Exploitation of Default Parameter Values in Automated Algorithm Configuration

Marie Anastacio[1] , **Chuan Luo**[1,2] , **Holger Hoos**[1,3]
[1]LIACS, Leiden University, Leiden, The Netherlands
[2]Microsoft Research, Beijing, China
[3]University of British Columbia (UBC), Vancouver, Canada
m.i.a.anastacio@liacs.leidenuniv.nl

Abstract

Over the last few years, automated algorithm configuration methods have achieved impressive success for a broad range of prominent AI problems. This increasingly encourages algorithm developers to expose parameters to the automated optimisation process, leading to substantially larger configuration spaces that are more challenging to search. In most cases, when exposing parameters, algorithm designers choose default values based on their intuition, understanding of the underlying behaviour involved and at least limited experimentation.

In this work, we first demonstrate that state-of-the-art automated algorithm configuration procedures moderately exploit information provided by default parameter values. Then, based on insights given by previous work on parameter response landscapes, we introduce two simple techniques to utilise this information by focusing the search around the default parameter values. By reducing the range of possible values, and thus the size of the search space, we improve the efficiency of prominent automatic configuration procedures for a broad range of widely used configuration scenarios. Specifically, our reduction techniques increases the efficacy of SMAC for 15 out of the 20 configuration scenarios we studied, that of GGA++ for 7 out of the 9 scenarios, and that of irace for a scenario on which previously only minor improvements could be achieved.

1 Introduction

The performance of state-of-the-art algorithms for many AI problems critically depends on the settings of certain parameters [Hutter *et al.*, 2009]. Traditionally, algorithm developers have manually tuned these parameters to find configurations that perform well across a range of benchmarks. However, this manual configuration process tends to be tedious and inefficient. For the past decade, automated algorithm configuration has emerged as an effective alternative to this manual approach. Prominent examples of general-purpose automatic algorithm configurators include irace [López-Ibáñez *et al.*, 2016], ParamILS [Hutter *et al.*, 2009], GGA/GGA++ [Ansótegui *et al.*, 2009; 2015] and SMAC [Hutter *et al.*, 2011]. These fully automated configurators are based on advanced machine learning and optimisation methods and have been demonstrated to be strikingly effective across a broad range of challenging problems of high significance to AI, including propositional satisfiability (SAT) (see, *e.g.*, [Hutter *et al.*, 2017]), mixed integer programming (MIP) (see, *e.g.*, [Hutter *et al.*, 2010]), automated planning (see, *e.g.*, [Fawcett *et al.*, 2011]) and supervised machine learning (ML) (see, *e.g.*, [Thornton *et al.*, 2013; Kotthoff *et al.*, 2017]).

The availability of effective automated algorithm configurators encourages algorithm designers to expose design choices as parameters. This has lead to a design paradigm known as programming by optimisation (PbO) [Hoos, 2012]. PbO is based on the idea that by avoiding premature choices in the design of algorithms, we can let automated methods, such as general-purpose algorithm configurators, make design choices specifically adapted to a use case and thus achieve better performance. However, the size of the combinatorial configuration spaces in this context grows exponentially with the number of exposed parameters, which can pose considerable challenges for state-of-the-art configurators.

Meanwhile, the choice of default parameter settings still tends to involve substantial human intuition, experience and at least limited manual experimentation. We conjecture that these default settings contain valuable information that can be exploited in the automatic configuration process.

In this work, we explore the degree to which this information is leveraged by state-of-the-art algorithm configurators. Our empirical results indicate that the degree to which default parameter settings matter highly depends on the configurator in question. While SMAC does not benefit much, irace tends to critically depend on good defaults. To take better advantage of the default settings, we propose two simple techniques that leverage the default values of continuous parameters to reduce the size of a given configuration space. Using these generic reduction techniques, we obtain improved results for two of the three prominent general-purpose configurators we study.

The remainder of this paper is structured as follows. Section 2 introduces the algorithm configuration problem, three state-of-the-art, general-purpose procedures for solving it, and the two research questions we investigate in the following. Section 3 describes the configurators, algorithms and benchmarks used in our study, as well as our experimental setup. In Section 4, we compare the performance of the configurators with and without giving them access to the default parameter values provided in our benchmark scenarios. In Section 5, we outline our two parameter reduction techniques. In Section 6, we show how the performance of SMAC, one of the most widely used algorithm configurators, is affected by these reduction techniques, and investigate how these observation generalise to other configurators. Finally, Section 7 draws general conclusions and insights from our results.

2 Automatic algorithm configuration

The algorithm configuration problem can be defined as follows. Given:

- a target algorithm A with parameters $p_1, p_2, ..., p_k$, a domain D_j of possible values and a default value $d_j \in D_j$ for each parameter p_j;
- a configuration space C, containing all valid combinations of parameter values of A;
- a set of problem instances I;
- a performance metric m that measures the performance of a configuration $c \in C$ of target algorithm A on I;

find $c^* \in C$ that optimises the performance of A on instance set I, according to metric m.

There are different types of parameters. Categorical parameters have an unordered, finite set of values; they are often used to select between several heuristic components or mechanisms.Boolean parameters typically activate or deactivate optional algorithm components. Numerical, integer- or real-valued parameters are often used to quantitatively control performance-relevant aspects of a given target algorithm, such as a learning rate. Some parameters may conditionally depend on others, in that they are only active for certain values of other (often categorical) parameters.

In recent years, much work has been done on automatic algorithm configuration, resulting in several general-purpose automatic algorithm configurators, based on different approaches. The configurator irace [Birattari *et al.*, 2010; López-Ibáñez *et al.*, 2016] leverages racing methods based on a statistical test in conjunction with model-based sampling of configurations. GGA [Ansótegui *et al.*, 2009] and GGA++ [Ansótegui *et al.*, 2015] are based on gender-based evolutionary algorithm. SMAC [Hutter *et al.*, 2011] is based on a sequential model-based (aka Bayesian) optimisation approach and utilises a random forest model and an expected improvement criterion to produce promising configurations.

Default parameter values. Algorithm developers usually provide a default parameter configuration that has been chosen to perform reasonably well across a broad range of problem instances. A recent study of the dependence of algorithm performance on the setting of numerical parameters has found that these configuration landscapes tend to be surprisingly benign [Pushak and Hoos, 2018]. This suggests that even limited manual tuning may produce useful information about promising parameter values. At the same time, it has been shown that by starting SMAC from a performance model learned on a another set of benchmark instances, considerable speedups can be obtained compared to SMAC runs without this kind of prior knowledge [Lindauer and Hutter, 2018]. This suggest 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 automatic algorithm configuration. In light of this, we consider the following research question :

(RQ1) How much impact does the default configuration have on current configurators? To obtain insight into the degree to which default parameter values are exploited by state-of-the-art algorithm configuration procedures, we compare performance of two prominent, yet very different, configurators, SMAC and irace, with or without using the default setting to initialize the configuration process.

Search space reduction. Recently, it has been demonstrated that automatic configuration with irace can be sped up by focusing the search process on specific areas of a given configuration space, by means of applying transformations to one real-valued parameter of a simulated annealing algorithm [Franzin *et al.*, 2018]. This work also produced evidence that poorly chosen transformation can degrade the automatic configuration process. As mentioned above, we have reason to believe that promising regions of values for individual parameters can often be found around expert-determined default values, and we aim to investigate this hypothesis. For scenarios with large configuration spaces in which high-performance configurations are difficult to find, we conjecture that searching configurations inside a reduced search space, obtained by restricting the ranges of certain parameters, can be more effective than searching the full configuration space. This leads us to our second research question:

(RQ2) Can we leverage the information contained in the default by reducing the range of the parameters? Specifically, we introduce two techniques that limit the search to a neighbourhood 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.

3 Experimental setup

To answer our research questions, we selected 20 configuration scenarios from the general algorithm configuration library AClib [Hutter *et al.*, 2014], covering four prominent AI problems: SAT, MIP, automated planning and automated machine learning. The algorithms and benchmarks included in AClib are well known and widely used in the automatic algorithm configuration literature. This section describes the target algorithms, benchmark instance sets and algorithm configurators, as well as our experimental protocol and execution environment.

3.1 Target algorithms

In our experiments, we used three solvers for propositional satisfiability (SAT), one automated planning system, one

Solver	Number of parameters			Problem
	Total	Real	Integers	
Clasp	75	7	30	SAT
Lingeling	322	0	186	SAT
SpToRiss	222	16	36	SAT
LPG	67	14	5	Planning
CPLEX	74	7	16	MIP
AutoWK	702	99	127	ML

Table 1: Target algorithms used in our experiments (from AClib).

solver for mixed integer programming (MIP) and one automated machine learning (ML) system. Details about their configuration spaces are provided in Table 1.

Three SAT solvers were selected based on their performance in the Configurable SAT Solver Challenge (CSSC) 2014 [Hutter *et al.*, 2017]. Lingeling [Biere, 2014] ranked first on the *industrial SAT+UNSAT* track and second on the *crafted SAT+UNSAT* track, Clasp [Gebser *et al.*, 2012] ranked first on the *crafted SAT+UNSAT* and *Random SAT+UNSAT* tracks and SpToRiss [Balint and Manthey, 2014] ranked second on the *Random SAT* track.

LPG [Gerevini *et al.*, 2003; 2008; 2011] is an automated planning system that has been successfully configured previously [Vallati *et al.*, 2011; Fawcett and Hoos, 2016] and is also available through AClib. We have also conducted preliminary experiments with FastDownward [Helmert, 2006; 2009], a prominent planner that showed excellent performance in planning competitions that has also been configured automatically [Fawcett *et al.*, 2011], but the parameter range reduction techniques we study in connection with RQ2 apply to numerical parameters only, of which FastDownward possesses only one. The MIP solver CPLEX from IBM is widely used in practice and in the literature [Hutter *et al.*, 2010].

AutoWK [Thornton *et al.*, 2013; Kotthoff *et al.*, 2017] is a prominent and widely used automated machine learning system based on, and distributed as part of, the WEKA machine learning and data mining workbench.

3.2 Benchmark instance sets

The AClib scenarios we considered are based on 14 sets of benchmark instances for SAT, MIP, planning and AutoML, as outlined in Table 3. For the SAT, MIP and planning scenarios, we used the training and testing sets provided in AClib [Hutter *et al.*, 2014]. For AutoML, the AClib scenarios involve minimisation of the cross-validated error rate. We follow the same procedure using the same folds.

3.3 Configurators

To investigate our research questions, we used three of the most prominent algorithm configurators: irace [López-Ibáñez *et al.*, 2016], GGA++ [Ansótegui *et al.*, 2015] and SMAC [Hutter *et al.*, 2011] (see Section 2). In light of the considerable computational expense involved in configuration experiments, we focused our investigation on SMAC (arguably, the most widely used of the three), and conducted smaller experiments to determine to which degree similar results could be obtained for the other two configurators.

Objective	Budget $[s]$	Cutoff $[s]$	Algorithm	Benchmark
PAR10	172800	300	Clasp	CF LABS UNSAT
			Lingeling	CF LABS UNSAT
			SpToRiss	CF LABS UNSAT
			LPG	Depots Satellite Zenotravel
		10000	CPLEX	CLS COR-LAT RCW2 REG200
error-rate	108000	9000	AutoWK	CAR GC WF WQW

Table 2: Configuration scenarios used in our experiments (from AClib)

In the beginning of each configuration run, irace evaluates the run-time of the target algorithm to evaluate the length of each iteration and the number of possible iterations given the time budget specified by the user. Depending on the evaluated run-time, irace refused to run on some of our scenarios, especially when starting from a random configuration, and ran over the time limit for many others. After discussing with the developers to make sure that it was not caused by our design choices, we decided to terminate irace after five times the allowed overall time budget; as a result, some results for irace are based on much larger time budgets than those for the other configurators, ranging from 110% of the time budget for LPG scenarios to 300% or more for SparrowToRiss.

Following recommendations of its developers, GGA++ was run on eight CPU cores and sometimes exceeded the given time limit.It is important to note that even though we specified the same wall-clock limit for all configurators, they did not use the same amount of CPU time. Therefore, it is problematic to compare performance between configurators; however, such comparisons are not required for answering our research questions.

3.4 Experimental protocol

All configuration scenarios were run according to the setup described in AClib and described in Table 2. The considered configuration objective is the minimization of PAR10 (average running time of the target algorithm, with the timed-out runs counted as ten times the cutoff time) for SAT, MIP and automated planning, and of the cross-validated error rate for AutoML. We ran each configurator independently 25 times on each scenario and evaluated the 25 resulting parameter configurations on our training and testing sets.

Benchmark	Instances		Description	Reference	Problem
	Train	Test			
CF	298	301	produced by a CNF fuzzing tool	[Brummayer *et al.*, 2010; Hutter *et al.*, 2017]	
LABS	350	350	low autocorrelation binary sequence problem converted into SAT instances	[Mugrauer and Balint, 2013; Hutter *et al.*, 2017]	SAT
UNSAT	299	249	unsatisfiable 5-SAT generated uniformly at random	[Hutter *et al.*, 2017]	
Depots	2000	2000	combines transportation and blocks problem	[Long and Fox, 2003]	
Satellite	2000	2000	control and observation scheduling of satellites	[Long and Fox, 2003]	Planning
Zenotravel	2000	2000	route planning	[Penberthy and Weld, 1994; Long and Fox, 2003]	
CLS	50	50	capacitated lot-sizing benchmark	[Atamtürk and Muñoz, 2004]	
COR-LAT	1000	1000	data for wildlife corridors in the Northern Rockies	[Gomes *et al.*, 2008]	
RCW2	495	495	MIP-encoded habitat preservation data for endangered red-cockaded woodpeckers	[Xu *et al.*, 2011; Ahmadizadeh *et al.*, 2010]	MIP
REG200	999	999	generated MIP problem instances	[Hutter *et al.*, 2010]	
CAR	1728	-	evaluation of cars		
GC	690	-	classifies people as good or bad credit risks	[Dua and Karra Taniskidou, 2017; Thornton *et al.*, 2013]	ML
WF	5000	-	generated waves		
WQW	4898	-	model wine quality based on physicochemical tests		

Table 3: Benchmark instance sets considered in our experiments (obtained from AClib).

Algorithm configurators are randomised, and their performance is known to vary substantially between multiple independent runs on the same scenario. To leverage this, it is common 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 he overall configuration process.

To capture the statistical variability of this standard protocol, we repeatedly sampled 8 runs uniformly at random and identified the best of these according to 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. We then compared the medians of these empirical distributions, using a one-sided Mann-Whitney U-test ($\alpha = 0.05$) to assess statistical significance of observed performance differences.

All experiments were performed on a computing cluster running CentOS on Dual 16-core 2.10GHz Intel Xeon E5-2683 CPUs with 40 MB cache and 94 GB RAM.

4 Impact of the default configuration

As mentioned previously, most configurable algorithms come with default parameter values that have been carefully chosen by the developer. Algorithm configuration procedures handle those default values differently. By default GGA++ ignores the default configuration specified by a given target algorithm. Although this behaviour can be changed by parameters, GGA++ creates its population randomly and optimizes from there. SMAC, on the other hand, uses the default configuration as one of the starting configurations for building the model that guides its search process. Finally, 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.

		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: 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 ($\alpha = 0.05$).

How much impact does the default configuration have on current configurators? We ran SMAC and irace with and without the default parameter values supplied in our AClib scenarios; then, we applied the protocol described in Section 3.4 to compare the resulting configurations (see Table 4). 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 1 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 interleaves the configurations determined using its random forest model with randomly chosen configurations. This is done to avoid stagnation of the model-based search process and limits the impact of a specific starting configuration. When given a limited time budget (as in all our experiments), irace uses the default configuration of the given target algorithm to estimate target algorithm running time. Consequently, when starting from a randomly chosen configuration, under which the target algorithm performs very badly, 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 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 out of 10 scenarios starting from a random configuration produces better results.

The benefit of a user-provided default configuration for a given target algorithm thus depends on the configuration procedure. While irace critically depends on meaningful default configurations, they do not consistently improve the performance of SMAC and are completely ignored by GGA++.

5 Search space reduction

Automatic algorithm configurators can be seen as searching the configuration space of a given target algorithm. Due to the various types of parameters, the conditions linking them and the effects of their interaction it can be hard to search the space of possible configurations. We will focus on integer- and real-valued parameters for our experiments, as their domains are often large intervals, reducing which can have a big impact on the size of a given configuration space.

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 $D_{k,sub}$. 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 varying in a logarithmic scale, we apply the reductions on 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:

$$D_{k,sub} = [0.1 \cdot d_k, 10 \cdot d_k]\,. \tag{R1}$$

Our second technique reduces the range to a tenth of D_k, centred around the default value:

$$D_{k,sub} = \left[d_k - \frac{d_{k,range}}{20}, d_k + \frac{d_{k,range}}{20}\right]. \tag{R2}$$

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

$$D_{k,red} = D_k \cap D_{k,sub} \tag{1}$$

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 $D_{k,sub}$ and we add values to $D_{k,red}$ 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 5: 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 considered algorithms configuration spaces? We applied both reductions to the six algorithms appearing in our configuration scenarios. As seen in Table 5, 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 large impact when configuring them. We note that the parameters of Clasp to which reduction is applied are almost all conditionally dependent 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.

6 Configuration 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 3.4 to compare the results obtained for the full configuration space and the space obtained using our two reduction techniques.

How do our reduction techniques impact the performance of SMAC? As seen in Table 6, 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 optimized configurations reached the quality of the default configuration (on testing data). For the five 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 occur over all AI problems and target algorithms we studied.

Do our observations generalize to other configuration approaches? 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 it. We kept three SAT, two planning, two MIP and two ML scenarios, resulting in 9 scenarios overall. The results for irace and GGA++ are shown in Table 7.

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

Table 6: 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 ($\alpha = 0.05$).

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 scenario. 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 Section 5). For SpToRiss on LABS, it is surprising that the Mann-Whitney test found evidence that the distribution for R1 is worse than R2 while 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 irace implements a mechanism that resembles our range reduction techniques by focusing sampling of configurations around combinations of parameter values known to give high performance. More precisely, when it generates a new configuration, it samples the values according to a Gaussian around the best known values and reduces the variance along the run to focus on promising parts of the configuration space.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, but search space reduction gives consistent improvements.

Can we leverage the information contained in the default by reducing the range of the parameters? The simple domain reduction techniques introduced here improve the efficacy of SMAC and GGA++ on over most of the configuration scenarios we considered, and produce improvements for target algorithms with a low number of conditionally dependent parameters when using irace. 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 Sparrow-ToRiss to those recently published for warm-starting SMAC [Lindauer and Hutter, 2018], we find that that 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.

Further investigation of our results. A possible concern arising from the way we reduce the search space is that with a strong reduction, we 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 8, 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 better configuration with R1 for Clasp, SpToRiss, LPG and AutoWK and with R2 for Lingeling. However, the results presented in Tables 6 and 7 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, still allowed us to reach better configurations on average. This indicates that by reducing the size of a given configuration space, it can become easier for existing configuration procedures to consistently find good local optima.

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.

			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**
SpToRiss	LABS	808.70	**803.03**	788.66	**787.47**	835.32	**804.31**	847.83
SpToRiss	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
LPG	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**
CPLEX	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**
AutoWK	WF	1.97	**0.230**	0.250	0.260	2.69	**2.40**	2.55

Table 7: 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 ($\alpha = 0.05$).

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 8: How many parameters of the best known configuration (testing set) take a value outside the reduced ranges (using R1 and R2, respectively)?

7 Conclusions and Future Work

In this work, we studied the current use of default parameter values in automated algorithm configuration and introduced a simple approach to exploit them more efficiently. Specifically, we investigated two research questions.

How much impact does the default configuration have 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 tends to be affected very differently by default settings: While SMAC only rarely benefits from good defaults, they are often crucial for irace.

Can we leverage the information contained in the default by reducing the range of the parameters? We studied this question based on two relatively straightforward techniques for reducing the ranges of numerical parameters of a given target algorithm, guided by default settings. Empirical results for well-known configuration scenarios for SAT, MIP, automated planning and automated machine learning clearly indicate that using these reduction technique, the state-of-the-art general-purpose configurators SMAC [Hutter *et al.*, 2011] and GGA++ [Ansótegui *et al.*, 2015] tend to find significantly better configurations within a given time budget. irace [López-Ibáñez *et al.*, 2016] 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 mostly depend on higher-level categorical choices, as for SpToRiss and AutoWK.

Obviously, our reduction techniques depend on the quality of the given default values and are targeting numerical parameters with wide ranges. 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. At the same time, many numerical parameters do have wide ranges, 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 and thus is often avoided by algorithm designers. Of course, reducing parameter ranges too aggressively will ultimately lead to degraded performance.

The results from our study indicate that default parameter settings often provide more information than is currently exploited in automated algorithm configuration. Further, significant improvements in performance can be achieved using simple, generic techniques for restricting the configuration space based on these defaults. 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 – *e.g.*, by using them as priors in sequential model-based optimisation, or as the basis for regularising the empirical performance models used by configurators such as SMAC.

References

[Ahmadizadeh *et al.*, 2010] K. Ahmadizadeh, B. N. Dilkina, C. P. Gomes, and A. Sabharwal. An empirical study of optimization for maximizing diffusion in networks. In *Proc. CP 2010*, pages 514–521, 2010.

[Ansótegui *et al.*, 2009] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proc. CP 2009*, pages 142–157, 2009.

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

[Atamtürk and Muñoz, 2004] A. Atamtürk and J. C. Muñoz. A study of the lot-sizing polytope. *Mathematical Programming*, 99(3):443–465, Apr 2004.

[Balint and Manthey, 2014] A. Balint and N. Manthey. Sparrowtoriss. In *Proc. SAT Competition 2014*, page 77–78, 2014.

[Biere, 2014] A. Biere. Yet another local search solver and lingeling and friends entering the sat competition 2014. 2014.

[Birattari *et al.*, 2010] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and iterated F-Race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. 2010.

[Brummayer *et al.*, 2010] R. Brummayer, F. Lonsing, and A. Biere. Automated testing and debugging of sat and qbf solvers. In *Theory and Applications of Satisfiability Testing – SAT 2010*, pages 44–57, 2010.

[Dua and Karra Taniskidou, 2017] D. Dua and E. Karra Taniskidou. UCI machine learning repository, 2017.

[Fawcett and Hoos, 2016] C. Fawcett and H. H. Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458, 2016.

[Fawcett *et al.*, 2011] C. Fawcett, M. Helmert, H. Hoos, E. Karpas, G. Röger, and J. Seipp. FD-Autotune: Domain-specific configuration using fast downward. In *Proc. the ICAPS Workshop, PAL 2011*, pages 13–20, 2011.

[Franzin *et al.*, 2018] A. Franzin, L. Pérez Cáceres, and T. Stützle. Effect of transformations of numerical parameters in automatic algorithm configuration. *Optimization Letters*, 12(8):1741–1753, 2018.

[Gebser *et al.*, 2012] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187-188:52 – 89, 2012.

[Gerevini *et al.*, 2003] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *J. Artif. Intell. Res.*, 20:239–290, 2003.

[Gerevini *et al.*, 2008] A. Gerevini, A. Saetti, and I. Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artif. Intell.*, 172:899–944, 2008.

[Gerevini *et al.*, 2011] A. Gerevini, A. Saetti, and I. Serina. An empirical analysis of some heuristic features for planning through local search and action graphs. *Fundam. Inform.*, 107(2-3):167–197, 2011.

[Gomes *et al.*, 2008] C. P. Gomes, W.-J. van Hoeve, and A. Sabharwal. Connections in networks: A hybrid approach. In *Proc. CPAIOR 2008*, pages 303–307, 2008.

[Helmert, 2006] M. Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[Helmert, 2009] M. Helmert. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6):503–535, 2009.

[Hoos, 2012] H. H. Hoos. Programming by optimization. *Commun. ACM*, 55(2):70–80, 2012.

[Hutter *et al.*, 2009] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res.*, 36:267–306, 2009.

[Hutter *et al.*, 2010] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proc. CPAIOR 2010*, pages 186–202, 2010.

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

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

[Hutter *et al.*, 2017] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. H. Hoos, and K. Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artif. Intell.*, 243:1–25, 2017.

[Kotthoff *et al.*, 2017] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18:25:1–25:5, 2017.

[Lindauer and Hutter, 2018] M. Lindauer and F. Hutter. Warmstarting of model-based algorithm configuration. In *Proc. AAAI-18, IAAI-18, and EAAI-18*, pages 1355–1362, 2018.

[Long and Fox, 2003] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res.*, 20:1–59, 2003.

[López-Ibáñez *et al.*, 2016] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Mugrauer and Balint, 2013] F. Mugrauer and A. Balint. Sat encoded low autocorrelation binary sequence (labs) benchmark description. In *Proc. SAT Competition 2013*, page 117–118, 2013.

[Penberthy and Weld, 1994] J. S. Penberthy and D. S. Weld. Temporal planning with continuous change. In *Proc. AAAI-94, Volume 2.*, pages 1010–1015, 1994.

[Pushak and Hoos, 2018] Y. Pushak and H. H. Hoos. Algorithm configuration landscapes: More benign than expected? In *Proc. PPSN 18*, pages 271–283, 2018.

[Thornton *et al.*, 2013] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD, KDD 2013*, pages 847–855, 2013.

[Vallati *et al.*, 2011] M. Vallati, C. Fawcett, A. Gerevini, H. H. Hoos, and A. Saetti. Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proc. RCRA 2011*, pages 111–123, 2011.

[Xu *et al.*, 2011] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proc. RCRA 2011*, pages 16–30, 2011.