis).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give

appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

6 Supplementary information

The implementation of the AC method in the version used for the experiments can be found in the commit 0fea3ba357a05ce3cd0b3f9390d71fa853c48015 on https://github.com/DOTBielefeld/selector.

References

Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., Tierney, K.: Model-based genetic algorithms for algorithm configuration. In: International Joint Conferences on Artificial Intelligence Organization (IJCAI) (2015)

Ansótegui, C., Pon, J., Sellmann, M., Tierney, K.: Reactive dialectic search portfolios for maxsat. In: AAAI Conference on Artificial Intelligence (2017)

Ansótegui, C., Pon, J., Sellmann, M., Tierney, K.: Pydgga: Distributed gga for automatic configuration. In: Li, C.-M., Manyà, F. (eds.) Theory and Applications of Satisfiability Testing – SAT 2021, pp. 11– 20 (2021)

Audemard, G., Simon, L.: Predicting learnt clauses quality in modern sat solvers. In: IJCAI International Joint Conference on Artificial Intelligence, pp. 399– 404 (2009)

Audemard, G., Simon, L.: Refining restarts strategies for sat and unsat. In: CP (2012)

Ansótegui, C., Sellmann, M., Shah, T., Tierney, K.: Learning to optimize black-box functions with extreme limits on the number of function evaluations. In: Simos, D.E., Pardalos, P.M., Kotsireas, I.S. (eds.) Learning and Intelligent Optimization, pp. 7–24. Springer, Cham (2021)

Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Principles and Practice of Constraint Programming, pp. 142–157 (2009). https://doi.org/10.1007/978-3-642-04244-7_14

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., Vanschoren, J.: Aslib: A benchmark library for algorithm selection. Artificial Intelligence 237, 41–58 (2016) https://doi.org/10.1016/j.artint.2016.04.003

Cowen-Rivers, A., Lyu, W., Wang, Z., Tutunov, R., Jianye, H., Wang, J., Ammar, H.: HEBO: heteroscedastic evolutionary bayesian optimisation. CoRR arXiv: abs/2012.03826 (2020)

Di Liberto, G., Kadioglu, S., Leo, K., Malitsky, Y.: Dash: Dynamic approach for switching heuristics. Eur. J. Oper. Res. **248**(3), 943–953 (2016)

El Mesaoudi-Paul, A., Weiß, D., Bengs, V., Hüllermeier, E., Tierney, K.: Pool-based realtime algorithm configuration: A preselection bandit approach. In: Kotsireas, I.S., Pardalos, P.M. (eds.) Learning and Intelligent Optimization, pp. 216– 232 (2020)

Escamocher, G., O'Sullivan, B., Prestwich, S.D.: Generating difficult sat instances by preventing triangles. arXiv: abs/1903.03592 (2019)

Fitzgerald, T., Malitsky, Y., O'Sullivan, B.: Reactr: Realtime algorithm configuration through tournament rankings. In: International Joint Conferences on Artificial Intelligence Organization (IJCAI), pp. 304–310 (2015)

Fitzgerald, T., Malitsky, Y., O'Sullivan, B.J., Tierney, K.: React: Real-time algorithm configuration through tournaments. In: Annual Symposium on Combinatorial Search (SoCS) (2014)

Han, J.M.: Enhancing SAT solvers with glue variable predictions. CoRR **abs/2007.02.559** (2020)

Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent Optimization (LION), pp. 507–523 (2011)

Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research (JAIR), 267–306 (2009)

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H., Leyton-Brown, K., Stützle, T.: Aclib: A benchmark library for algorithm configuration. In: International Conference on Learning and Intelligent Optimization (LION), pp. 36–40 (2014). https://doi.org/10.1007/978-3-319-09584-4_4

IBM: IBM ILOG CPLEX Optimization Studio: CPLEX User's Manual. (2016). https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.studio.help/pdf/usrcplex.pdf

Indu, M.T., Velayutham, C.S.: Differential evolution ensemble designer. Expert Systems with Applications 238, 121674 (2024) https://doi.org/10.1016/j.eswa.2023.121674

Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: Survey and perspectives. Evol. Comput. **27**(1), 3–45 (2019)

Kemminer, R., Lange, J., Kempkes, J., Tierney, K., Weiß, D.: Configuring mixed-integer programming solvers for large-scale instances. Operations Research Forum **5** (2024) https://doi.org/10.1007/s43069-024-00327-7

Kool, W., Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=ByxBFsRqYm

Lindauer, M.T., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Sass, R., Hutter, F.: Smac3: A versatile bayesian optimization package for hyperparameter optimization. J. Mach. Learn. Res. **23**, 54–1549 (2022)

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

López-Ibáñez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. Eur. J. Oper. Res. **235**(3), 569–582 (2014)

OPTANO: OPTANO Algorithm Tuner Documentation (2022). https://docs.optano.com/algorithm.tuner/current/ Accessed 2022-11-08

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

Park, H.-S., Jun, C.-H.: A simple and fast algorithm for k-medoids clustering. Expert Systems with Applications 36(2, Part 2), 3336–3341 (2009) https://doi.org/10.1016/j.eswa.2008.01.039

Polikar, R., Polikar, r.: Ensemble based systems in decision making. ieee circuit syst. mag. 6, 21-45. Circuits and Systems Magazine, IEEE 6, 21–45 (2006) https://doi.org/10.1109/MCAS.2006.1688199

Schede, E., Brandt, J., Tornede, A., Wever, M., Bengs, V., Hüllermeier, E., Tierney, K.: A survey of methods for automated algorithm configuration. Journal of Artificial Intelligence Research 75, 425–487 (2022) https://doi.org/10.1613/jair.1.13676

Scott, J., Niemetz, A., Preiner, M., Nejati, S., Ganesh, V.: Machsmt: A machine learning-based algorithm selector for SMT solvers. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12652, pp. 303–325. Springer, (2021). https://doi.org/10.1007/978-3-030-72013-1_16

Stuetzle, T.: ACOTSP, Version 1.0. http://www.aco-metaheuristic.org/aco-code. Accessed: 2023-12-29 (2004)

Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., Guyon, I.: Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In: Escalante, H.J., Hofmann, K. (eds.) Proceedings of the NeurIPS 2020 Competition and Demonstration Track. Proceedings of Machine Learning Research, vol. 133, pp. 3–26 (2021). https://proceedings.mlr.press/v133/turner21a.html

Tornede, A., Gehring, L., Tornede, T., Wever, M., Hüllermeier, E.: Algorithm selection on a meta level. Mach. Learn. **112**(4), 1253–1286 (2023)

Timgates42: scikit-optimize (2020)

Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997). https://doi.org/10.1109/4235.585893

Weiss, D., Tierney, K.: Realtime gray-box algorithm configuration. In: Simos, D.E., Rasskazova, V.A., Archetti, F., Kotsireas, I.S., Pardalos, P.M. (eds.) Learning and Intelligent Optimization, pp. 162–177 (2022)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.