Universiteit Leiden
The Netherlands

Sampling strategies in automated algorithm configuration

Anastacio, M.I.A.

4

Default Value

General-purpose automated algorithm configuration procedures have enabled significant improvements in the state of the art for a wide range of challenging problems. This increasingly encourages algorithm designers to expose more parameters to the configuration process, leading to larger configuration spaces. To search these vast combinatorial spaces of parameter settings is challenging. Successful configurators combine techniques such as racing, estimation of distribution algorithms, Bayesian optimisation and model-free stochastic search (see Section 2.2.2). However, most methods disregard the default parameter values typically provided by the developers or merely use them as a starting point for the configuration process. This chapter[1] explores how and to what extent these default parameter values can be used as prior knowledge to guide the configuration search. First, we propose a simple method to reduce the size of the search space and restrict the configurator to parameter values close to a given default. Based on the encouraging results of this naïve approach, we introduce a principled method for integrating the default value into the configuration process by focusing the search around it.

[1] Parts of this chapter have been published as Anastacio et al. (2019); Anastacio and Hoos (2020a,b).

4.1 Introduction

The availability of effective automated algorithm configuration (AAC) procedures allows algorithm designers to expose design choices as parameters, following a design paradigm known as programming by optimisation (PbO) (Hoos, 2012b). By avoiding premature choices in algorithm design, automated methods can adapt the behaviour of algorithms to each specific use case, thereby achieving better performance. However, the size of the combinatorial configuration spaces encountered in this context grows exponentially with the number of exposed parameters. In the following, we will look in more depth into the configuration space and more closely to the default parameter settings and the knowledge hidden in it. These settings are typically set based on the algorithm designers' intuition on the inner workings of their methods, as well as experiments that were conducted to evaluate it. As such, they contain more information than random values would. Whilst state-of-the-art algorithm configurators combine sophisticated methods to search the parameter space effectively, we argue that they often overlook precious information that could help them focus their search on the most promising values.

4.1.1 Background

As explained in Chapter 2, the AAC problem consists of an algorithm A, a configuration space Ω, a set of problem instances $\mathcal{I}$ and a performance metric m. Ω contains all valid combinations of parameters $\mathcal{P}$ and their possible values $\mathcal{D}$. More precisely, each parameter $p_j \in \mathcal{P}$ has a domain $\mathcal{D}_j \in \mathcal{D}$ of possible values and typically a default value $D_j \in \mathcal{D}_j$. This default value would typically be used if a user is unsure of what would work best for their particular problem case or when they try a new algorithm. Algorithm developers usually provide a default parameter configuration that has been chosen to perform reasonably well across a broad range of problem instances, which involves substantial human intuition, experience and at least limited manual experimentation.

Because the configuration landscapes of numerical parameters tend to be benign (Pushak and Hoos, 2018), even limited manual tuning may produce valuable information about promising parameter values. Moreover, starting the model-based configurator SMAC (Hutter et al., 2011b) from a performance model learned on another set of benchmark instances allows considerable speedups (Lindauer and Hutter, 2018). This suggests that knowledge regarding high-quality configurations can be transferred between sets of problem instances. Together, these observations indicate that default

parameter configurations may contain valuable information that can be exploited for AAC. For scenarios with large configuration spaces in which high-performance configurations are complex to find, we conjecture that searching configurations inside a reduced search space, obtained by restricting the ranges of specific parameters, can be more effective than searching the whole configuration space. Despite the fact that pruning the search space could potentially exclude good configurations, we show that searching on this reduced search space improves the performance of irace, GGA++ and SMAC. Since this first method would require the user to define those smaller search spaces, we want to integrate a mechanism into a configurator that focuses the search around the default without leaving out parts of the search space. We propose focusing the search around the default value by modifying the sampling distribution of SMAC.

4.1.2 Related work

The idea of excluding part of the search space in optimisation algorithms is common and originates as far as the work of Megiddo (1983). It has been applied in the context of automated machine learning (autoML) by excluding areas deemed to be poorly performing (Wistuba et al., 2015), or by focusing on the hyperparameters found to have a significant impact on the performance of the target algorithm (Li et al., 2022). In the context of AAC, the configurator GPS (Pushak and Hoos, 2020) is also based on a pruning approach.

To focus the search around the default value, we use a similar intuition as the estimation of distribution included in irace. Moreover, it has been demonstrated that automatic configuration with irace (López-Ibáñez et al., 2016) can be sped up by focusing the search process on specific areas of a given configuration space. Franzin et al. (2018) applied transformations to a real-valued parameter of a simulated annealing algorithm and showed that the right transformation can improve the performance of irace. A similar kind of mechanism has been developed later for Bayesian Optimisation (Souza et al., 2021).

4.1.3 Research questions

In this chapter, we focus on two of the research questions introduced in Section 1.2.

RQ4 *How and to which extent do current configurators use the default values usually provided by algorithm developers?*

Considering that there is information in the default value, the first thing to question

is how much current methods benefit from this information. Oftentimes, this value is used as a starting point for the search. To evaluate the extent to which the default parameter values are exploited by state-of-the-art AAC procedures, we compare the performance of two prominent configurators, SMAC and irace, with or without using the default setting to initialise the configuration process.

RQ5 *How can we make better use of known good parameter values – in our case, the default value – to guide the search strategy?* We separate this question into two parts as follows:

RQ 5.1. Can we reach state-of-the-art performance by searching only around the default value?

A naïve approach to leverage the default value of continuous parameters is to exclude any value that is deemed too far from it. This also significantly reduces the size of the search space of the configurators. We define two neighbourhoods of the default values and evaluate their impact on the performance of several state-of-the-art configurators across a broad range of configuration scenarios.

RQ 5.2. What is the impact of focusing the search around the default value?

A more refined approach to focusing the search around the default value would not completely exclude values that are further away from it, but sample more values in the promising area around the default than in the least promising areas of the search space. To achieve this, we use a truncated normal distribution centred around the default value. We evaluate the impact of such a distribution on SMAC.

4.2 Protocol of experiments

To answer our three research questions, we will need to compare the performance of configurators. For each such comparison, we followed the same protocol.

Scenarios

All configuration scenarios were run according to the setup described in AClib (Hutter et al., 2014a). The considered configuration objective is the minimisation of PAR10 (average running time of the target algorithm, with the timed-out runs counted as ten times the cutoff time) for Boolean satisfiability (SAT), mixed integer programming (MIP) and automated planning (AI planning). More details regarding the benchmarks and algorithms can be found in Chapter 2 and the scenarios' characteristics

are described in Table 2.4. To this set, we add four autoML scenarios originating from AClib on which we optimise the cross-validated error rate. They all use the same machine learning system, autoweka (autoWK) (Thornton et al., 2013; Kotthoff et al., 2017), an autoML system based on, and distributed as part of, the WEKA machine learning and data mining workbench. We configure this system on four datasets (from Dua and Karra Taniskidou, 2017; Thornton et al., 2013): CAR contains 1728 instances of car evaluations, GC contains 690 instances classifying people as good or bad credit risks, WF contains 5000 generated waves, and WQW contains 4898 instances modelling wine quality based on physicochemical tests.

Protocol

Algorithm configurators are randomised, and their performance is known to vary substantially between multiple independent runs on the same scenario. Thus, we ran each configurator independently 24 times for each scenario and evaluated the resulting 24 parameter configurations on our training and testing sets.
Moreover, to leverage this variability, it is best practice to perform multiple independent runs of a configurator on a given scenario (usually in parallel) and to report the best configuration (evaluated on the training instances) as the final result of the overall configuration process (see the recommendations of Eggensperger et al., 2019). To capture the statistical variability of this standard protocol, we repeatedly sampled 8 runs uniformly at random and identified the best one according to its performance on the training set. We used 10 000 such samples to estimate the probability distribution of the quality of the result produced by each configurator on each configuration scenario. Note that the outcome of such a sampling approach should follow the same distribution as the one obtained following Section 3.2.1. We then compared the medians of these empirical distributions, using a one-sided Mann-Whitney U-test ($\alpha = 0.05$) to assess the statistical significance of observed performance differences.

Configurators

We included SMAC, irace and GGA++ in this comparison. More details on those configurators can be found in Chapter 2. We excluded paramILS since it requires a discrete search space, which might interfere with the methods we propose. GPS had not been proposed at the time of this study. For irace, when the estimated running time of the algorithm was too long and it refused to run within the given time budget, we applied the same protocol to the successfully completed runs. This happened in

particular for the CPLEX scenarios, where AClib prescribes a cutoff time of 10 000 seconds. In such cases, when the random seed (ranging from 1 to 24) results in the selection of a hard instance for evaluating the running time of the default configuration, irace determines that the running time is too high for the given configuration budget. The decision to apply the standard protocol to successful runs of irace leads to a positive bias in the irace results for CPLEX on CLS, where 19 runs were completed successfully. However, in cases such as the scenarios for CPLEX on RCW2 and on REG200, where only 7 and 9 runs, respectively, terminated successfully, we omit the results from our analysis, as application of the standard protocol would lead to extreme distortions from realistically achievable performance.

Code and execution environment

All experiments were performed on a computing cluster with CentOS using dual 16-core 2.10 GHz Intel Xeon E5-2683 CPUs with 40 MB cache and 94 GB of RAM. Our source code and results are available at `ada.liacs.nl/projects/smacps`.

4.3 Impact of the default

As mentioned previously, most configurable algorithms come with default parameter values that the developer has carefully chosen. Each AAC procedure handles those default values differently and thus is impacted differently by it. In this section, we explore the impact that the default configuration has on existing configuration procedures.

To answer RQ4 – *How and to which extent do current configurators use the default values usually provided by algorithm developers?* – let us look into the behaviour of two configurators. SMAC uses the default configuration as one of the starting configurations for building the model that guides its search process. irace not only uses the default as one of its initial racing configurations, but also as a way to estimate the running time of the target algorithm and thus the number of iterations.

For two of them, we compare their performance when provided with the default value given in AClib against their performance when no default value is given. The results obtained following the protocol described in Section 4.2 are reported in Table 4.1.

SMAC

SMAC finds better configurations for 5 out of 10 scenarios when starting from default parameter values, for 4 out of 10 scenarios starting from a random configuration produces better results, and for one scenario, no significant difference is observed. Overall, there seems to be no clear advantage in using the default configuration as a starting point. This is not surprising, considering that SMAC very quickly bases its search on the predictions of its random forest model and randomly chosen configurations. The latter, included to avoid stagnation of the model-based search process, likely limits the impact of using a specific starting configuration.

irace

When given a limited time budget (as in all our experiments), irace uses the default configuration of the given target algorithm to estimate the running time of the target algorithm. Consequently, when starting from a randomly chosen configuration, under which the target algorithm performs very poorly, irace may refuse to run, since it assumes there is insufficient time for the racing process to produce meaningful results within the given time budget. The results reported in Table 4.1 show only the results for scenarios for which at least 8 configurator runs finished within 5 times the given overall time budget for configuration. irace was unable to run on 2 of our 10 benchmark scenarios. For 5 others, it performs better when given access to the default configuration, and for the 3 remaining scenarios, starting from a random configuration produces better results. We note that irace systematically ran over time when starting from a random configuration.

What is the impact of the default value on current configurators?

To answer this question, we compared how widely used configurators perform with and without access to meaningful default parameter settings. We found that the performance of two state-of-the-art configurators, SMAC and irace, is affected very differently by default settings: While SMAC only rarely benefits from reasonable defaults, they are often crucial for irace.

4.4 Reduction of the search space

Automatic algorithm configurators are sophisticated search algorithms searching the configuration space of a given target algorithm. Due to the various types of parameters,

		Default	SMAC		irace	
			def	rand	def	rand
SpToRiss	CF	424.43	223.27	**196.62**	224.10	**223.05**
	LABS	885.78	**780.94**	**787.00**	**803.03**	815.06
	UNSAT	152.33	1.25	**1.13**	1.36	**1.20**
CPLEX	CLS	3.46	**2.14**	2.45	4.44	**3.99**
	COR-LAT	52.14	**7.75**	7.98	**10.64**	14.56
	REG200	10.83	**4.04**	4.13	**4.55**	4.86
AutoWK	CAR	0.500	**0.250**	0.270	**0.300**	-
	GC	0.590	0.330	**0.280**	**0.240**	-
	WF	1.97	0.340	**0.220**	**0.230**	0.300
	WQW	1.83	**0.350**	0.360	**0.370**	0.410

Table 4.1: Results for SMAC (left) and irace (right) for default and random initial configurations; median PAR10 (in CPU sec) for SAT and MIP, 10-fold cross-validated error rate for ML; best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$).

the conditions linking them and the effects of their interaction, it can be hard to search the space of possible configurations. As argued earlier in this chapter, we have reasons to believe that the default value is located in a promising area of the search space. To test this hypothesis and answer RQ 5.1 – *Can we reach state-of-the-art performance by searching only around the default value?* – we restrict the search space of the configurator to the neighbourhood of the configuration.

4.4.1 Reduction methods

We will focus on integer- and real-valued parameters for our experiments, as their domains are often large intervals and reducing them can significantly impact the size of a given configuration space. Moreover, there is no straightforward way to reduce the range of a categorical parameter in a meaningful manner. We note that some solver have values in their integer-valued parameters with a specific meaning, different from the expected one. This is particularly true for CPLEX, which uses -1 as a special value, meaning that CPLEX will decide itself which value to use for this parameter. However, we did not treat those values differently in our reduction.

How to reduce parameter domains?

For $k \in \{1, 2, ..., n\}$, the domain of the integer- or real-valued parameter p_k is defined by its lower bound $D_{k,min}$, its upper bound $D_{k,max}$ and its default value D_k. The length of its range is denoted $D_{k,range} = D_{k,max} - D_{k,min}$.

We consider two different reduction techniques to calculate a new domain $\mathcal{D}_{k,sub}$. To avoid reducing small ranges, we apply domain reduction only to parameters with at least 10 possible values for integers and a range of 1 for real numbers. For parameters that vary on a logarithmic scale, we apply the reductions in the logarithmic domain.

Our first technique reduces the domain of a given parameter so it begins at one-tenth of the default value and ends at ten times the default value:

$$\mathcal{D}_{k,sub} = [0.1 \cdot D_k, 10 \cdot D_k]\,. \tag{R1}$$

This technique has the advantage of scaling up when the default value is big. For example, given the parameter p_k with a domain $D_k = [0, 10\,000]$, a default value $D_{k,\text{def}} = 500$ leads to a large range of $D_{k,\text{reduced}} = [50, 5\,000]$ while a default value of $D_{k,\text{def}} = 10$ leads to a large range of $D_{k,\text{reduced}} = [1, 100]$. However, it handles domains that span both positive and negative values poorly. For example if p_k has a domain $D_k = [-100, 100]$ and a default value $D_{k,\text{def}} = 5$ then the reduced range will become $D_{k,\text{reduced}} = [0.5, 50]$.

Our second technique reduces the range to a tenth of $\mathcal{D}_k$, centred around the default value:

$$\mathcal{D}_{k,sub} = \left[D_k - \frac{D_{k,range}}{20}, D_k + \frac{D_{k,range}}{20}\right]. \tag{R2}$$

Unlike the previous one, this technique does not have any issues with domains that span both positive and negative values.

For example if p_k has a domain $D_k = [-100, 100]$ and a default value $D_{k,\text{def}} = 5$, the reduced range will become $D_{k,\text{reduced}} = [-15, 25]$.

As $\mathcal{D}_{k,sub}$ thus defined could exceed the bounds of $\mathcal{D}_k$, we obtain the reduced range by taking the intersection of the two ranges. The final reduced range $\mathcal{D}_{k,red}$ is defined as

$$\mathcal{D}_{k,red} = \mathcal{D}_k \cap \mathcal{D}_{k,sub} \tag{4.1}$$

Moreover, each step of the reduction is made such that we keep at least 10 possible values for integers and a length of 1 for real numbers, meaning that this rule applies to $\mathcal{D}_{k,sub}$ and we add values to $\mathcal{D}_{k,red}$ as a final step if needed.

Solver	Parameters		Reduced parameters			
	Total	Numerical	R1		R2	
Clasp	75	37	25	(24)	29	(28)
Lingeling	322	185	154	(0)	163	(0)
SpToRiss	222	52	10	(9)	20	(16)
LPG	67	19	7	(4)	12	(8)
CPLEX	74	23	20	(2)	20	(2)
AutoWK	702	226	60	(60)	111	(111)

Table 4.2: Target algorithms parameter space description; The total number of numerical parameters; the number of parameters reduced by our techniques and, in parentheses, the number of these conditionally dependent on at least one other parameter.

How does the range reduction affect the configuration spaces of the considered algorithms?

We applied both reductions to the parameter space of the six algorithms appearing in our configuration scenarios. As seen in Table 4.2, for Clasp, CPLEX and Lingeling, the ranges of between 27 and 50% of the total number of parameters are reduced, which suggests that there is potential for a large impact when configuring them. We note that the parameters of Clasp to which reduction is applied almost all conditionally depend on at least one other parameter value, which may reduce the effect of domain reduction in this case. For SpToRiss, LPG and AutoWK, only 5 to 17% of the parameters are affected by domain reduction, and we thus expect the effect on configurator performance to be less pronounced.

4.4.2 Results

To evaluate the impact of search space reduction on the configuration process, we ran extensive experiments with SMAC, as well as more limited experiments with irace and GGA++. We then applied the protocol described in Section 4.2 to compare the results obtained for the full configuration space with those obtained using our two reduction techniques.

How do our reduction techniques impact the performance of SMAC?

As seen in Table 4.3, reducing the search space allowed SMAC to perform better for 15 out of the 20 studied scenarios, although for one of these, none of the optimised configurations reached the quality of the default configuration (on testing data). For the 5

		Quality			
		Default	full	R1	R2
Clasp	CF	174.05	**164.72**	173.67	173.60
	LABS	**718.15**	728.87	720.62	736.70
	UNSAT	0.847	0.380	**0.376**	0.383
Lingeling	CF	278.52	245.48	**187.11**	228.65
	LABS	808.70	830.63	**788.36**	849.25
	UNSAT	2.03	1.07	**0.984**	1.04
SpToRiss	CF	424.43	223.27	212.04	**204.21**
	LABS	885.78	780.94	**756.55**	805.10
	UNSAT	152.33	**1.25**	1.30	1.29
LPG	Depots	26.77	0.776	**0.709**	0.826
	Satellite	16.72	3.83	**3.70**	3.79
	Zenotravel	22.49	**1.67**	1.75	1.72
CPLEX	CLS	3.46	**2.14**	2.43	2.69
	COR-LAT	52.14	7.66	**6.01**	14.85
	RCW2	64.98	**52.64**	65.11	54.60
	REG200	10.83	4.04	**3.53**	3.73
AutoWK	CAR	0.500	0.250	**0.220**	0.280
	GC	0.590	0.330	0.280	**0.270**
	WF	1.97	0.340	0.370	**0.280**
	WQW	1.83	0.350	0.390	**0.340**

Table 4.3: Results for SMAC; median PAR10 (in CPU sec) for SAT, MIP and Planning, 10-fold cross-validated error rate for ML; best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$).

remaining scenarios, significantly better results were obtained for the full configuration space. Reductions R1 and R2 lead to improved results for 12 and 8 scenarios, respectively. These improvements are observed across all AI problems and target algorithms that we studied.

Do our observations generalise to other configuration approaches?

Since each evaluation requires long and computationally expensive configuration runs, we ran limited experiments with irace and GGA++. We tested one target algorithm for each type of problem and chose among the benchmarks based on the results obtained by SMAC. We kept, for each problem type, one scenario on which each reduction technique improved the results of SMAC, as well as one for which it worsened them.

			irace quality			GGA++ quality		
		Default	full	R1	R2	full	R1	R2
SpToRiss	CF	424.43	**224.10**	**206.72**	246.99	233.11	235.83	**225.82**
	LABS	808.70	**803.03**	788.66	**787.47**	835.32	**804.31**	847.83
	UNSAT	152.33	**1.24**	1.28	**1.23**	**1.53**	1.61	1.54
LPG	Satellite	16.72	**4.94**	6.41	6.45	**3.69**	3.96	3.78
	Zenotravel	22.49	2.35	1.90	**1.84**	6.28	3.32	**3.07**
CPLEX	RCW2	64.98	69.62	68.16	**60.48**	63.87	63.69	**63.43**
	REG200	10.83	4.55	4.69	**3.86**	12.01	11.30	**10.57**
AutoWK	CAR	0.500	**0.210**	0.370	0.230	0.300	0.630	**0.260**
	WF	1.97	**0.230**	0.250	0.260	2.69	**2.40**	2.55

Table 4.4: Results for irace and GGA++; median PAR10 (in CPU sec) for SAT, MIP and Planning, 10-fold cross-validated error rate for ML; best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$).

We kept three SAT, two AI planning, two MIP and two ML scenarios, resulting in a total of 9 scenarios. The results for irace and GGA++ are shown in Table 4.4.

irace performed better on our reduced spaces for 3 out of 9 scenarios, worse for 3 out of 9, and showed no significant performance differences for the remaining 3. We also notice that for CPLEX on RCW2, irace could not find a significantly better configuration than the default unless using our search space reduction techniques. Thus, exploiting the knowledge contained in the default parameter values through our reduction techniques benefited irace for these challenging scenarios. On the other hand, there are limited or no improvements for AutoWK and SpToRiss, which is unsurprising, considering that most of their reduced parameters are conditionally dependent on other parameter values (see Table 4.2). For SpToRiss on LABS, it is surprising that the Mann-Whitney test found evidence that the distribution for R1 is worse than that for R2, whereas this was not the case for the full configuration space. Further investigation revealed that the distribution of configuration quality obtained for R1 contains some significantly worse results than those for R2 and the full space, which causes the observed result. We note that irace implements a mechanism that resembles our range reduction techniques by focusing on sampling configurations around combinations of parameter values known to yield high performance. More precisely, when it generates a new configuration, it samples the values according to a Gaussian distribution centred around the best-known values and reduces the variance along the run to focus on promising parts of the configuration space. This focused sampling could explain

why the reduction did not significantly impact the performance of irace and is also the inspiration behind the work presented in section 4.5. GGA++ performed better on our reduced spaces for 7 out of 9 scenarios and worse for the remaining 2. There is no specific advantage to one or the other of the reductions; however, search space reduction consistently yields improvements.

Further investigation of our results.

A possible concern arising from the way we reduce the search space is that a strong reduction may exclude the global optimum from the search space. Unfortunately, there are no known methods for determining configurations that are guaranteed globally optimal for the kinds of scenarios we consider. However, by looking at the best-known configurations seen over all configurator runs for each scenario, we can at least develop some intuition. In Table 4.5, we show how many of the parameters have their best known value outside of the reduced ranges. We see that for Clasp, SpToRiss, LPG and AutoWK, the range reduced according to R1 contains the best known value for almost all the parameters, for Lingeling the range reduced according to R2 contains almost all the best known values, and for CPLEX the ranges reduced according to the two reductions contain almost all the best known values except for the CLS scenario. We could then expect to reach a better configuration with R1 for Clasp, SpToRiss, LPG and AutoWK and with R2 for Lingeling. However, the results presented in Tables 4.3 and 4.4 show that for AutoWK, Lingeling and LPG, the configuration results do not correspond to those expectations. Thus, for those three scenarios, excluding promising parts of the configuration space and possibly losing globally optimal configurations, we were still able to reach better configurations on average. This suggests that reducing the size of a given configuration space can make it easier for existing configuration procedures to find good local optima consistently.

In some cases, more particularly Lingeling on UNSAT and AutoWK on CAR and GC, the high number of parameter values that are outside of the reduced range might suggest that the default parameter value may not be a good focal point for the configuration process. However, for those three cases, one of the reductions actually contained the best-known configuration. Moreover, in preliminary experiments based on the idea of running a first short configuration run to find a better starting point than the default, we failed to observe better results than those obtained by reduction around the default.

Algorithm	Benchmark	R1	R2	Numerical Parameters
Clasp	CF	0	0	37
	LABS	0	0	
	UNSAT	2	7	
Lingeling	CF	0	0	185
	LABS	0	9	
	UNSAT	53	0	
SpToRiss	CF	1	0	52
	LABS	0	9	
	UNSAT	1	14	
LPG	Depots	0	6	19
	Satellite	0	9	
	Zenotravel	2	10	
CPLEX	CLS	7	7	23
	COR-LAT	0	2	
	RCW2	2	1	
	REG200	0	1	
AutoWK	CAR	0	42	226
	GC	0	40	
	WF	1	0	
	WQW	0	0	

Table 4.5: Number of parameters of the best known configuration (testing set) that take a value outside the reduced ranges (using R1 and R2, respectively)

Can we reach state-of-the-art performance by searching only around the default value?

Empirical results for well-known configuration scenarios for SAT, MIP, AI planning and autoML clearly indicate that using these reduction techniques, the state-of-the-art general-purpose configurators SMAC and GGA++ tend to find significantly better configurations within a given time budget. irace benefits from default-guided range reduction for scenarios with many numerical parameters that are not dependent on higher-level categorical design choices, such as CPLEX and LPG, while we did not observe benefits for scenarios with few numerical parameters, or numerical parameters that mainly depend on higher-level categorical choices, as for SpToRiss and AutoWK. This supports our hypothesis that configurators can benefit from the information contained in expert-determined default values for target algorithm parameters. Also, comparing our results on SparrowToRiss to those recently published for warm-

starting SMAC (Lindauer and Hutter, 2018), we find that the default-guided search space reduction gives similar, yet complementary benefits, in the sense that we obtain improvements where they do not, and vice versa. However, in contrast to warm-starting, our approach is applicable to a broader range of automatic configuration procedures.

The efficacy of our default-guided range reduction techniques suggests an interesting, largely unexplored direction for improving automated algorithm configurators, based on the idea of more strongly exploiting default configurations in the underlying search process, which we explore in the following section.

4.5 Probabilistic sampling

As shown previously, simply reducing the ranges of numerical parameters around the given default values can lead to significant performance improvements for SMAC (see Section 4.4 or Anastacio et al. (2019)). However, by pruning the search space, our previous approach completely excluded regions that, in specific cases, could be relevant to explore, and we did so in a configurator-agnostic, yet somewhat ad-hoc manner. Moreover, we saw that those reduction methods had a limited impact on the performance of irace since it already includes a mechanism to sample more configurations around its elite configurations. Building on those findings, our intuition is that sampling near good configurations will very likely lead to other good configurations, while sampling uniformly, as done in SMAC, will likely waste time on some underperforming parameter settings.

4.5.1 Extension with probabilistic sampling

We propose a principled approach for including the prior knowledge contained in the default value of parameters into the search process, by applying probabilistic sampling in the context of sequential model-based optimisation. As we will demonstrate, this can yield significant improvement for general-purpose AAC. To implement our approach, we extended SMAC with a simple mechanism for sampling new values according to a truncated normal distribution centred around the given default values, which replaces the uniform random sampling used in the original version of SMAC.

Algorithm 4.1 outlines at a high level our new configuration method, which we refer to as SMACPS. SMACPS differs from SMAC only in the sampling distribution

Algorithm 4.1 SMACPS

Require: Ω: configuration space, ω_d: the default configuration, Ω_{rand}, Ω_{prom} and Ω_{new}: sets of configurations.

1: $\mathcal{R}$: target algorithm runs performed. M: performance model.
2: $\omega_{\text{inc}} \leftarrow \omega_d$
3: $\mathcal{R} \leftarrow run\,(\omega_d)$
4: **while** Budget not exhausted **do**
5: $\quad M \leftarrow update\,(M, \mathcal{R})$
6: $\quad \Omega_{\text{prom}} \leftarrow uniform\_sample\_configurations\,(\Omega)$
7: $\quad \Omega_{\text{prom}} \leftarrow local\_search\,(M, \Omega_{\text{prom}})$
8: $\quad \Omega_{\text{rand}} \leftarrow normal\_sample\_configurations\,(\Omega)$
9: $\quad \Omega_{\text{new}} \leftarrow interleave\,(\Omega_{\text{rand}}, \Omega_{\text{prom}})$
10: $\quad inc \leftarrow intensify\,(\Omega_{\text{new}}, \omega_{\text{inc}})$
11: **end while**
12: **Return** ω_{inc}

used in line 7; further details of SMAC can be found in the original paper[2] (Hutter et al., 2011b). New configurations are sampled in two places: as starting points for the local search process (line 5) and for non-model-based diversification (line 7). The two sets of configurations thus obtained are then interleaved and raced against the current incumbent (line 9). We change the sampling distribution used for non-model-based diversification (line 7) – note that in Anastacio and Hoos (2020a) it was changed at both places. Each parameter is sampled independently from a distribution that depends on the parameter type and domain. For each numerical parameter p_n, we first normalise the range to $[0, 1]$ (if p_n is specified as "log scale", we first apply a log base 10 transformation). We then sample a value from a truncated normal distribution with mean equal to the default value of p_n and variance 0.05; this value was chosen based on preliminary experiments on configuration scenarios different from the ones used in our evaluation, namely the SAT solvers *Spear* and *Satenstein* on benchmark *SWGCP*; additional details can be found on AClib. For a categorical parameter p_c with k values, we sample the default value with probability 0.5, and each other value with probability $0.5/(k-1)$.

In cases where the default configuration is far away from the most promising areas of a given configuration space, this approach might be counterproductive. We note, however, that model-based local search, starting from uniformly sampled configurations, has the potential to counterbalance this effect.

[2]Recent versions of SMAC behave slightly differently, picking challengers from C_{prom} or C_{rand} with given probabilities instead of alternating.

Our probabilistic sampling approach is inspired by estimation of distribution algorithms, which are known to provide an effective way for leveraging prior knowledge when solving complex optimisation problems (Hauschild and Pelikan, 2011). It is also conceptually related to the approach taken by irace, which, unlike SMAC, does not use an empirical performance model to map parameter configurations to performance values, but instead uses the best-known configurations as a basis for estimating where promising parameters may be located. Indeed, irace employs an intensification mechanism that involves sampling new values for numerical parameters from truncated normal distributions, in conjunction with a racing mechanism for updating the incumbent configuration and the locations of the sampling distributions. We note that the probabilistic sampling process used in SMAC is simpler, as it maintains fixed sampling distributions throughout the configuration process. We decided on this design to evaluate the extent to which a simple probabilistic sampling mechanism solely focused on exploiting information from expert-chosen default values would already enable improvements over state-of-the-art algorithm configurators, such as SMAC and irace.

4.5.2 Results

In this section, we compare the algorithm described in Section 4.5.1 against the state-of-the-art configurations SMAC and irace following the protocol described in Section 4.2.

Comparison based on median PAR10 scores

We compare the performance obtained for the configuration scenarios we studied, following the protocol described in Section 4.2, which produces statistics on how state-of-the-art configurators are commonly used in practice. Table 4.6 shows the median PAR10 values we obtained; the missing results for CPLEX on RCW2 and REG200 are since irace refused to start more than half of the 24 runs, since it considered the configuration budget to be insufficient.

Comparing the results for SMAC and irace, we note that in most cases, SMAC achieves better performance than irace. SMAC achieved a statistically significantly lower median PAR10 score in 10 out of the 16 configuration scenarios we studied, while irace outperformed SMAC in the remaining 6 scenarios. This indicates complementary strengths of our two baseline configurators. While SMAC has an edge on most of the scenarios, there is at least one case where irace finds substantially better

Table 4.6: Results for SMAC, SMACPS and irace; median PAR10 (in CPU sec); best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$). Right columns highlight the result of the pairwise comparison between SMACPS and the two baselines; ✓ if it is better, ✗ if not; parentheses indicate that the difference is not statistically significant.

Solver	Benchmark	Default	SMAC	irace	SMACPS	≻ SMAC	≻ irace
Clasp	CF	193.87	193.00	**174.00**	**173.61**	✓	(✓)
	LABS	745.74	**837.93**	847.10	**837.61**	(✓)	✓
	UNSAT	0.885	**0.362**	**0.359**	**0.359**	(✓)	(✓)
Lingeling	CF	327.00	**261.06**	328.214	300.40	✗	✓
	LABS	873.67	866.99	959.35	**863.73**	✓	✓
	UNSAT	2.41	**1.59**	2.36	1.65	✗	✓
SpToRiss	CF	472.78	**226.51**	236.61	253.72	✗	✗
	LABS	911.25	857.67	846.40	**805.46**	✓	✓
	UNSAT	222.26	1.56	1.56	**1.55**	✓	✓
LPG	Depots	34.68	1.14	**1.08**	1.25	(✗)	✗
	Satellite	22.40	5.43	8.04	**5.19**	✓	✓
	Zenotravel	29.64	2.61	2.93	**2.56**	✓	✓
CPLEX	CLS	4.06	3.36	4.06	**2.95**	✓	✓
	COR-LAT	24.81	21.84	**10.04**	21.26	✓	✗
	RCW2	82.51	**71.98**	–	78.10	✗	✓
	REG200	13.08	5.56	–	**5.09**	✓	✓

configurations (CPLEX on COR-LAT) – looking back at Chapter 3 allows us to see that the performance of SMAC3 on this scenario is poor.

Comparing SMACPS against SMAC, which is the strongest of our two baselines, and served as the starting point for our new configuration procedure, we notice that for 11 of the 16 scenarios, SMACPS reaches a lower median PAR10 score. In all but two of those cases (Clasp on LABS and UNSAT), the performance differences are statistically significant.

Finally, compared to irace, SMACPS achieves better performance on 13 of our 16 scenarios. In all but two cases (Clasp on CF and UNSAT), the differences are statistically significant. SpToRiss on LABS shows a case where SMAC could not achieve better result than irace, while SMACPS outperforms irace.

Overall, these results indicate clearly that SMACPS represents a significant improvement over both baselines.

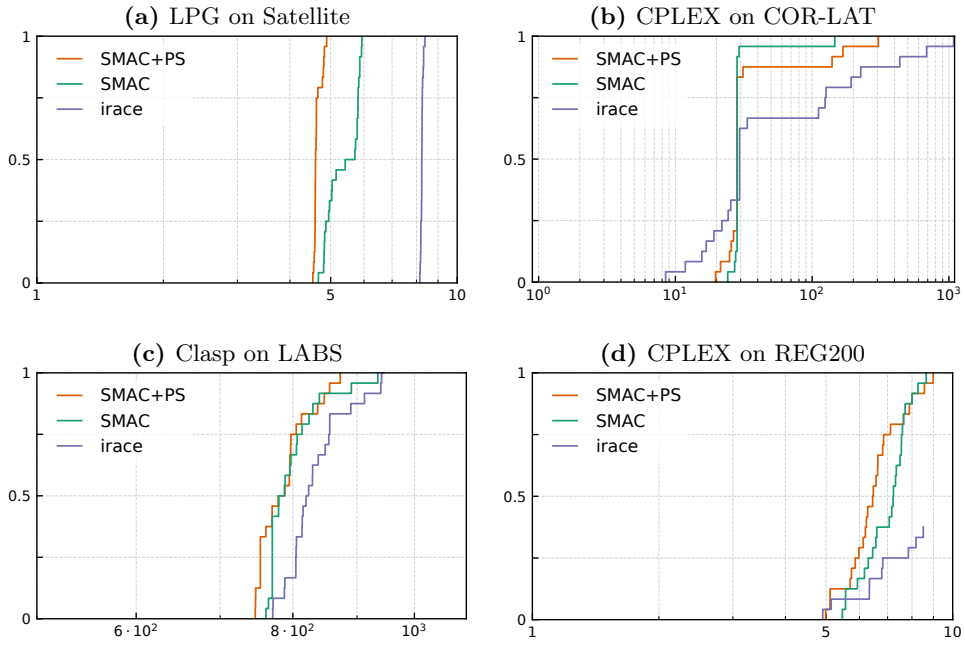

Figure 4.1: Cumulative distribution functions for PAR10 scores (in CPU seconds, x-axis) over independent configurator runs on the testing set.

Distributions of PAR10 scores over multiple configurator runs

To examine the performance of SMAC, irace and SMACPS in more detail, we studied the empirical cumulative distribution functions over individual configurator runs (without applying the standard protocol) – see Figure 4.1. As we minimise PAR10 scores, better performance is indicated by CDFs closer to the top left corner of the plots. We present results for four scenarios, each yielding a qualitatively distinct outcome in our comparison.

Figure 4.1a, LPG on Satellite, is a case in which the difference between the CDFs for SMAC and SMACPS is particularly pronounced. In this scenario, SMACPS clearly dominates the two other configurators.

Figure 4.1b, CPLEX on COR-LAT, is a case in which irace performs better than the two other configurators. SMAC and SMACPS give rise to similar performance distributions. irace has an edge, reinforced by the standard protocol, which leverages the left tail of the performance distributions.

Figure 4.1c, Clasp on LABS, a scenario on which SMAC and SMACPS are statistically tied, looks qualitatively similar, except that the difference in the left tail between SMAC and SMACPS is too small to be reliably exploitable using the standard protocol. irace, on the other side, is probabilistically dominated on this scenario.

Figure 4.1d shows a scenarios on which irace terminated prematurely (as described in Section 4.2). However, examining the partial CDFs for the successfully completed configurator runs, there is no reason to expect that irace would have performed significantly better than SMAC and SMACPS.

What is the impact of focusing the search around the default value?

We proposed a simple yet effective method to probabilistically bias the sequential model-based algorithm configurator (SMAC) (Hutter et al., 2014a) towards a given default configuration. To do so, we replaced its uniform random sampling mechanism with a probabilistic sampling approach for numerical parameters.

We evaluated the resulting procedure, dubbed SMACPS, against two widely used and freely available state-of-the-art general-purpose algorithm configurators, SMAC and irace (López-Ibáñez et al., 2016). For this comparison, we used 16 running time optimisation scenarios from AClib, a widely used library of benchmarks for AAC (Hutter et al., 2014a).

We found that SMACPS performs better than SMAC on 11 of those 16 scenarios, and better than irace on 12 of them, and thus represents a significant improvement over the state of the art in AAC for running time minimisation. Whether similar results can be obtained for different performance metrics is an open question.

4.6 Conclusion

This chapter explored approaches to leverage the expert knowledge contained in the default value. We believe that for most state-of-the-art algorithms for challenging AI problems, default parameter values are chosen with care, in many cases not only based on limited experiments, but also on deep insights into the heuristics controlled by the parameters.

Answering RQ4 in Section 4.3, we demonstrated that each configurator relies on the default parameter values in different ways. As a proof of concept and to answer RQ 5.1, Section 4.4 introduces a naïve approach to prune the configuration space around the default value, thereby searching only among values near it. We obtained significant

improvement in configurator performance. However, our reduction techniques depend on the quality of the given default values and hold the risk of excluding promising areas of the search space if the default value was poorly chosen. It is also expected to have a greater impact when targeting numerical parameters with wide ranges. This is the case with many numerical parameters, because, in principle, extreme values produce valid behaviour of the algorithm, and the effort to manually restrict the range to good values can be considerable; thus, it is often avoided by algorithm designers. We also expect that reducing parameter ranges too aggressively will ultimately lead to degraded performance.

Section 4.5 introduced a more subtle incorporation of the default in the search strategy to answer RQ 5.2. By using a truncated distribution centred around the default value, we bias the search towards the promising area around the default while maintaining a probability of deviating from it if the default was poorly chosen. We compared our methods on a set of running time optimisation scenarios. We demonstrated that, in the majority of scenarios, the introduced methods outperform configurators that do not utilise the knowledge held in the default. We thus argue that the often-overlooked insights of experts can and should be included in configurators, as they hold the potential to guide the search. Our results are consistent with recent work showing that the configuration landscapes (*i.e.*, the functions relating parameter values to target algorithm performance) are far more benign than one might have expected (Pushak and Hoos, 2018), which suggests that sampling around known good parameter values provides an efficient way towards finding new good values, an assumption also leveraged by irace.

Our work also opens an interesting path towards richer mechanisms for allowing algorithm designers to express prior knowledge about parameter values. Note that, following the publication of results presented in this chapter, subsequent work explored this path further by allowing users (or algorithm designers) to define detailed priors on each parameter (see *e.g.* Souza et al., 2021; Hvarfner et al., 2022).