Universiteit
Leiden
The Netherlands

Sampling strategies in automated algorithm configuration

Anastacio, M.I.A.

Citation

Anastacio, M. I. A. (2026, June 23). *Sampling strategies in automated algorithm configuration*. Retrieved from https://hdl.handle.net/1887/4307419

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4307419

Note: To cite this publication please use the final published version (if applicable).

3

Critical Assessment of the State of the Art in Automated Algorithm Configuration

As seen in the previous chapter, the automated algorithm configuration (AAC) literature contains many different scenarios. Each time a new configurator is introduced, it is evaluated on its own set of old and new scenarios, sometimes completely disjoint from each others, which greatly complicates comparisons between their performance based on their respective publications. Moreover, different publications tend to follow a different setup of experiments and run the algorithms on different machine execution environments which, for running time optimisation tasks, makes the comparison between results from several papers nearly impossible. In this chapter, we evaluate a diverse set of state-of-the-art configurators on the scenarios described in the previous chapter and present the first such evaluation for running time optimisation AAC scenarios. We present the obtained data and analyse them in light of the characteristics described in the previous chapter. To facilitate further research, we make all performance data openly available.

3.1 Background

As described in Chapter 2, there have been many attempts to build configurators, each focusing on specific challenges ofthe AAC problem and providing more insights on it. However, due to the variation in focus of the community, the continuous development of new solvers and the evolution of hardware, they were often tested on different datasets and benchmarks. An attempt at proposing a unified library of configuration scenarios has been made by Hutter *et al.* with AClib (Hutter et al., 2014a) but due to the extensive computation time typically required by configurators, researchers will typically limit the scope of their experiments to a subset of those scenarios. To the best of our knowledge, there has been no attempt in extensively comparing the current prominent configurators on a large set of scenarios nor to draw clear conclusions about their strength and weaknesses. This is the aim of the present chapter. To answer RQ2 – *How do state-of-the-art configurators compare to each others on a various set of scenarios?* – we run state of the art configurators on a large set of scenarios and compare their expected performance at the end of the given configuration budget. Then, we use the scenario features defined in Chapter 2 to answer to RQ3 – *How do scenario characteristics influence the performance of state-of-the-art configurators?* – and attempt to draw general conclusions about the strength and weaknesses of each configurator.

3.2 Evaluating the state of the art

Existing configurators, introduced in the previous chapter, have been developped and tested on different kind of target algorithms and with different applications in mind. Many configurators come with specific sets of features that reflect the requirements of the intended application. When performing an evaluation such as the one conducted here, it is hard to account for those differences, and for practical purposes, we reduced the considered options to those shared among most configurators and not requiring extensive pre-processing by end users (such as organising instances by families, deciding in which order they should be included in the search, or coding complex wrappers for the target algorithm). Moreover, we are aware that there is potential in tuning the hyper-parameters of the configurators, but this would involve a large computational cost. Thus, we are applying all configurators out of the box, following the developper's recommendations on their use. In the remainder of this section, we introduce the choices we made regarding the evaluation protocol and how we handled the level of

parallelisation of the configurators as well as their limitations in term of search space.

3.2.1 Evaluation protocol

To decide how we should evaluate a configurator, we have to go back to what we aim at measuring. As we configure, the aim of the configurator is to produce a configuration that will best generalise to other instances from the same underlying distribution. Since the configurators are not deterministic, the standard protocol is to run the configuration process several times on a training set and keep the configuration that performs best on that set (see the recommendations of Eggensperger et al., 2019). Because in many cases only a subset of instances is seen during the configuration process, it is common practice to validate on the training set rather than on a separate validation set (Eggensperger et al., 2019). To evaluate the ability to generalise, we can then look at the performance on a test set. In a case were the goal would be to improve the algorithm performance or analyse its best performing configurations, one would then compare the performance of the default configuration, or an expert-chosen set of parameter values, and the configured algorithm (*e.g.* Fokkinga et al., 2019).

When evaluating the configurator, we want to know how likely it is that the target algorithm performs better once configured than using the default configuration. Thus, we are interested in the distribution of possible configurations that this procedure would output. To do so, we would need to perform the above described procedure several times. To simulate this, we configure the target algorithm N times to create a pool of best found configurations, sample $n < N$ configurations that represent the set of configurator runs performed by a practitioner, and keep the best performing configuration on the training set out of those n configurations.

Definition 3.1. We call *standard protocol* the following protocol:

1. run $N \in \mathbb{N}$ times the configurator to create a set of N configurations
2. evaluate the performance of those configurations on the training set
3. uniformly sample $n < N$ configurations
4. keep the configuration with the best performance among the n samples
5. go back to step 3 until you gather the desired number of configurations

This protocol allows us to gather a distribution of *expected best configurations*. Rather than running steps 3 to 5 of the standard protocol, we can calculate the probability of each configuration to be chosen.

Theorem 3.1. Given a list of N configurations ω_i with $i \in [0, N-1]$ in ascending order based on their performance on the training instances, *i.e.*

$$\forall j \in [1, N-1], \mathcal{M}(\omega_{j-1}, \mathcal{I}_{\text{train}}) < \mathcal{M}(\omega_j, \mathcal{I}_{\text{train}}),$$

with $\mathcal{I}_{\text{train}}$ the training instances set and $\mathcal{M}(\omega_i, \mathcal{I}_{\text{train}})$ the performance of the configuration ω_i on the set of instances $\mathcal{I}_{\text{train}}$ that we try to minimise.

The probability for ω_i to be sampled through the standard protocol can be expressed as follows:

$$P(\omega_i) = \frac{n}{N-i} \times \frac{\binom{N-i}{n}}{\binom{N}{n}}$$

Proof. The total number of sets $\mathcal{S}$ of n configurations is $\binom{N}{n}$.
There are i configurations performing better than ω_i, thus the number of sets $\mathcal{S}$ such that $\forall \omega \in \mathcal{S}, \mathcal{M}(\omega_i, \mathcal{I}_{\text{train}}) \leq \mathcal{M}(\omega, \mathcal{I}_{\text{train}})$ is $\binom{N-i}{n}$. Thus, the probability that $\forall \omega \in \mathcal{S}, \mathcal{M}(\omega_i, \mathcal{I}_{\text{train}}) \leq \mathcal{M}(\omega, \mathcal{I}_{\text{train}})$ is $\frac{\binom{N-i}{n}}{\binom{N}{n}}$. Finally, the probability that $\omega_i \in \mathcal{S}$ is $\frac{n}{N-i}$.
All together, the probability of ω_i being the best performing sample in $\mathcal{S}$ is

$$\frac{n}{N-i} \times \frac{\binom{N-i}{n}}{\binom{N}{n}}.$$

□

We note that, if ω_0 is sampled it will always be outputed, its probability in the final distribution is thus $\frac{n}{N}$. On the other opposite, the $n-1$ last configurations will never be outputed as there will always be one better configuration chosen.

In the following, when comparing configurators, we will always consider the distribution of performance values for the configurations obtained by means of the standard protocol.

3.2.2 Configurators specificities

Despite our intention to consider all configurators on equal ground, there are difference in the way they handle the scenarios, in the variety of scenarios they can consider and on the way they use the resources given to them. Those specificities and their expected impact are listed below.

Parallelisation

Among the considered configurators (listed in Section 2.2.2), GGA++ and GPS are the only ones which need parallelisation to work as intended. The typical case for both would be to launch 8 workers according to discussion with the developpers and the paper which introduced GPS (Pushak and Hoos, 2020). On the other hand, while SMAC and irace support parallelisation, it is not core to their usage and they would typically not be used in that fashion. To allow us to compare parallel algorithms to purely sequential algorithms, we follow an approach similar to the one used by Pushak and Hoos in the paper introducing GPS.

For each configurator and each scenario, we run the sequential configurators 24 times and apply the standard protocol with a sample size of 8, while we run the parallel configurators 8 times with 8 workers and apply bootstrap sampling to obtain a larger sample size. These two approaches allow us to obtain the same number of samples for each configurator while avoiding the large computational incurred by running the parallel configurators as many times as the sequential ones. Moreover, Eggensperger et al. (2019) recommends using parallel runs of sequential configurators. By comparing 8 parallel runs to one run on 8 cores, we simulate a situation where a practitioner has access to one machine with 8 cores.

Search spaces

Configurators have limitations regarding the search space they are able to handle. paramILS, for example, can only handle discrete parameter space, which means that we made a discrete version of the search spaces. To generate this discrete search space, we take 10 evenly spaced values and add the default value to this set if it is not part of the chosen values. Since EAX has only 2 hyperparameters, which are both integers, we did not need to modify its search space.

An other limitation in search space is brought in by the way it is described. GGA++ and GPS search spaces are presented in a tree-like way, which limits the ability to handle conditionals. In particular, if one parameter depends on more than one other parameters, the user needs to duplicate it and create dummies that the configurator will not consider as linked. Since this requires a non-negligible amount of work, those were run only on solvers which do not have this particular feature. In particular, Clasp and SpToRiss have many conditionals and have been excluded. LPG had three such parameters, which made the required changes easy to implement, so we kept it.

Configuration budget

While we consider the budget given to a configurator in terms of wall-clock time, it is not possible to give it as-is to all configurators. In particular, irace and GGA++ were not developped with a concept of wall-clock time as a budget. By design, the budget of irace is expressed as a number of iterations. For compatibility, it includes a mechanism that allows to estimate the running time of the target algorithm and through it the number of iteration which can be done within the given configuration budget. This mechanism sometimes leads to irace running overtime or to it refusing to run if the given budget is found to be too short. GGA++, on the other hand, needs to be configured in terms of size of population and number of generation for the genetic algorithm. OPTANO, the freely available implementation of GGA++ we are using, provides a tool which allows to estimate how long the configuration run will take based on a few of its parameters. Based on the 48 hours budget of most of our scenarios (see Table 2.4) and our 8 workers, this tool led us to use a population of 100 configurations over 100 generations.

For both configurators, we stop after the given wall-clock time and take the last incumbent found.

3.2.3 Normalised score

For some of our analysis, we want to be able to aggregate or put next to each others performance data collected over several scenarios. Because there is a large variation in the running time of the respective solvers, we normalise the performance of the solvers using the default value and the best final performance as reference points.

For any given $\omega \in \Omega$, the normalised performance $\mathcal{M}'$ is thus defined as:

$$\mathcal{M}'_K(\omega) = \frac{\mathcal{M}(\omega_{\text{def}}, \mathcal{I}) - \mathcal{M}(\omega, \mathcal{I})}{\mathcal{M}(\omega_{\text{def}}, \mathcal{I}) - \mathcal{M}(\omega^*_K, \mathcal{I})}$$

where ω^*_K denotes the best known configuration for a scenario K. Since we do not know the best configuration overall, we evaluate all obtained configurations after the full configuration time and take the best performing configuration on the test set as an estimate for the lower bound.

3.2.4 Implementation details

Eggensperger et al. (2019) listed avoidable pitfalls and best practices in algorithm configuration. To avoid inconsistencies in the evaluation of the target algorithm running

time, we follow AClib in using a standard wrapper based on the `runsolver` software to evaluate the running time of each run in a similar way. Moreover, we make sure to move all instance files into the RAM of the compute node on which the target algorithm is run to avoid the speed of the file system to impact the time required to read the instances from external memory (such as a hard drive). To decrease latency, the code and executables are all stored on a `BeeGFS` file system. We perform all experiments on a computing cluster running Rocky Linux 9.3. Each node is equipped with 2 AMD EPYC 7543 32-core CPUs with 256 MB L3 cache and has 1TB of memory.

3.3 Evaluation results

This section shows the results of the experiments described in Section 3.2.1. All numbers are based on the expected performance described there. We first show the results and then draw high-level conclusions about the evaluated configurators. All our results are available on `ada.liacs.nl/ac-comparison`.

3.3.1 Overall performance

Table 3.1 summarises the results of our experiments comparing configurators performance. It shows the median of the expected performance following the protocol described in Section 3.2.1 with the configurations found at the end of the configuration budget (as specified in Table 2.4). The lowest median values are underlined. To evaluate the statistical significance of the difference between the configurator reaching the lowest median and the others, we applied a Mann-Whitney U-test ($\alpha = 0.05$). We tested both the significance of the difference in performance on the raw distribution of performances from and on the distribution of best configurations obtained through our full protocol. Since our protocol resamples the distribution with a bias towards the lower tail, it enlarges the differences between them and the test indicated significant differences in all cases. The result of the test applied to the raw distributions is shown in the Table 3.1 by putting in bold values for which the underlying raw distribution is statistically tied with the best one.

As could be expected, our random baseline ROAR can not find a better configuration than the default one in most scenarios. It improves on it in 9 of our 20 scenarios, meaning that in more than half of our scenarios, running random configurations for two days failed to achieve improvements over the default configurations. This is consistent with the premise that configuration scenarios are difficult and reinforces the

need for more sophisticated search algorithms. We note, however, that for SpToRiss on CF ROAR outperforms more sophisticated approaches.

Looking at the difference between SMAC2 and SMAC3 shows a clear advantage for the former. On all of our mixed integer programming (MIP) and traveling salesperson problem (TSP) scenarios, as well as on more than half of our Boolean satisfiability (SAT) scenarios, SMAC2 performs better than SMAC3. While SMAC3 is a Python reimplementation of SMAC2, some changes that might affect its performance have been introduced. In discussions with the authors, we discovered that the procedure for introducing random configuration among the challenger configurations tested against the incumbent was changed, because the one used in SMAC2 was too exploratory for hyperparameter optimisation of machine learning models. Instead of alternating between a random one and one based on the surrogate model, like the earlier version, they use a random configuration with a set probability. Compared to other configurators, SMAC2 shows very strong performance.

Despite being model-free and configuring on a discrete space, paramILS is among the best-performing configurators. One could expect that the best performing configurations might be excluded from the search space in many cases due to discretisation, but it seems that the reduction in search space size often compensates for this. It would be interesting to evaluate if running the other configurators on this same discrete space would improve their performance.

The performance of irace varies widely across target algorithms. Notably, it is among the best on all SpToRiss scenarios but performs very poorly on all Lingeling scenarios. While it has been argued in the past that the high cutoff time of the CPLEX scenarios hinders the performance of irace (Cáceres et al., 2017), we do not observe a large drawback. We note, however, that the CPUs on which our experiments are running have a significantly higher performance. This leads to most configurations to complete runs within the cutoff time. Thus, larger cutoff time does not make much difference in the number of instances finishing in time. The results reported in Chapter 4, which ran on a different machine, point in that same direction.

GGA++ (Ansótegui et al., 2015) performs unevenly. The output on both TSP scenarios, which have a low configuration time, is especially problematic since it is worse than the default; On the other hand, it achieves the best performance on both LPG scenarios and among the best on several others (*e.g.* Kissat on CF, Lingeling on UNSAT).

GPS is the only configurator that never ranks first. On many scenarios, it struggles to do better than the default. We note that the fact GPS runs a database to keep track

Table 3.1: Median expected performance of configurators on continuous search space, the best values are underlined and the values statistically tied with the best are boldfaced

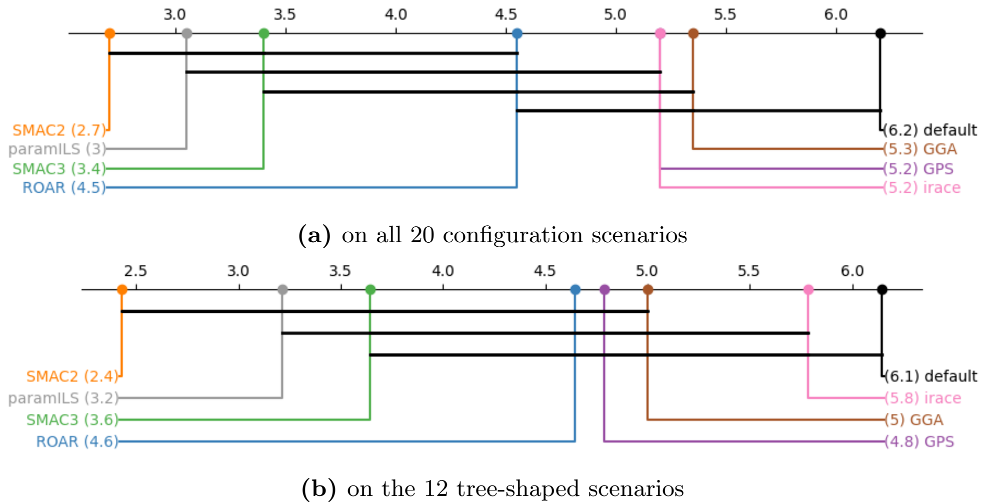
scenario	default	ROAR	SMAC2	SMAC3	irace	GGA++	GPS	paramILS
Clasp								
CF	103.29	103.29	102.73	**102.01**	**112.72**	X	X	110.81
LABS	707.25	707.25	**743.58**	699.40	**734.13**	X	X	**685.97**
UNSAT	0.42	0.19	0.18	0.18	0.18	X	X	**0.18**
Kissat								
CF	110.91	**110.91**	92.80	100.83	**140.48**	**90.57**	**100.85**	**89.83**
LABS	667.63	667.63	**691.16**	**676.01**	725.13	708.86	667.72	**666.69**
UNSAT	0.62	0.48	0.19	0.28	0.23	0.31	0.58	**0.19**
Lingeling								
CF	215.53	215.53	**165.12**	184.03	406.30	206.67	**224.52**	171.19
LABS	796.01	**796.01**	**780.05**	**795.42**	863.15	813.84	787.54	796.29
UNSAT	1.22	1.22	0.66	0.79	1.65	0.58	0.79	**0.55**
SpToRiss								
CF	326.40	**145.82**	**162.55**	**182.24**	151.37	X	X	183.35
LABS	811.09	747.13	751.49	752.49	744.46	X	X	**737.89**
UNSAT	151.42	0.97	0.84	0.88	**0.79**	X	X	0.83
CPLEX								
CLS	1.71	1.89	**1.24**	1.28	2.15	1.54	1.88	**1.07**
COR-LAT	10.95	10.47	**2.84**	9.46	5.06	18.96	10.47	10.65
RCW2	38.71	38.71	**25.94**	33.70	44.95	38.71	35.85	30.60
REG200	6.32	3.31	1.93	2.68	2.40	5.48	3.94	**1.73**
LPG								
Satellite	8.03	1.93	2.35	1.91	2.45	**1.90**	8.02	2.49
Zenotravel	12.56	**1.24**	1.06	**1.13**	0.90	**0.82**	12.52	**1.29**
EAX								
rue-1000-3000	120.82	**81.53**	**75.27**	**84.59**	**103.17**	**763.49**	**75.59**	**99.33**
LKH								
rue-1000-3000	229.22	**229.22**	**139.08**	**180.65**	227.97	586.82	**194.72**	**213.91**

of the performed runs might have affected our runs as a result of slow communication compared to the read and write speed in RAM for other configurators.

3.3.2 Comparative analysis

While the previous section explored the results in details, we now take a step back and look at the bigger picture. To do so, we first look at the widely used critical difference plot to determine if one of the configurators performs significantly better in terms of ranking over all of our datasets. Then, because aggregate performance over a broad range of scenarios is not always the desired goal, we evaluate the contribution each configurator would make to a configurator portfolio if we were to build one.

Since two of our configurators (GGA++ and GPS) could not run on all of the scenarios, we show our analysis separately for those compatible scenarios, named *tree-shaped scenarios* in the following, since they apply to target algorithms with a



(a) on all 20 configuration scenarios

(b) on the 12 tree-shaped scenarios

Figure 3.1: Critical difference diagram of the average score ranks of the configurators. Configurators linked with a black line were not found statistically different.

parameter space that can be represented as a tree.

Note that in this section, we base our evaluation on the average running time without penalty on timeouts rather than the penalised average running time. The factor applied could impact the soundness of the results of the statistical tests.

Critical difference diagram

In the machine learning community, a common approach for evaluating results across multiple datasets is the critical difference plot. Following the work of Demšar (2006), we first applied a non-parametric Friedman test with a significance threshold of 0.05 on our expected performances to verify that they come from different distributions. The test rejected the null hypothesis that the performance of the configurators is similar. We then applied a post hoc Nemenyi test with a significance threshold of 0.05 that compares each pair of configurators. The null hypothesis is that the two configurators have the same performances. We visualise the result in Figure 3.1, where the horizontal axis shows the average ranking of the configurator and pairs on which the Nemenyi test could not reject the null hypothesis are linked with a black line.

When looking at all scenarios in Figure 3.1a, we see that while all approaches achieve improvements over the default configurations, the differences are not statistically significant for ROAR, irace, GPS, and GGA++. This goes against the expecta-

tion that configurators consistently achieve improvements over the default configurations.

As mentioned in Section 3.3.1, the performance of paramILS is particularly strong considering that it is model-free and limited to searching a discrete space of parameter configurations. However, while it ranks first more often than SMAC2, looking at the average ranking over all scenarios gives SMAC2 an edge, albeit not a statistically significant one. ROAR, our random baseline with racing, performs fairly well and ends up tied with the best configurators: SMAC2, paramILS and SMAC3.

GGA++, irace and GPS perform similarly, with a slightly lower rank for GGA++ than for the other two. When looking at Figure 3.1b, we see that for tree-shaped scenarios, they are close to the performance of ROAR and join the statistically tied group of best-performing configurators. Interestingly, despite never placing first, GPS still achieves a better mean rank than GGA++ and irace.

Each configurator is expected to have strengths and weaknesses depending on the type of scenarios considered. Since we chose the scenarios in such a way that they cover a wide range of domains and characteristics, the fact that none of the configurators largely outperforms all the others in a statistically significant way aligns with our expectations.

Contribution to a portfolio

When evaluating the state of the art of NP-hard problems solvers, the idea of studying the contribution each solver makes to the state of the art, their *marginal contribution*, was first discussed by Xu et al. (2012) in the context of SAT solvers. To quantify the contribution of a solver, they compared the performance of portfolio techniques, such as automated selection or parallel portfolios, built with this solver to ones built without this solver. However, this approach suffers from several drawbacks, as pointed out by Fréchette et al. (2016). In particular, if two solvers solve the same set of instances, they would both have the same marginal contribution as a solver that solves no instance at all. Indeed, their marginal contribution compared to one another could be null.Removing both solvers from the portfolio would, however, degrade the performance of the portfolio. Thus, Fréchette et al. (2016) compute the contribution of the configurators with the Shapley value. For our application, the scores they used need to be slightly modified. In their case, they wanted to account more for the ability of a solver to solve an instance and thus computed a score that allowed them to give precedence to this objective. In our case, we are interested in the performance improvement achieved through automated configuration of a given target algorithm

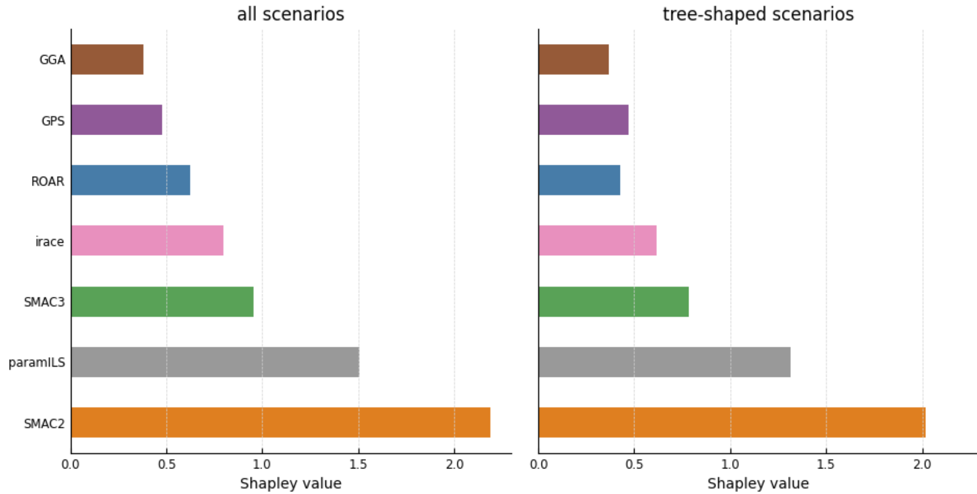


Figure 3.2: Shapley values of the configurators in an oracle portfolio

compared to using it with default parameter values. Thus, we use the normalised performance described in Section 3.2.3 and subtract it from 1 to obtain an objective function to be maximised.

Considering a set of configurators $\mathcal{C}$. The Shapley value $\phi(C)$ of a configurator $C \in \mathcal{C}$ is computed as follows:

$$\phi(C) = \frac{1}{|\mathcal{C}|} \cdot \sum_{G \subseteq \mathcal{C} \setminus \{C\}} \frac{v(G \cup \{C\}) - v(G)}{\binom{|\mathcal{C}-1|}{|G|}}$$

, where v returns the score of the best configurator for a given configuration scenario among the given set.For our set of scenarios $\mathcal{K}$,

$$v(G) = \sum_{K \in \mathcal{K}} \max_{C \in G} score_K(C) \text{ with } score_K(C) = 1 - \mathcal{Q}'_K(C)$$

, where $\mathcal{Q}'_K(C)$ denotes the median of the performance values of configurator C, normalised as per Section 3.2.3.

The obtained contributions are shown in Figure 3.2.

Consistent with the results shown in Section 3.3.2, SMAC2 achieves the highest contribution, followed by paramILS and SMAC3. However, despite its low overall ranking, irace makes a higher contribution than GGA++ and GPS, indicating that it complements the top three configurators. When only considering the tree-shaped

scenarios, GGA++ and GPS reach a higher contribution, but still fall short of irace.

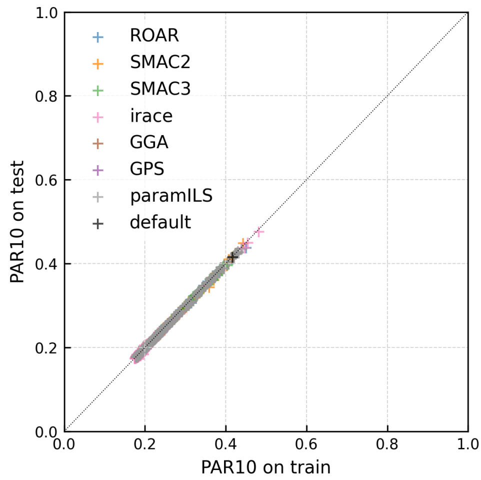
3.3.3 Overtuning analysis

A known problem of AAC is the difficulty to find configurations that generalise well to previously unseen instances, as shown by Eggensperger et al. (2019). To evaluate how much the configurators we studied are impacted by this difficulty, we traced back all configurations evaluated and stored throughout the configuration run and compare their performance on the training and testing sets. This allows us to visualise the overtuning behaviour described in prior work (see *e.g.* Schneider et al., 2025; Eggensperger et al., 2019). The plots thus obtained are shown in Figure 3.3.

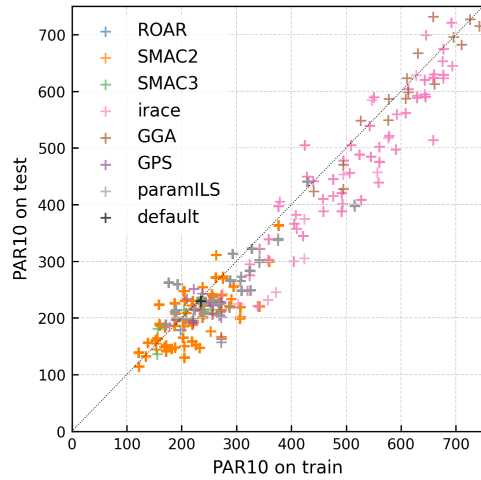
Figure 3.3 shows scenarios with the four types of behaviours we observed. Clasp on UNSAT (Figure 3.3a) is a case in which there is no sign of overtuning. The performance improvement on the training set correlates with the improvement on the testing set. Figure 3.3b shows an example for which the correlation is weaker, but still clearly visible. This type of behaviour is found more often than the previous one on the UNSAT dataset.On the other hand, CPLEX on CLS (Figure 3.3c) shows clear signs of overtuning from SMAC2, which kept many configurations with low PAR10 values on the training set and high PAR10 values on the testing set. Finally, Figure 3.3d shows a scenario for which the performance on the training and testing sets show lower correlation, though the PAR10 values still loosely follow the diagonal. This can indicate that the training and testing sets are related in a different way than the linear correlation detected with the Pearson correlation we used. Except for the outlier case on SMAC2 shown in Figure 3.3c, all configurators show a high correlation between the performances on the training and testing sets for our scenarios, from 0.6 on scenarios such as SpToRiss on LABS (Figure 3.3d) to above 0.9 for cases scenarios such as Clasp on UNSAT (Figure 3.3a)

3.3.4 Scenario features analysis

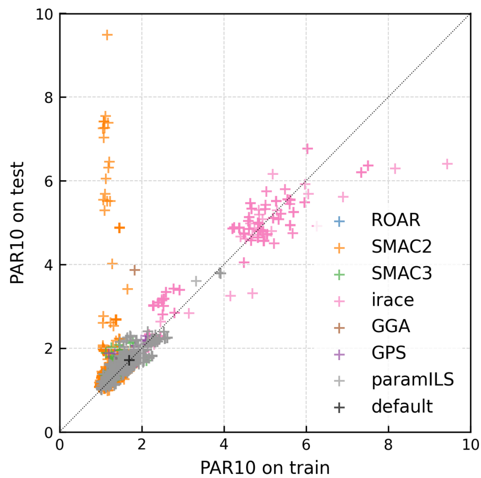
Finally, we investigate the impact of the features defined in Chapter 2 on the performance of the configuration approaches. To do so, we first compute the correlation between the features and normalised configuration performance. Then, we use the four CPLEX scenarios to investigate the impact of the choice of cutoff time for the target algorithm, since those scenarios have been used in the literature with two different cutoffs.



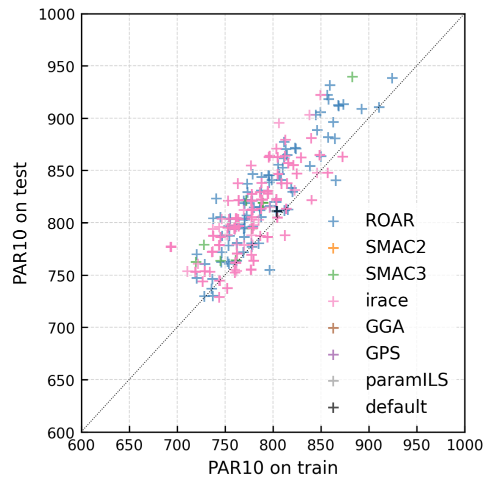
(a) Clasp on UNSAT



(b) LKH on rue-1000-3000



(c) CPLEX on CLS



(d) SpToRiss on LABS

Figure 3.3: PAR10 values on the train and test set for a subset of our scenarios

Table 3.2: Correlation between the normalised performance of the configurators and the scenario features

	ROAR	SMAC2	SMAC3	irace	GGA++	GPS	paramILS
Origin	0.21	-0.35	0.14	0.04	0.04	-0.07	0.12
Training size	-0.57	-0.43	-0.46	-0.52	-0.33	0.04	-0.34
Testing size	-0.56	-0.41	-0.44	-0.51	-0.23	0.01	-0.29
Features	-0.50	-0.40	-0.50	-0.38	-0.46	0.22	-0.42
Clusters	0.03	-0.07	0.14	-0.03	0.04	0.04	0.21
Deterministic	0.22	-0.42	-0.08	0.06	0.00	-0.10	-0.19
Total	0.16	0.22	0.16	0.34	-0.38	0.02	0.03
Categoricals	-0.05	0.10	0.02	0.10	-0.37	0.13	-0.06
Integers	0.37	0.31	0.29	0.52	-0.27	-0.12	0.13
Continuous	-0.58	-0.30	-0.42	-0.52	-0.29	0.35	-0.31
Conditionals	-0.36	0.06	-0.11	-0.32	-0.13	0.29	-0.03
Forbidden	-0.55	-0.11	-0.31	-0.44	-0.21	0.29	-0.19
Cutoff	0.21	-0.42	-0.08	0.06	-0.01	-0.09	-0.19
Timeouts	0.39	0.75	0.66	0.25	0.30	0.09	0.70
Budget	-0.07	-0.09	-0.12	-0.05	-0.82	0.36	-0.26

Correlation analysis

As a first high-level analysis regarding the impact of scenario features on the performance of the configurators, we look at the correlation between those features and the normalised performance of the configurators. We show those correlations in Table 3.2. We note that the normalisation depends on the performance of other configurators.

We observed that the size of the training and testing sets are negatively correlated with the performance of most configurators: a larger number of instances corresponds to a better performance (since lower is better). We notice that the overall number of parameters does not seem to impact the performance much, while the type of those parameters impacts the outcome. Continuous parameters seem easier to configure that integers and categoricals, except for GPS. Regarding timeouts, the more there are, the worse performance we reach for all but GPS. This aligns with expectations, since more timeouts means less information for the model to learn from and more time spent on running on those hard instances. Methods for spending less time on those challenging instances will be further developped in Chapters 5 and 6.

Table 3.3: Median expected performance of configurators on CPLEX scenarios, the best value for each configurator on a scenario is underlined

configurator	CLS		COR-LAT		RCW2		REG200	
	long	short	long	short	long	short	long	short
default	1.72		23.12		115.97		6.13	
ROAR	1.75	1.72	22.44	23.12	116.10	115.97	3.31	3.22
SMAC2	1.24	1.21	3.02	3.64	52.05	57.06	1.93	1.91
SMAC3	1.44	1.55	18.98	22.27	89.92	94.17	2.68	2.92
irace	2.15	1.57	7.74	6.46	149.58	120.60	2.40	2.25
GGA++	1.54	1.60	41.17	3.70	115.97	116.17	5.48	2.08
GPS	1.88	1.83	22.08	12.52	113.81	114.45	3.94	3.95
paramILS	1.19	2.92	22.43	8.62	86.11	83.35	1.73	1.75

Impact of the cutoff time

As previously discussed, the number of timeouts impacts the performance of the configurator. A characteristic related to the number of timeouts is the cutoff time, the maximum time given to the target algorithm before it is terminated. To investigate the impact of this characteristic, we ran the CPLEX scenarios with both the AClib based cutoff time and the one used by Cáceres et al. (2017). We compared the results of the configurators on those scenarios in Table 3.3. We note that the reduction of cutoff time did not impact the number of timeouts for CLS and REG200, while it brought it from 0.1 to 0.3 percent on COR-LAT and from 0 to 3 percent on RCW2.

Overall, we observed that the use of a shorter cutoff time does not have a clear impact, even when it leads to a higher number of timed out runs. Looking at the results per configurator, we observe that a shorter cutoff consistently benefits irace, while a longer cutoff time consistently benefits SMAC3.

3.4 Conclusion

In this chapter, we evaluated state-of-the-art configurators on a diverse set of scenarios for running time optimisation as introduced in Chapter 2. We showed that configurators typically find configurations better performing than the default value, when given the time to perform 500 or more target algorithm run. However, they do not always surpass ROAR, a simple random search with a racing mechanism. We found that on the configuration scenarios we considered paramILS was best on the most scenarios, while SMAC2 had the best average ranking. Though these advantages

are not statistically significant over all other configurators. Overall, we found that configurators show complementary strength and weaknesses. In particular, while irace did not perform strongly overall, it showed a significant contribution to a portfolio. This complementarity was already visible in the results, for example when configuring SpToRiss.

We analysed to what extend the performance of the best found configurations differ on the testing set to their performance on the training set to detect if the configurators are prone to overtuning, (*e.g.*finding a well performing configuration on the training set that does not generalise to the testing set). We did not find any strong evidence of it occurring on the scenarios we studied, with only one clear case with SMAC2 when optimising CPLEX for CLS.

Finally, we made an attempt at drawing high-level insights regarding the impact of scenario specific features such as the ones defined in Chapter 2. We found that the type of the parameters have more impact on the difficulty of a configuration scenario than their overall number, that scenarios with more instances to train on are typically easier and that the performance of GGA++ is strongly correlated with the configuration budget. Though we note that our scenarios did not include many variations of configuration budget and this correlation might be related to other elements of the small budget scenarios.

While we performed various analysis on the collected data, we believe that more can be done, such as defining new scenarios features to analyse or trying to predict which configurator should be used on a new scenario, and thus made it openly available for further exploration.