Model-Based Algorithm Configuration with Default-Guided Probabilistic Sampling

Marie Anastacio[1] and Holger Hoos[1,2]

[1] Leiden University, Leiden, The Netherlands
[2] University of British Columbia, Vancouver, Canada

m.i.a.anastacio@liacs.leidenuniv.nl, hh@liacs.nl

Abstract. In recent years, general-purpose automated algorithm configuration procedures have enabled impressive improvements in the state of the art in solving a wide range of challenging problems from AI, operations research and other areas. To search vast combinatorial spaces of parameter settings for a given algorithm as efficiently as possible, the most successful configurators combine techniques such as racing, estimation of distribution algorithms, Bayesian optimisation and model-free stochastic search. Two of the most widely used general-purpose algorithm configurators, SMAC and irace, can be seen as combinations of Bayesian optimisation and racing, and of racing and an estimation of distribution algorithm, respectively. Here, we propose a first approach that combines all three of these techniques into one single configurator, while exploiting prior knowledge contained in expert-chosen default parameter values. We demonstrate significant performance improvements over irace and SMAC on a broad range of running time optimisation scenarios from AClib.

1 Introduction

For many combinatorial problems, the performance of state-of-the-art algorithms critically depends on parameter settings [24]. Traditionally, algorithm developers have manually tuned these parameters to find a configuration that performs well across a range of benchmarks. However, this manual configuration process tends to be tedious and inefficient. For the past decade, automated methods for configuring those algorithms have been broadly established as an effective alternative to this manual approach. Prominent examples of general-purpose automated algorithm configurators include irace [31], ParamILS [24], GGA/GGA++ [4, 3] and SMAC [23]. These fully automated configuration procedures are based on advanced machine learning and optimisation methods. Many examples of prominent and challenging computational problems on which automated configurators are strikingly effective can be found in the literature; these include propositional satisfiability (SAT) (see, *e.g.*, [25]), mixed integer programming (MIP) (see, *e.g.*, [22]), automated planning (see, *e.g.*, [12]) and supervised machine learning (ML) (see, *e.g.*, [36, 27]).

The availability of those highly 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) [21]. 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 broadly achieve better performance. However, the size of the combinatorial configuration spaces encountered in this context grows exponentially with the number of exposed parameters. To search these spaces effectively, state-of-the-art algorithm configuration procedures combine sophisticated methods, such as racing, estimation of distribution algorithms, Bayesian optimisation and model-free stochastic local search.

In particular, SMAC and irace can be seen as combinations of Bayesian optimisation and racing, and of racing and an estimation of distribution algorithm, respectively. We propose a first approach that combines all three of these techniques into a single general-purpose automated algorithm configuration procedure we call SMAC+PS. Based on work exploring the exploitation of prior knowledge contained in expert-chosen default parameter values [2], we bias the sampling of new configurations towards these values using truncated normal distributions centred on them. To test our new configuration approach, we applied it to a broad range of widely used running time minimisation scenarios from AClib [26]. As we will report later, we obtained significant performance improvements over irace and SMAC, the two state-of-the-art configurators SMAC+PS is based on. Our sampling method improves over SMAC in more than two thirds, and over irace in three quarters of the benchmark scenarios we considered.

The remainder of this paper is structured as follows: After a brief introduction to the algorithm configuration problem and several current state-of-the-art configurators (Section 2), we introduce our approach and the underlying motivation (Section 3). Next, we describe the experimental protocol we used to evaluate SMAC+PS (Section 4), followed by our results (Section 5), some general conclusions and a brief outlook on future work(Section 6).

2 Automated Algorithm Configuration

The algorithm configuration problem (see, *e.g.*, [20]) 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; and 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.

Configuration spaces may contain 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. As a special case, Boolean parameters typically activate or deactivate optional algorithm components. Nu-

merical, integer- or real-valued parameters are often used to quantitatively control performance-relevant aspects of a given target algorithm or heuristic, such as the strength of a diversification mechanism. Some parameters may conditionally depend on others, in that they are only active for certain values of other (often categorical or Boolean) parameters.

In recent years, much work has been done on automatic algorithm configuration, resulting in several high-performance, general-purpose automatic algorithm configurators, based on different approaches. The most prominent configurators include irace [8, 31], GGA++ [3] and SMAC [23].

Among the main challenges of automated algorithm configuration are the time required to evaluate the performance of a configuration and the size of the configuration space. To minimise the number of time-consuming evaluations, SMAC and GGA++ are using an empirical performance model to predict how well a configuration will perform without running the target algorithm. irace leverages racing methods based on a statistical test to discard configurations when they are unlikely to perform well. Approaches to terminate less promising runs before they finish have shown great potential (see *e.g.*, [11, 32]), but can only be applied to any-time algorithms, thus excluding many prominent algorithms for decision problems, such as SAT.

To handle the combinatorial space of possible configurations, a key component of any algorithm configurator is the way it samples the parameter values to be evaluated. irace uses a truncated normal distribution around the best known values to sample promising configurations. GGA++ uses its surrogate model to genetically engineer part of the offspring when creating a new generation of configurations. SMAC generates two sets of random configurations; one of these is used for diversification and the other is improved by local search, using an empirical performance model and an expected improvement criterion.

Each of the state-of-the-art configurators we mentioned (and indeed, any such configurator we are aware of) combines several fundamental mechanisms into a sophisticated search procedure.

3 SMAC+PS

Our recent study [2] has demonstrated that algorithm developers tend to provide default parameter configurations that have been chosen to perform reasonably well across a broad range of problem instances.

We propose a new method to exploit the prior knowledge contained in such expert-chosen default parameter values. Our method is based on combining the widely used and freely available, state-of-the-art general-purpose automated algorithm configurator, SMAC, with the sampling approach of irace. We call our new configurator SMAC+PS, as it combines the sequential model-based approach of SMAC with probabilistic sampling. Before explaining SMAC+PS, we briefly review SMAC.

3.1 The SMAC configurator

SMAC is based on a sequential model-based optimisation approach (also known as Bayesian optimisation). It creates a random forest model to predict the performance of the given target algorithm for arbitrary parameter configurations, based on performance data collected from specific target algorithm runs. In each iteration, SMAC selects a candidate configuration and runs it on some problem instances from the given training set. The random forest model is then updated with the performance observed in this run. This process continues until a given time budget (usually specified in terms of wall-clock time) has been exhausted.

The step that is of interest in the context of this work is the way in which SMAC selects new configurations. This is done using two different mechanisms. On the one hand, to exploit current knowledge from its random forest model, SMAC performs local search from many randomly generated configurations, attempting to maximise expected improvement over the best configuration encountered so far. On the other hand, to diversify, explore unknown parts of the search space and avoid being misguided by its model, SMAC selects new configurations uniformly at random from the entire (usually vast) configuration space. SMAC alternates between these two mechanisms, so that overall, each of them contributes half of the new target algorithm configurations that are evaluated.

While running random configurations helps SMAC to balance exploration and exploitation, those runs can also waste substantial time and resources. Especially when dealing with a set of challenging training instances, randomly chosen configurations can easily time out and thus provide little useful information.

3.2 Extension with probabilistic sampling

As shown in previous work, simply reducing the ranges of numerical parameters around the given default values can lead to significant performance improvements for SMAC [2]. This was inspired by two other recent findings; one from a study of the dependence of algorithm performance on the setting of numerical parameters, which showed that the configuration landscapes tend to be surprisingly benign [35] and suggests that limited manual tuning may lead to useful information about promising parameter values; and one indicating that SMAC can be sped up considerably by starting it from a performance model learned on another set of benchmark instances [29], thus suggesting that knowledge regarding high-quality configurations can be transferred between sets of problem instances. 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.

Building on those findings, our intuition was 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 very bad parameter settings. Here, we propose a principled approach for exploiting the prior knowledge contained in the default value of parameters, by including probabilistic sampling in the context of sequential model-based optimisation. As we

Algorithm 1 SMAC+PS
C: configuration space. c_d: the default configuration.
C_{rand}, C_{prom} and C_{new}: sets of configurations.
$\mathcal{R}$: target algorithm runs performed. $\mathcal{M}$: performance model.

```
1: inc ← c_d
2: R ← run(c_d)
3: while Budget not exhausted do
4:     M ← update(M, R)
5:     C_prom ← uniform_sample_configurations(C)
6:     C_prom ← local_search(M, C_prom)
7:     C_rand ← normal_sample_configurations(C)
8:     C_new ← interleave(C_rand, C_prom)
9:     inc ← intensify(C_new, inc)
10: end while
11: Return inc
```

will demonstrate, this can yield significant improvement for general-purpose algorithm configuration. 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 1 outlines our new configuration method, which we refer to as SMAC+PS, at a high level. SMAC+PS differs from SMAC only in the sampling distribution used in line 7; further details of SMAC can be found in the original paper [23]. 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). 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", after applying a log base ten 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. 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 of counterbalancing this effect, and preliminary experiments in the appendix provide some evidence that this is sufficient for preventing major performance degradation compared to SMAC.

Our probabilistic sampling approach is inspired by estimation of distribution algorithms, which are known to provide an effective way for leveraging prior

Table 1: Benchmark instance sets from AClib considered in our experiments.

Problem	Benchmark	Instances		Reference
		Train	Test	
SAT	CF	298	301	[9, 25]
	LABS	350	350	[33, 25]
	UNSAT	299	249	[25]
Planning	Depots	2000	2000	[30]
	Satellite	2000	2000	[30]
	Zenotravel	2000	2000	[34, 30]
MIP	CLS	50	50	[5, 22]
	COR-LAT	1000	1000	[18, 22]
	RCW2	495	495	[38, 1]
	REG200	999	999	[28, 22]

knowledge when solving complex optimisation problems [19]. It is also conceptually related to the approach taken by irace, which – unlike SMAC – does not make use of an empirical performance model mapping parameter configurations to performance values, but uses the best known configurations as a basis for estimating where promising parameters may be located. Indeed, irace uses an intensification mechanism based on sampling new values for numerical parameters from truncated normal distributions, in combination with a racing mechanism for updating the incumbent configuration and the location of the sampling distributions. We note that the probabilistic sampling process we use in SMAC is simpler, since it keeps the sampling distributions fixed throughout the configuration process. We decided on this design in order to evaluate to which extent 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 Experimental setup

To evaluate SMAC+PS (available at `ada.liacs.nl/projects/smacps`), we selected 16 configuration scenarios for running time minimisation from the general algorithm configuration library, AClib [26], covering three prominent and widely studied combinatorial problems: propositional satisfiability (SAT), AI planning and mixed integer programming (MIP). The algorithms and benchmarks included in AClib are well known and widely used in the automatic algorithm configuration literature, and running time minimisation is an important and widely studied special case of automated algorithm configuration. This section describes the benchmark instance sets and target algorithms, as well as our experimental protocol and execution environment.

4.1 Benchmark instance sets

The AClib scenarios we considered are based on 10 sets of randomly generated and real-world SAT, MIP and AI planning instances [26]. We used the training and testing sets provided by AClib, as shown in Table 1.

Table 2: Target algorithms used in our experiments (as provided by AClib).

Problem	Solver	Number of parameters			
		Total	Real	Integers	Conditionals
SAT	Clasp	75	7	30	55
	Lingeling	322	0	186	0
	SpToRiss	222	16	36	176
Planning	LPG	67	14	5	22
MIP	CPLEX	74	7	16	4

Our three SAT benchmarks originate from the configurable solver SAT challenge [25]: a set of instances generated by a CNF fuzzing tool (CF) [9], a set of low auto-correlation binary sequence problems converted into SAT (LABS) [33], and a set of 5-SAT problems generated uniformly at random from which only unsatisfiable instances have been kept (UNSAT).

Our three automated planning benchmarks originate from the third international planning competition [30]; a set that combines transportation and blocks problem (Depots), a set about the control and observation scheduling of satellites (Satellite), and a set of route planning problem (Zenotravel) [34].

Three of our four MIP benchmarks originate from a study on MIP solver configuration [22]: a set of capacitated lot-sizing benchmark (CLS) [5], a set of MIP problem instances generated with the combinatorial auction test suite [28], and a set of real-life data for wildlife corridors for grizzly bears in the Northern Rockies [18]. The fourth benchmark stems from work on combining algorithm configuration and selection [38]: a set of MIP-encoded habitat preservation data for the endangered red-cockaded woodpecker [1].

4.2 Target algorithms

In our experiments, we used five prominent solvers for SAT, AI planning and MIP (see Table 2). Our SAT solvers were selected based on their performance in the Configurable SAT Solver Challenge (CSSC) 2014 [25]: Lingeling [7] ranked first on the *industrial SAT+UNSAT* track and second on the *crafted SAT+UNSAT* track, Clasp [14] first on the *crafted SAT+UNSAT* and *Random SAT+UNSAT* tracks and SparrowToRiss (SpToRiss) [6] second on the *Random SAT* track. For automated planning, we selected LPG [15–17], as it has been successfully configured previously [37, 13] and is also available through AClib. For MIP, we chose IBM's CPLEX solver, as it is widely used in practice and has shown great potential for performance improvement through automated configuration [22].

4.3 Configurators

We compare our approach to SMAC [23] (SMAC3 v0.10.0), the state-of-the-art general-purpose algorithm configurator on which SMAC+PS is based. As a second baseline, we chose irace with capping [10, 31] (v3.3.2238:2239), another

Table 3: Configuration scenarios from AClib used in our experiments.

Problem	Benchmark	Algorithm	Cutoff [s]	Budget [s]
SAT	CF LABS UNSAT	Clasp	300	172800
	CF LABS UNSAT	Lingeling		
	CF LABS UNSAT	SpToRiss		
Planning	Depots Satellite Zenotravel	LPG		
MIP	CLS COR-LAT RCW2 REG200	CPLEX	10000	

widely used, state-of-the-art configurator, which provided some inspiration for our sampling method. We did not compare to GGA++ [3], since it is not publicly available and, unlike the other configurators we considered, appears to critically require parallel execution.

In configuration scenarios where the budget is total CPU time, irace evaluates the running time of the target algorithm to estimate the length of each iteration and the number of possible iterations given the time budget specified by the user. In cases where the given cutoff time is significantly larger than the typical running time of the target algorithm, irace has difficulty estimating how many runs of the target-algorithm can be performed within the budget and may either refuse to run, due to insufficient budget, or exceed the budget. This affected 4 of the 16 scenarios we studied, namely the AClib MIP scenarios, but even if qualitatively different results were obtained when modifying those scenarios, this would not affect the overall conclusions drawn from our experiments (Section 5). We decided to terminate irace after 150% of the allowed overall time budget – that is, in our case, 3 instead of 2 days of configuration time; as a result, some results for irace are based on larger time budgets than those of SMAC. In two scenarios, too few of the configuration runs finished within this time budget, and we do not report results for irace in those cases (see Section 4.5).

4.4 Configuration scenarios

All configuration scenarios were run according to the setup defined in AClib, including default configurations, and making sure that the configurators used those defaults (see Table 3). As a configuration objective, we used minimisation of PAR10 (average running time of the target algorithm, with timed-out runs counted as ten times the cutoff time).

4.5 Evaluation protocol

We ran each configurator independently 24 times on each scenario and evaluated the 24 resulting parameter configurations on our training and testing sets.

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 the 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 the statistical significance of observed performance differences.

For irace, when the estimated running time of the algorithm was too long and it refused to run within the given time budget, we decided to apply 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 those cases, when the random seed (ranging from 1 to 24) leads to 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 positive bias on the irace results for CPLEX on CLS, where 19 runs finished 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 omitted the results from our analysis, as application of the standard protocol would lead to extreme distortions from realistically achievable performance.

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

5 Results

We now present the results of the experiments described in Section 4.

5.1 Comparison based on median PAR10 scores

We compare the performance obtained for the configuration scenarios we studied, following the protocol described in Section 4.5, which produces statistics over the way state-of-the-art configurators are commonly used in practice. Table 4 shows the median PAR10 values we obtained; the missing results for CPLEX on RCW2 and REG200 are due to the fact that 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 reached a statistically

Table 4: Results for SMAC, SMAC+PS 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 SMAC+PS and the two baselines; ✓ if it is better, ✗ if not; parentheses indicate that the difference is not statistically significant.

Solver	Benchmark	Default	SMAC	irace	SMAC+PS	≻ 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**	✓	✓

significantly lower median PAR10 score for 10 out of the 16 configuration scenarios we studied, and irace outperformed SMAC for 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 configurations (CPLEX on COR-LAT).

Comparing SMAC+PS against SMAC, which is not only the stronger of our two baselines, but also served as the starting point for our new configuration procedure, we notice that for 11 of the 16 scenarios, SMAC+PS 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, SMAC+PS 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 SMAC+PS outperforms irace.

Overall, these results indicate clearly that SMAC+PS represents a significant improvement over both baselines, and hence an advance in the state of the art in automated algorithm configuration for running time minimisation.

As SMAC+PS relies on the assumption that the provided default values are well-chosen, we performed some preliminary experiments to study its robustness to misleading default values. The results, available in the appendix, show no significant loss in performance compared to SMAC.

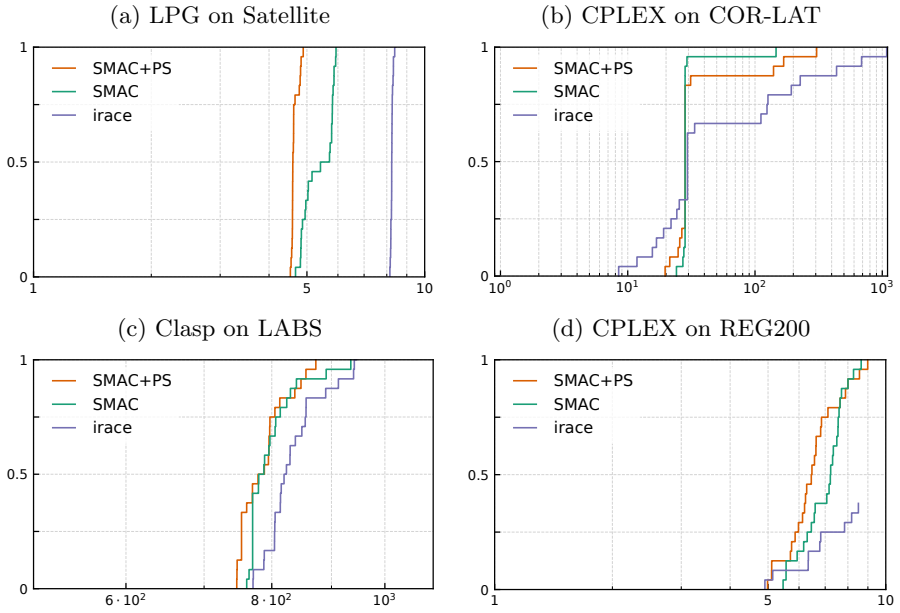

Fig. 1: Cumulative distribution functions for PAR10 scores (in CPU seconds, x-axis) over independent configurator runs on the testing set.

5.2 Distributions of PAR10 scores over multiple configurator runs

To examine the performance of SMAC, irace and SMAC+PS in more detail, we studied the empirical cumulative distribution functions over individual configurator runs (without applying the standard protocol) – see Figure 1. As we minimise PAR10 scores, better performance is indicated by CDFs closer to the top left corner of the plots. We show results for one scenario on which SMAC+PS performs better than the two other configurators (Figure 1a), one scenario on which irace performs better than SMAC+PS and SMAC (Figure 1b), one in which SMAC and SMAC+PS are tied (Figure 1c), as well as one on which a large number of irace did not terminate successfully (Figure 1d).

LPG on Satellite (Figure 1a) is a case in which the difference between the CDFs for SMAC and SMAC+PS is particularly pronounced. In this scenario, SMAC+PS clearly dominates the two other configurators.

CPLEX on COR-LAT (Figure 1b) is a case in which irace performs better than the two other configurators. SMAC and SMAC+PS give rise to similar performance distributions. irace has an edge, reinforced by the standard protocol, which leverages the left tail of the performance distributions.

Clasp on LABS (Figure 1c), a scenario on which SMAC and SMAC+PS are statistically tied, looks qualitatively similar, except that the difference in the left tail between SMAC and SMAC+PS is too small to be reliably exploitable using the standard protocol. irace, on the other side, is probabilistically dominated on this scenario.

Figure 1d shows a scenarios on which irace terminated prematurely (as described in Section 4). However, looking at the partial CDFs for those configurator runs that were completed successfully, there is no reason to expect that irace would have performed significantly better than SMAC and SMAC+PS.

6 Conclusion and Future Work

Building on previous work on exploiting the prior knowledge included in expert-chosen default parameter values in automated algorithm configuration [2], we propose a simple, yet effective way to probabilistically bias the sequential model-based algorithm configurator SMAC [26] towards a given default configuration. To do so, we replaced its uniform random sampling mechanism by a probabilistic sampling approach for numerical parameters. While this may seem like a relatively minor change, we note that it represents a substantial conceptual departure from SMAC, whose uniform random sampling mechanism ensures the diversification of the search process, while our new mechanism intensifies the search based on expert-defined default parameter values.

We evaluated the resulting procedure, dubbed SMAC+PS, against two widely used and freely available state-of-the-art general-purpose algorithm configurators, SMAC and irace [31]. For this comparison, we used 16 running time optimisation scenarios from AClib, a widely used library of benchmarks for automated algorithm configuration [26]. The scenarios we selected cover well-known algorithms and benchmark instance sets from three widely studied combinatorial problems – propositional satisfiability (SAT), mixed integer programming (MIP) and AI planning.

We found that SMAC+PS 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 automated algorithm configuration for running time minimisation. Whether similar results can be obtained for different performance metrics is an open question.

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 [35], which suggests that sampling around known good parameters values provides an efficient way towards finding new good values, an assumption also leveraged by irace.

In future work, we plan to extend our approach with a mechanism for adapting the median and variance of the distributions used for sampling values for each parameter, such that we overcome more effectively the prior knowledge from poorly chosen defaults, while good defaults are still exploited from the beginning of the configuration process. Such a mechanism also offers several avenues for better exploiting prior knowledge from the default values of categorical parameters. Finally, our work on SMAC+PS opens an interesting path towards richer mechanisms for allowing algorithm designers to express prior knowledge about parameter values.

References

1. Ahmadizadeh, K., Dilkina, B.N., Gomes, C.P., Sabharwal, A.: An empirical study of optimization for maximizing diffusion in networks. In: Proc. CP 2010. pp. 514–521 (2010)
2. Anastacio, M., Luo, C., Hoos, H.: Exploitation of default parameter values in automated algorithm configuration. In: Workshop Data Science meets Optimisation (DSO), IJCAI 2019 (aug 2019)
3. Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., Tierney, K.: Model-based genetic algorithms for algorithm configuration. In: Proc. IJCAI 2015. pp. 733–739 (2015)
4. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Proc. CP 2009. pp. 142–157 (2009)
5. Atamtürk, A., Muñoz, J.C.: A study of the lot-sizing polytope. Mathematical Programming **99**(3), 443–465 (Apr 2004)
6. Balint, A., Manthey, N.: Sparrowtoriss. In: Proc. SAT Competition 2014. p. 77–78 (2014)
7. Biere, A.: Yet another local search solver and lingeling and friends entering the sat competition 2014 (2014)
8. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: An overview. In: Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336 (2010)
9. Brummayer, R., Lonsing, F., Biere, A.: Automated testing and debugging of sat and qbf solvers. In: Theory and Applications of Satisfiability Testing – SAT 2010. pp. 44–57 (2010)
10. Cáceres, L., López-Ibánez, M., Hoos, H., Stützle, T.: An experimental study of adaptive capping in irace. In: Proc. LION 11. LNCS, vol. 10556, pp. 235–250. Springer (2017)
11. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Yang, Q., Wooldridge, M.J. (eds.) IJCAI 2015. pp. 3460–3468. AAAI Press (2015)
12. Fawcett, C., Helmert, M., Hoos, H., Karpas, E., Röger, G., Seipp, J.: FD-Autotune: Domain-specific configuration using fast downward. In: Proc. the ICAPS Workshop, PAL 2011. pp. 13–20 (2011)
13. Fawcett, C., Hoos, H.H.: Analysing differences between algorithm configurations through ablation. Journal of Heuristics **22**(4), 431–458 (2016)
14. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187-188**, 52 – 89 (2012)
15. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in LPG. J. Artif. Intell. Res. **20**, 239–290 (2003)
16. Gerevini, A., Saetti, A., Serina, I.: An approach to efficient planning with numerical fluents and multi-criteria plan quality. Artif. Intell. **172**, 899–944 (2008)
17. Gerevini, A., Saetti, A., Serina, I.: An empirical analysis of some heuristic features for planning through local search and action graphs. Fundam. Inform. **107**(2-3), 167–197 (2011)
18. Gomes, C.P., van Hoeve, W.J., Sabharwal, A.: Connections in networks: A hybrid approach. In: Proc. CPAIOR 2008. pp. 303–307 (2008)
19. Hauschild, M., Pelikan, M.: An introduction and survey of estimation of distribution algorithms. Swarm and Evolutionary Computation **1**(3), 111–128 (Sep 2011)

20. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 37–71. Springer (2012)
21. Hoos, H.H.: Programming by optimization. Commun. ACM **55**(2), 70–80 (2012)
22. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Proc. CPAIOR 2010. pp. 186–202 (2010)
23. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. LION 5. pp. 507–523 (2011)
24. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
25. Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H.H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). Artif. Intell. **243**, 1–25 (2017)
26. Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M.T., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Aclib: A benchmark library for algorithm configuration. In: Proc. LION 8. LNCS, vol. 8426, pp. 36–40 (2014)
27. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. Journal of Machine Learning Research **18**, 25:1–25:5 (2017)
28. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a universal test suite for combinatorial auction algorithms. In: Jhingran, A., Mackie-Mason, J., Tygar, D.J. (eds.) Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00), Minneapolis, MN, USA, October 17-20, 2000. pp. 66–76. ACM (2000)
29. Lindauer, M., Hutter, F.: Warmstarting of model-based algorithm configuration. In: Proc. AAAI-18, IAAI-18, and EAAI-18. pp. 1355–1362 (2018)
30. Long, D., Fox, M.: The 3rd international planning competition: Results and analysis. J. Artif. Intell. Res. **20**, 1–59 (2003)
31. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
32. Luo, C., Hoos, H.H., Cai, S., Lin, Q., Zhang, H., Zhang, D.: Local search with efficient automatic configuration for minimum vertex cover. In: IJCAI-19. pp. 1297–1304. International Joint Conferences on Artificial Intelligence Organization (2019)
33. Mugrauer, F., Balint, A.: Sat encoded low autocorrelation binary sequence (labs) benchmark description. In: Proc. SAT Competition 2013. p. 117–118 (2013)
34. Penberthy, J.S., Weld, D.S.: Temporal planning with continuous change. In: Proc. AAAI-94, Volume 2. pp. 1010–1015 (1994)
35. Pushak, Y., Hoos, H.H.: Algorithm configuration landscapes: More benign than expected? In: Proc. PPSN 18. pp. 271–283 (2018)
36. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In: The 19th ACM SIGKDD, KDD 2013. pp. 847–855 (2013)
37. Vallati, M., Fawcett, C., Gerevini, A., Hoos, H.H., Saetti, A.: Automatic generation of efficient domain-optimized planners from generic parametrized planners. In: Proc. RCRA 2011. pp. 111–123 (2011)
38. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In: Proc. RCRA 2011. pp. 16–30 (2011)

Appendix: Robustness to misleading default values

To obtain better insights into the robustness of our approach to misleading default values, we generated random configurations until we found one that performed worse than the default, but produced time-outs on fewer than a third of the training instances from our three SAT benchmarks. Then, we repeated a few configuration experiments from Section 5.1, using these new, misleading default configurations (using the same protocol as described earlier). The results of this experiment are shown in Table 5.

Table 5: Results for SMAC and SMAC+PS when given a default generated to be misleading. The numbers shown are median PAR10 scores in CPU seconds; best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$).

Solver	Benchmark	Misleading default	SMAC	SMAC+PS
SpToRiss	CF	650.27	246.94	**227.08**
	UNSAT	165.80	**1.49**	1.51
	LABS	966.75	822.42	**814.75**