Algorithm Configuration Landscapes:
More Benign than Expected?

Yasha Pushak[1] and Holger Hoos[2,1]

[1] Department of Computer Science, The University of British Columbia, Vancouver, Canada
[2] LIACS, Universiteit Leiden, The Netherlands

ypushak@cs.ubc.ca, hh@liacs.nl

Abstract Automated algorithm configuration procedures make use of powerful meta-heuristics to determine parameter settings that often substantially improve the performance of highly heuristic, state-of-the-art algorithms for prominent $\mathcal{NP}$-hard problems, such as the TSP, SAT and mixed integer programming (MIP). These meta-heuristics were originally designed for combinatorial optimization problems with vast and challenging search landscapes. Their use in automated algorithm configuration implies that algorithm configuration landscapes are assumed to be similarly complex; however, to the best of our knowledge no work has been done to support or reject this hypothesis. We address this gap by investigating the response of varying individual numerical parameters while fixing the remaining parameters at optimized values. We present evidence that most parameters exhibit uni-modal and often even convex responses, indicating that algorithm configuration landscapes are likely much more benign than previously believed.

1 Introduction

Automated algorithm configuration procedures are able to find configurations that are often substantially better than expert-determined default settings. Current methods are heavily-based on meta-heuristics (such as ParamILS [12], an iterated local search procedure; GGA [2], a gender-based genetic algorithm, and SMAC [9], a model-based stochastic search algorithm), which are typically used to solve $\mathcal{NP}$-hard combinatorial optimization problems with complex search landscapes. To the best of our knowledge, the properties of the algorithm configuration search landscapes have not been investigated so far. In this work, we conduct such an investigation, focusing on numerical parameters, which prominently occur in many algorithm configuration scenarios, and provide evidence that these configuration landscapes are more benign than one might assume.

As a motivating example, consider the problem of configuring a stochastic local search algorithm A with a numerical parameter p that controls the tradeoff between intensification and diversification. Let us assume that low values result in low intensification, and high values in high intensification. Intuitively, one

would expect a single optimal setting to exist for this parameter; for lower values of p, the performance of A would deteriorate due to insufficient diversification (resulting in stagnation behaviour), and for higher values of p, insufficient intensification would degrade the performance of A. Therefore, the response of A's performance to p would be uni-modal, perhaps even convex. We hypothesize that many – perhaps: most – numerical parameters have similar characteristics.

Specifically, in this work, we investigate two research questions regarding the way the performance of a given algorithm depends on its parameter settings. **RQ 1. When all other parameters are fixed, are the responses of individual numerical parameters uni-modal or convex; if so, how often?** Here, the response of a parameter p refers to the function mapping values of p to the performance of the given algorithm. In the context of automated configuration of a given target algorithm A, the performance of A is assessed (and optimized) on a set I of problem instances. Following many state-of-the-art configurators [9,12,14], we asses the performance of A using PAR10 on I, *i.e.*, mean running time with timed-out runs counted at 10 times their running time cutoff. Ideally, the response of A to its parameters would be identical for all instances in I; however, we cannot assume that this will always be the case. This gives rise to our second research question: **RQ 2. When all other parameters are fixed, are the responses of individual numerical parameters uni-modal or convex on individual instances; if so, how often?** To obtain robust estimates for the performance of A on each $i \in I$, we took medians over 10 independent runs per instance. We note that the aggregation of performance over a set of instances (as in RQ 1) could lead to more complex parameter responses – *i.e.*, a negative answer to RQ 1 – even if the responses on individual instances were benign. However, more likely, aggregation should have a smoothing effect on the parameter responses, so that a negative answer to RQ 2 might still be consistent with a positive answer to RQ 1.

The answers to these research questions matter, because existing configurators make only weak assumptions about the landscapes they search. Indeed, the only assumption made by well-known, high-performance configurators, such as SMAC [9], ParamILS [12] and GGA [2], is that the performance of one configuration is likely to be correlated with the performance of nearby configurations. Of the configurators we consider here, only irace [14] (by the nature of the generative probabilistic model it uses to sample promising configurations) assumes a smooth response for numerical parameters. Additional structural insights, such as uni-modality or convexity of individual parameter responses, could at least in theory be exploited to substantially improve configurator performance.

In the literature on combinatorial optimization and meta-heuristics, landscape analysis is a well-established topic. Two particularly prominent approaches are based on the analysis of fitness-distance correlation and of landscape correlation functions (see, *e.g.*, Chapter 5 of [8] and the references therein). When used in empirical work for the characterization of landscapes, both approaches assess global landscape properties, while our work is focused on local properties. Furthermore, to yield reasonably accurate results, they both require large sets

of samples from a given landscape, which, in the context of algorithm configuration, are expensive to obtain (every sample involves many runs of the given target algorithm, one for each problem instance in the given training set.) Finally, both approaches require normalization to deal with parameters whose domains show large differences in range or number of values (for discrete parameters), as frequently encountered in typical algorithm configuration scenarios, and have difficulties dealing with integer-valued parameters with small ranges (such as `BACKBONE_TRIALS` for LKH [7] considered in our study).

Somewhat related to our work, in the algorithm configuration literature, there has been a recent focus on analyzing parameter importance, *i.e.*, the degree to which individual parameters affect the performance of a given algorithm. Importance analysis approaches such as ablation analysis [6], fANOVA [11] and forward selection based on empirical performance models [10] are orthogonal to our work, since they quantify the impact of parameters on algorithm performance, but do not provide direct insights into the shape of the parameter responses or structural aspects of the configuration landscapes arising from these shapes.

The remainder of this paper is structured as follows. In Section 2, we explain our methods for investigating the structure of configuration landscapes. Then, in Section 3, we introduce the setup used for our experimental investigation. The results of this investigation are presented in Section 4, and finally, in Section 5, we summarize our findings and outline several avenues for future work.

2 Methods

Our approach to analyzing configuration landscapes is severely constrained by the fact that for typical configuration scenarios, obtaining performance measurements for all configurations or even sampling a substantial fraction of the landscape would be prohibitively expensive. For example, the smallest configuration scenario we study (which involves two integer-valued parameters for the TSP solver, EAX [17]), would take over 500 CPU years on our reference machines to obtain complete evaluation of the corresponding configuration landscape. Configuration spaces grow exponentially with the number of parameters, so even relatively sparse samples quickly become unaffordable. Therefore, to perform our analysis of configuration landscapes, we restricted ourselves to a small number of configurations. In the light of this, and consistent with the aims of our investigation, we focused on individual slices of the parameter responses.

2.1 Parameter Response Slices

Given a target algorithm A, a *response slice for parameter p* is obtained by fixing all other parameters of A to some value and measuring the performance of A as a function of p. Intuitively, this corresponds to a slice through the configuration landscape of A, and technically, it can be seen as a conditional response, subject to all other parameters taking a specific value. Since we are primarily interested in the parameter responses near high-quality configurations, we first performed

25 independent runs of SMAC [9] for each scenario (configuring both numerical and categorical parameters), and subsequently evaluated the resulting configurations on the entire training set. We then used the best-performing configuration on the training set as the centre point for each parameter slice.

Even evaluating every possible value for each parameter response slice in our EAX scenario (described in more detail in Section 3) would take 6 CPU years, so we further reduced our slices to only 15 parameter values each. If a parameter had less than 15 possible values, then we used all of them; otherwise, to obtain high resolution near the best-known parameter setting, we increased the spacing between adjacent values exponentially with increasing distance from the best-known value. We added an additional constraint to ensure that we obtained some coverage of the parameter response on either side of the best-known parameter setting: we restricted at most 75% of the points to be on one side of the best-known value (note that if the best-known value took the maximum or minimum allowed value, we could not enforce this constraint). Since there were still many possible locations for the points satisfying these constraints, we multiplied the grid of points by a randomly chosen weight to choose their exact location. For any integer-valued parameter, we first determined a set of values as for real-valued parameters, and then rounded each setting thus obtained to the nearest valid and previously unused value. Finally, to obtain robust performance estimates for each value in a given parameter slice, we performed 10 independent runs per instance and determined a median performance value from these.

2.2 Bootstrap Analysis and Confidence Intervals

To account for the variance between independent target algorithm runs and problem instances, we used a nested bootstrap procedure similar to the one by Mu *et al.* [15] with 1 001 outer and inner bootstrap samples. To be precise, for each parameter value and problem instance, we created 1 001 (inner) bootstrap samples of the 10 independent runs to obtain a distribution of median performance values; from these, we determined 95% bootstrap confidence intervals. Next, we created 1 001 bootstrap samples of the instance set, and for each occurrence of an instance in a sample we sampled at random from the corresponding distribution of median performance values. Finally, we calculated medians and 95% confidence intervals for the performance observed at each parameter value. We used Bonferroni multiple testing correction when calculating confidence intervals, since each slice had up to 15 parameter values, and linear interpolation to estimate confidence intervals between adjacent parameter values.

2.3 Tests for Convexity and Uni-Modality

We designed two testing procedures to determine if there is sufficient evidence to reject the hypotheses of convexity and uni-modality of a given parameter response slice with 95% confidence. These tests attempt to fit a piece-wise linear curve that is constrained to be uni-modal or convex, respectively, through the previously described bootstrap confidence intervals. We say that a test rejects

uni-modality or convexity if no such line exists. If the upper bound of a parameter value was censored, we excluded it from the test, since there is insufficient information to properly reason about it. When considering individual instance response slices (RQ 2), if there were too few uncensored data points to perform a test, we considered it to be insufficient data to reject the hypotheses of convexity or uni-modality. For aggregate response slices (RQ 1), we used PAR10 scores, counting censored runs at 10 times our running time cutoff).

2.4 Identifying "Interesting" Parameter Response Slices

Parameters with almost flat responses (*i.e.*, robust ones, whose settings have little or no effect on the performance of the algorithm) are of limited interest to our investigation. We therefore used a simple heuristic procedure to identify parameters with interesting (*i.e.*, non-flat or sensitive) responses, based on the sizes of and overlap between the bootstrap confidence intervals for each value in the respective parameter response slice. To be precise, we define a parameter's response to be *interesting* if the size (in terms of the log of the performance measure) of the overlap between the two confidence intervals with the least amount of overlap is at most half of the average size of the confidence intervals.

2.5 Counting the Number of Modes (Local Minima)

One commonly used feature to describe the ruggedness of a search landscape is the number and density of local optima (see, *e.g.*, Chapter 5 of [8]). We partially capture this notion of ruggedness by counting the number of modes that occur along a parameter response slice. To do this, we use a very similar procedure to our tests for uni-modality and convexity: we fit the flattest possible piece-wise linear curve within the region defined by the 95% confidence intervals along a given parameter response slice and then count the number of modes in that line.

2.6 Fitness Distance Analysis

Since traditional fitness distance analysis would have been too expensive, considering the constraints on our computational budget, we applied it locally to the sets of data points belonging to each parameter response slice. We also calculated the FDC for each bootstrap sample of a parameter response slice to obtain medians and confidence intervals for each slice.

3 Experimental Setup

We studied 10 different algorithm configuration scenarios, spanning three widely studied, $\mathcal{NP}$-hard problems (SAT, MIP and TSP), 6 prominent algorithms for these and 5 well-known instance sets. All of these scenarios involve the minimization of running time, measured in terms of PAR10, *i.e.*, mean running time with timed-out runs counted at 10 times the running time cutoff. In Table 1, we

Table 1. The instance sets we studied from ACLib scenarios and the configuration budgets and training/testing running time cutoffs we used for their scenarios.

Problem	Instance Set	Configuration Budget [CPU days]	Training Running time cutoff [CPU sec]	Test Running time cutoff [CPU sec]
SAT	circuit-fuzz	2	300	600
	BMC08	2	300	600
MIP	CLS	2	10000	10000
	Regions200	2	10000	10000
TSP	tsp-rue-1000-3000	1	86	3600

Table 2. The 6 algorithms we studied. *We configured all 185 numerical parameters, but only studied slices for the 10 most important (see text for details).

Problem	Algorithm	Version	Numerical Parameters	Categorical Parameters
SAT	CaDiCaL	sc17	40	22
	lingeling	azf	185*	137
	cryptominisat	4.1	22	36
MIP	CPLEX	12.6	22	52
TSP	EAX+restart	JDL	2	0
	LKH+restart	2.0.7	12	9

summarize the configuration budgets and running time cutoffs used for our scenarios. All instance sets are readily available online in ACLib scenarios that have been identified as interesting and challenging benchmarks for algorithm configurators [13]. For the SAT and MIP instance sets, we used the same budgets and running time cutoffs as specified in the corresponding ACLib scenarios. We increased the running time cutoff for the test set (and parameter slices) for the SAT and TSP scenarios, in order to better assess poorly performing configurations.

Table 2 provides an overview of the 6 algorithms we studied. We introduce a few new, state-of-the-art algorithms not found in existing ACLib scenarios.

For SAT, we studied CaDiCaL [3], because it was one of the top-performing, configurable solvers in the application track of the 2017 SAT competition; lingeling [3], because it was the winner of the 2014 Configurable SAT Solver challenge on the circuit-fuzz and BMC08 instances; and cryptominisat [18], because it is a variant of the well-known and commonly used minisat algorithm. Reference implementations of lingeling and cryptominisat were directly obtained from ACLib 2.0, whereas that of CaDiCaL was taken from the 2017 SAT competition.

For TSP, we chose two extensively studied [4,16], state-of-the-art, inexact solvers: EAX [17] and LKH [7]. We used the same implementations as Mu *et al.* [16] and Dubois-Lacoste *et al.* [4], which were modified to use a restarting mechanism and terminate upon reaching optimal solution quality values (known from long runs of an exact solver). The TSP scenarios in ACLib configure for solution quality, so we chose these solvers to focus on running time minimization.

For MIP, we studied the high-performance commercial solver IBM ILOG CPLEX [1], version 12.6 (featured in several ACLib scenarios), which terminates upon finding an optimal solution to a given MIP instance and completing a proof of optimality. We slightly modified the CPLEX scenarios from ACLib, by

Table 3. PAR10 values on the test sets for the default configuration versus the configuration with the best training PAR10. All times are in CPU seconds.

Problem	Algorithm	Instance Set	Default PAR10	Configured PAR10	Speedup
SAT	CaDiCaL	circuit-fuzz	468.71	252.34	1.86
		BMC08	638.57	637.93	1.00
	lingeling	circuit-fuzz	382.23	279.29	1.37
		BMC08	692.80	691.51	1.00
	cryptominisat	circuit-fuzz	444.68	276.83	1.61
		BMC08	938.61	970.07	0.97
MIP	CPLEX	CLS	40.39	3.39	11.91
		Regions200	106.77	6.40	16.68
TSP	EAX	tsp-rue-1000-3000	65.99	56.84	1.16
	LKH	tsp-rue-1000-3000	428.60	228.62	1.87

treating CPLEX as a randomized algorithm. Earlier versions of CPLEX used a fixed random seed that was not exposed to the user; however, CPLEX is in fact a randomized solver, and treating it as such avoids potential problems arising from bias due to the use of a specific random seed.

For every algorithm except lingeling we were able to evaluate parameter slices for all of their numerical parameters; however, since lingeling had so many parameters, we restricted our analysis to a subset of them. Falkner *et al.* [5] reported the 10 most important parameters according to fANOVA [11] (all of which were numerical) for lingeling on the circuit-fuzz instance set, so we only used these 10. We also slightly modified the ranges for a few parameters for LKH and CPLEX. Some of the numerical parameters use values 0 or -1 to encode special behaviour, *e.g.*, the automatic setting of the parameter value or deactivation of the mechanism controlled by the parameter. In cases where the documentation was unclear, we erred on the side of caution and removed a parameter value or treated the special value as a categorical parameter.

In Table 3, we show the results from configuring our 10 scenarios, using 25 runs of SMAC [9] per scenario. These results are consistent with the literature. We note that in some cases, configuration did not result in significant performance improvements over the default parameter settings of a given algorithm; this is unproblematic, since our goal in performing automated configuration was not to obtain improved performance, but rather to ensure we used high-performance configurations as reference points for the parameter response slices that formed the basis for our configuration landscape analysis.

We ran all of our experiments on Ada, a cluster of 20 nodes, equipped with 32 2.10GHz Intel Xeon E5-2683 v4 CPUs with 40960KB cache and 96 GB RAM each, running openSUSE Leap 42.1 (x86_64). To minimize detrimental cache effects and memory contention, in all experiments, we used a single core per CPU and limited RAM use to 3 GB. In total, we used 43.5 CPU years for automated configuration and collection of parameter response slice data.

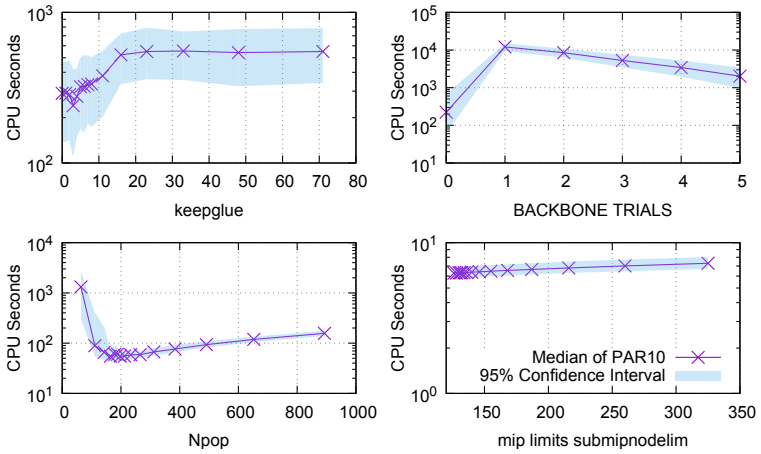

Figure 1. Four parameter response slices. From left to right top to bottom: CaDiCaL's `keepglue` on the circuit-fuzz instance set, LKH's `BACKBONE_TRIALS` on the tsp-rue-1000-3000 instance set, EAX's `Npop` on the tsp-rue-1000-3000 instance set and CPLEX's `mip_limits_submipnodelim` on the Regions200 instance set.

4 Results

We collected and analyzed 193 parameter slices for instance sets and individual problem instances, as motivated in Section 1 and outlined in Sections 2 and 3.

4.1 RQ 1. Uni-Modality and Convexity on Instance Sets

Overall, the parameter response slices for instance sets appear to be more benign than one might expect. Our tests for uni-modality and convexity failed to reject for all but 1 of the 193 parameter slices. That is, 99.48% of the slices we measured appear to be both uni-modal and convex. Somewhat surprisingly, our heuristic method outlined in Section 2.4 identified only 18 of the slices as interesting. In Figure 1, we show 4 parameter response slices that are representative of the qualities we observed in this set of 18 (the remaining 14 are available in our online, supplementary material, available at `http://ada.liacs.nl/projects/ac-landscapes`). To our surprise, neither lingeling nor cryptominisat had any interesting parameter response slices. The parameter that fANOVA rated to be the most important for lingeling [5] shows a slight dip at the smallest parameter value in the slice for the circuit-fuzz instance set. To investigate further, we evaluated an additional 15 parameter values, but still found it to be un-interesting according to our criterion.

The only parameter we found having a non-unimodal and non-convex response was LKH's `BACKBONE_TRIALS` parameter (see Figure 1), which specifies the number of backbone trials in each run. Apart from `BACKBONE_TRIALS`=0, even this response slice appears to be convex and uni-modal. To the best of our

knowledge, a value of 0 does not have special meaning, apart from the obvious semantic difference of *some* versus *no* backbone trials, which may alone account for this difference, since it likely corresponds to a (poorly performing) heuristic component of the algorithm that is turned on or off.

Some of the parameter responses (*e.g.*, `keepglue` in Figure 1), appear to be flat for poorly performing parameter values, and hence non-convex overall. Our tests were unable to reject convexity, despite this visual evidence, because of the relatively wide bootstrap confidence intervals. However, we believe that these flat regions are an artifact of how PAR10 scores treat censored runs, and that a sufficiently large running time cutoff would yield convex responses.

Interestingly, in the three SAT scenarios involving the BMC08 instance set, we found only one parameter response slice considered interesting according to our criterion: CaDiCaL's `restartmargin`. This is consistent with the fact that SMAC was unable to achieve significant performance improvements for these scenarios. We further note that the default value for `restartmargin` is very near to the best-known value. Hence, it appears that better configurations may not exist rather than being hard to find due to highly irregular or rugged landscapes.

4.2 RQ 2. Uni-Modality and Convexity on Individual Instances

To study our second research question, we ran our tests for convexity and uni-modality on the parameter response slices for each individual problem instance. We consider the parameters independently by looking at statistics of their responses on each instance. For example, on the left pane of Figure 2 we plot a cumulative distribution function (CDF) showing on the y-axis the percentage of parameters that had convex responses on a percentage of instances less than or equal to the value specified on the x-axis. Surprisingly, there is a large percentage of parameters with convex responses slices for most instances. However, nearly half of the parameters with interesting response slices on the entire instance set tend to have much fewer convex parameter responses on the individual instances. Our procedure (outlined in Section 2.3), sometimes assumes uni-modality or convexity when there is insufficient data to perform a test. On average, over all parameters, this happened for only 6.2% of the instances we considered, and at most, on 16.9% of the instances. Hence, even if all of these cases were instead assumed to be non-unimodal or non-convex, our overall results would not be substantially different.

Furthermore, looking at the CDF of the average numbers of modes for each instance parameter slice on the right pane of Figure 2, we see that just under 50% of the interesting parameters have an average of more than one mode for their individual instance responses. On the other hand, most of the parameters have an average of only one mode per instance, which is consistent with the fraction of parameters with primarily convex instance response slices.

Overall, there are a surprisingly large number of parameter response slices that are both uni-modal and convex on most or all of their individual instances. In Table 4, we show a summary of these results, in addition to the corresponding results for the parameter responses on entire instance sets. Note that for the

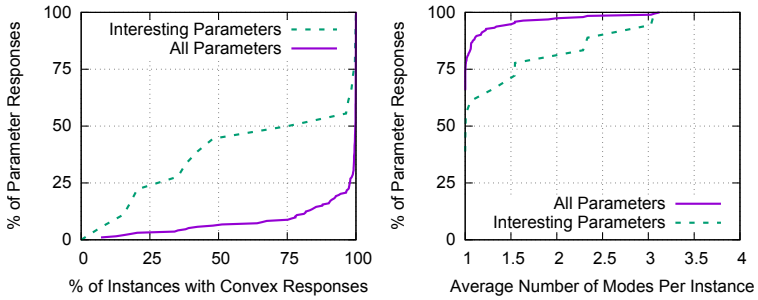

Figure 2. CDFs summarizing our findings for individual instances. Left: for each parameter we computed the percentage of instances on which it had a convex response, and then we plot the CDF of these percentages; right: the CDF of the average number of modes observed in the responses for a parameter on each instance.

Table 4. Left: the percentages of uni-modal and convex parameter response slices on entire instance sets; right: we computed the percentage of instances with convex or uni-modal responses for each parameter, and then show the average percentages over the parameters, i.e., we show the percentage of convex and uni-modal instance responses for the "average parameter".

	Instance Set		**Individual Instances**	
	% Uni-Modal Parameters	**% Convex Parameters**	**Average % Uni-Modal Instances**	**Average % Convex Instances**
All Parameters	99.5	99.5	95.3	92.6
Interesting Parameters	94.4	94.4	76.1	66.1

aggregate instance set parameter responses, we show the percentage of uni-modal and convex parameter response slices observed on different instance sets, whereas for the individual instances, we first computed the percentage of instances with uni-modal and convex responses for each parameter, and then report the average percentages over the set of all parameters.

Our analysis of the fitness distance correlation coefficient (FDC) for the parameter response slices supports our hypothesis that parameter responses on individual instances are more rugged than the aggregate responses on entire instance sets. In particular, we found that 80% of the parameters have an average instance response slice FDC less than 0.25, compared to 0.4 for the instance set responses. However, through manual inspection of the instance parameter slices, we found that some responses obtained low FDC scores simply because they are relatively flat (hence deviations in parameter value have low correlation with deviations in algorithm performance). Still, the high average numbers of modes observed for some of the parameters indicate that these responses are truly rugged.

To check that these were not spurious results, we performed exact replicates for three scenarios that were near the Pareto front of the largest average number of modes and the smallest average FDC: CaDiCaL's `posize` and `elimint` on circuit-fuzz instances, and CPLEX's `mip_limits_cutpasses` on CLS instances.

Then, for each parameter, we chose three instances near to their respective Pareto fronts. In all cases, the replicates were qualitatively identical to the original ones. Additional details on these experiments and our FDC analysis can be found at `http://ada.liacs.nl/project/ac-landscapes`.

5 Conclusions and Future Work

Overall, we found strong support for our hypothesis that parameter responses on instance sets tend to be uni-modal and convex. We also found evidence that many parameters have convex and uni-modal responses on individual problem instances; however, these responses tend to be (in some cases substantially) more rugged than their aggregate counterparts. We were surprised to find that a small percentage of parameters appear to have highly rugged responses on most of the instances. However, even though these parameters have rugged response slices on individual instances, their aggregate responses still tend to be uni-modal and convex on the entire instance set, after performing bootstrap sampling to account for the variability over instances. This may be why the simple Gaussian model used for generating promising configurations in irace [14], which inherently exploits smoothness in individual parameter responses, works rather well.

Our results do not preclude the possibility of complex parameter interactions that result in configuration landscapes with many local optima. Future work could study parameter interactions and investigate whether or not local minima are rare, or at least easy to escape. Categorical parameters also play an important role in many algorithm configuration scenarios. Here, we set categorical parameters to values found in high-quality solutions; however, it would be interesting to explore whether similar results to those we reported here hold for other settings of the categorical parameters of a given target algorithm. Moreover, it would be very interesting to investigate to which extent our findings interact with parameter importance, as assessed by fANOVA and ablation analysis.

We note that our method for collecting data is focussed on high-performance regions of the configuration spaces we considered, as is the search process of algorithm configurators. Therefore, our results may only hold in these regions; this would be at least of theoretical interest and could be investigated in future work. Another direction is to extend our analysis to scenarios that involve optimization of solution quality, such as loss scenarios involving hyperparameter optimization of machine learning algorithms. Finally, and perhaps most interestingly, we strongly believe that our results open the door to designing new algorithm configuration procedures that exploit the relatively benign characteristics of typical configuration landscapes discovered in this work.

Acknowledgements

YP was supported by an NSERC Vanier Scholarship. HH acknowledges funding through an NSERC Discovery Grant, CFI JLEF funding and startup funding from Universiteit Leiden.

References

1. IBM Corp. IBM ILOG CPLEX Optimizer. `https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer`, 2018. Last accessed on March 30th, 2018.
2. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Proceedings of CP 2009. pp. 142–157 (2009)
3. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT Competition 2017. In: Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions. pp. 14–15 (2017)
4. Dubois-Lacoste, J., Hoos, H., Stützle, T.: On the empirical scaling behaviour of state-of-the-art local search algorithms for the Euclidean TSP. In: Proceedings of GECCO. pp. 377–384 (2015)
5. Falkner, S., Lindauer, M., Hutter, F.: SpySMAC: Automated configuration and performance analysis of SAT solvers. In: SAT. pp. 215–222 (2015)
6. Fawcett, C., Hoos, H.: Analysing differences between algorithm configurations through ablation. JOH **22**(4), 431–458 (2016)
7. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. EJOR **126**, 106–130 (2000)
8. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
9. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of LION. LNCS, vol. 6683, pp. 507–523 (2011)
10. Hutter, F., Hoos, H., Leyton-Brown, K.: Identifying key algorithm parameters and instance features using forward selection. In: Proceedings of LION. LNCS, vol. 7997, pp. 364–381 (2013)
11. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Proceedings of ICML. pp. 754–762 (2014)
12. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR **36**, 267–306 (2009)
13. Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H., Leyton-Brown, K., Stützle, T.: AClib: A benchmark library for algorithm configuration. In: Proceedings of LION. pp. 36–40 (2014)
14. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. ORP **3**, 43–58 (2016)
15. Mu, Z., Hoos, H.: Empirical scaling analyser: An automated system for empirical analysis of performance scaling. In: Proceedings of GECCO. pp. 771–772 (2015)
16. Mu, Z., Hoos, H., Stützle, T.: The impact of automated algorithm configuration on the scaling behaviour of state-of-the-art inexact TSP solvers. In: Proceedings of LION. LNCS, vol. 10079, pp. 157–172 (2016)
17. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS JOC **25**(2), 346–363 (2013)
18. Soos, M.: CryptoMiniSat v4. In: Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions. p. 23 (2014)