CAVE: Configuration Assessment, Visualization and Evaluation

André Biedenkapp(✉), Joshua Marben, Marius Lindauer, and Frank Hutter

University of Freiburg, Freiburg, Germany
{biedenka,marbenj,lindauer,fh}@cs.uni-freiburg.de

Abstract. To achieve peak performance of an algorithm (in particular for problems in AI), algorithm configuration is often necessary to determine a well-performing parameter configuration. So far, most studies in algorithm configuration focused on proposing better algorithm configuration procedures or on improving a particular algorithm's performance. In contrast, we use all the collected empirical performance data gathered during algorithm configuration runs to generate extensive insights into an algorithm, given problem instances and the used configurator. To this end, we provide a tool, called *CAVE*, that automatically generates comprehensive reports and insightful figures from all available empirical data. *CAVE* aims to help algorithm and configurator developers to better understand their experimental setup in an automated fashion. We showcase its use by thoroughly analyzing the well studied SAT solver *spear* on a benchmark of software verification instances and by empirically verifying two long-standing assumptions in algorithm configuration and parameter importance: (i) Parameter importance changes depending on the instance set at hand and (ii) Local and global parameter importance analysis do not necessarily agree with each other.

1 Introduction

In the AI community, it is well known that the algorithm parameters have to be tuned to achieve peak performance. Since manual parameter tuning is a tedious and error-prone task, several methods were proposed in recent years to automatically optimize parameter configurations of arbitrary algorithms [1–5]. This led to performance improvements in many AI domains, such as propositional satisfiability solving [6], AI planning [7], the traveling salesperson problem [8], set covering [9], mixed-integer programming [10], hyper-parameter optimization of machine learning algorithms [11] and architecture search for deep neural networks [12]. These studies focus either on proposing more efficient automated parameter optimization methods or on improving the performance of a particular algorithm (the so-called *target algorithm*).

To determine a well-performing parameter configuration of a given algorithm on a given instance set, algorithm configuration procedures (in short *configurators*) have to collect a lot of empirical performance data. This entails running the algorithm at hand with different parameter configurations on several problem

R. Battiti et al. (Eds.): LION 12 2018, LNCS 11353, pp. 115–130, 2019.
https://doi.org/10.1007/978-3-030-05348-2_10

instances to measure its performance. Only the best performing configuration is returned in the end and all the collected performance data is typically not used further, although it was very expensive to collect.

In this paper, we reuse this data to further analyze all parts involved in the configuration process. This includes the target algorithm, its parameters, the used instances as well as the configurator. Hence, users will not only obtain a well-performing parameter configuration by using our methods in combination with a configurator but several additional insights.

A potential use-case is that algorithm developers implemented a new algorithm and want to empirically study that algorithm thoroughly as part of a publication. In a first step, they run a configurator to optimize their algorithm's parameters on a given instance set to achieve peak performance. As the next step, our automatic analysis tool *CAVE* (Configuration Assessment, Visualization and Evaluation) generates figures that can be directly used in a publication.

In *CAVE*, we build on existing methods to analyze algorithms and instances, reaching from traditional visualization approaches (such as scatter plots) over exploratory data analysis (used, e.g., for algorithm selection [13,14]) to recent parameter importance analysis methods [15–18]. We combine all these methods into a comprehensive analysis tool tailored to algorithm development and configuration studies, and propose two new approaches to complement their insights. Specifically, the contributions of our paper are:

1. We give an overview of different approaches to analyze a target algorithm, a given instance set and the configurator's behavior based on collected empirical data. We use these for an exemplary analysis of the SAT solver *spear* [19] on a benchmark of SAT-encoded software verification instances.
2. We propose a new qualitative analysis of configurator footprints based on a recently proposed similarity metric of configurations [20].
3. We propose a new parameter importance analysis by studying the impact on performance when changing one parameter at a time, thus exploring the immediate neighborhood of the best found configuration.
4. We provide a ready-to-use toolkit, called *CAVE*[1], for such analyses, which can be directly used in combination with the configurator *SMAC* [3].
5. We show the value of our tool and the need of such comprehensive analyses by verifying two common assumptions for algorithm configuration:
   (a) Parameter importance changes depending on the instance set at hand.
   (b) Local and global parameter importance analysis do not necessarily agree with each other and hence complement each other.

2 Related Work

Empirical evaluation of algorithms is as old as computer science. One of the first systematic approaches for ensuring reproducibility and insights in comparing a set of algorithms is the PAVER service [21,22]. The tool is primarily tailored to

[1] https://github.com/automl/CAVE

mixed integer programming solvers and provides some tables and visualization of the performance of algorithms. In contrast to PAVER, our tool also considers parameters of an algorithm and is designed to be used on arbitrary algorithms.

In the context of algorithm selection [23], a lot of performance data of different algorithms is collected, which can be used to analyze algorithms and instance sets. An example of an exploratory data analysis for algorithm selection is part of the algorithm selection library ASlib [13] that provides some simple performance and distribution tables and corresponding plots, e.g., scatter and box plots.

The first system that included an automatic analysis of algorithm performance in the context of algorithm configuration was the high-performance algorithm laboratory *HAL* [24]. Its main purpose was to help algorithm developers to apply automated algorithm design methods (such as algorithm configuration and selection) in a uniform framework by handling all interfaces, data storage, running experiments and some basic aggregation and visualizations of results, such as scatter and performance distribution plots. In contrast to *HAL*, *CAVE* focuses on the analysis part and provides a far more extensive analysis.

The tool *SpySMAC* [25] followed a similar approach as *HAL* but was specifically tailored to the needs of the propositional satisfiability (SAT) community. It provided an easy-to-use interface to run the configurator *SMAC* [3] for optimizing parameter configurations of a SAT solver. Our approach is inspired by *SpySMAC* but with the focus on the analysis part and extends it substantially. Furthermore, our new approach is no longer specific to SAT and *SMAC*, but it can be applied to any algorithm and configurator.

In the context of black-box optimization and in particular for hyperparameter optimization of machine learning algorithms, Golovin et al. [26] proposed *Google Vizier*. Similar to *HAL* and *SpySMAC*, it is a service to run optimization benchmarks and also provides some visualizations, such as performance-over-time of the target function or parallel coordinate visualizations [27]. Since *Google Vizier* focuses on black-box optimization, it does not have a concept of instances, which are an integral part of algorithm configuration.

Lloyd et al. [28] proposed automatically constructed natural language reports of regression models, giving raise to the automatic statistician tool. Although we have the same goal (providing automatically constructed reports to help users to get more out of their data), our goal is not to provide a natural language report, but to leave the interpretation of the results to the users.

3 Generation of Algorithm Configuration Data

In this section, we describe the general work-flow of generating algorithm configuration data (see Fig. 1), which will be the input for *CAVE*'s analyses in the next section. The typical inputs of configurators are[2]:

[2] We ignore in this simplified view that several budgets have to be defined, such as, the configuration budget (e.g., time budget or maximal number of algorithm calls) and resource limits of the target algorithm runs (e.g., runtime and memory limits).

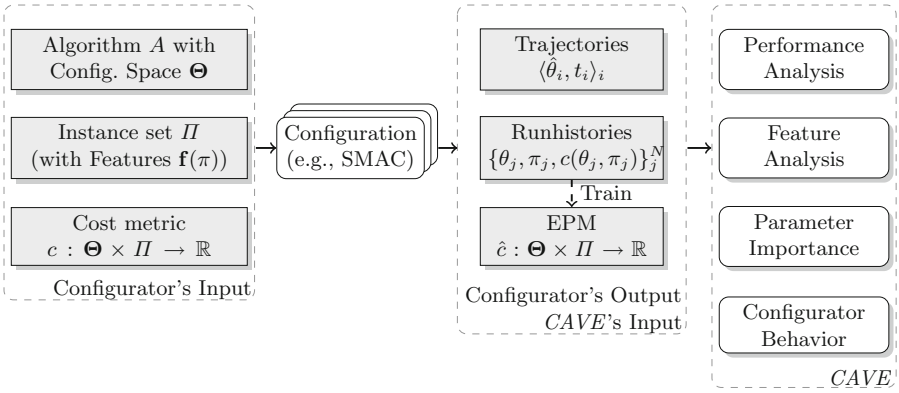


Fig. 1. Work-flow of algorithm configuration (AC) and analysis.

- *a target algorithm* A with a description of the parameter configuration space $\mathbf{\Theta}$, i.e., all possible parameter configurations,
- *an instance set* Π drawn from a distribution $\mathcal{D}_\Pi$ over all possible instances[3],
- *a cost metric* $c : \mathbf{\Theta} \times \Pi \to \mathbb{R}$ to be optimized (e.g., the runtime of an algorithm, the quality of a plan for an AI planning problem instance or the error of a machine learning algorithm).

A configurator's task is to find $\theta^* \in \mathbf{\Theta}$ such that the cost c of running A using θ^* across Π is optimized, i.e., $\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \sum_{\pi \in \Pi} c(\theta, \pi)$. Since a configurator iteratively improves the currently best configuration $\hat{\theta}$ (called *incumbent*), its *trajectory* includes the incumbent $\hat{\theta}_i$ at each time point t_i.

In this process, the configurator follows a strategy to explore the joint space of configurations and instances, e.g., local search [1], genetic algorithms [2] or model-based [3] and it collects empirical cost data. Internally, a configurator keeps track of a set $\{\theta_j, \pi_j, c_j\}_{j=1}^N$ of all evaluated configurations θ_j on instance π_j with the corresponding cost c_j, called the *runhistory*. The runhistory is an optional output, but it is crucial for *CAVE*'s analyses. If several runs of a configurator were performed, *CAVE* can use all resulting outputs as input.

To guide the search of configurators, a recent approach is to use *empirical performance models (EPMs)* $\hat{c} : \mathbf{\Theta} \times \Pi \to \mathbb{R}$ that use the observed empirical data from the runhistory to predict the cost of new configuration-instance pairs [29], e.g., using random forests [30]. Based on these predictions, a configurator can decide whether to explore new regions in the configuration space or to exploit knowledge of the presumably well-performing regions in the configuration space. Model-based configurators [3,4] can return these EPMs directly, and for model-free configurators [1,5], such a model can be subsequently learned based on the

[3] Typically, the instance set is split into a training and a test set. On the training set, the target algorithm is optimized and on the test set, an unbiased cost estimate of the optimized parameter configuration is obtained.

returned runhistories. To this end, an optional input to configurators are instance features providing numerical descriptions of the instances at hand [29,31]. Hutter et al. [29] showed that predictions of new cost data is fairly accurate if enough training data is provided. Further, Eggensperger et al. [32] showed that EPMs trained on runhistory data from configurator runs are good surrogates for real target algorithm runs. Thus, we also use EPMs trained on the union of all runhistories for our analyses, e.g., to impute missing cost data of configurations that were evaluated only on some but not all instances.

Our tool *CAVE* analyzes all this data as described in the next section. Thereby we use an extended version of the output format defined for the second edition of the algorithm configuration library AClib [33] such that *CAVE* can be in principle used with any configurator. Right now, we have a ready-to-use implementation in combination with the configurator *SMAC* [3].

4 Analyzing Algorithm Configuration Data

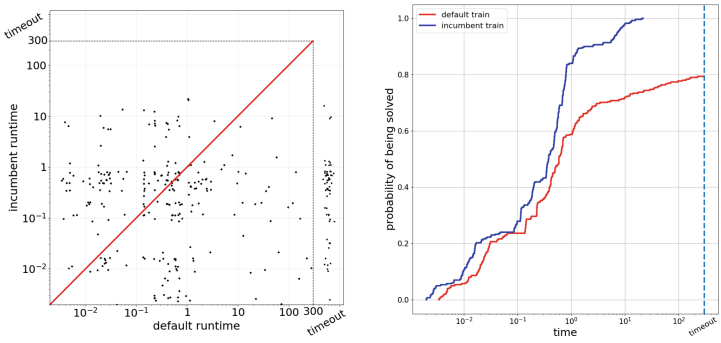
In this section, we give a brief overview of all components of our analysis report generated based on the trajectory, runhistory data and EPMs described in the last section. A detailed description of the individual elements of *CAVE* can be found in the online appendix[4]. As a running example, we show figures for studying the SAT solver *spear* [19] on SAT instances encoding software verification problems based on three 2-day SMAC runs.[5] In addition to the data generated by SMAC, we validated all incumbent configurations to decrease the uncertainty of our EPM in the important regions of the configuration space.

4.1 Performance Analysis

The performance analysis of *CAVE* mainly supports different ways of analyzing the final incumbent and the performance of the algorithm's default parameter configuration (the results obtained by downloading the algorithm and running it with its default parameter settings). In particular, the performance analysis part of *CAVE* consists of a qualitative tabular analysis providing aggregated performance values across all instances, scatter plots showing default performance vs. optimized performance for each instance (Fig. 2a), empirical cumulative performance distribution (eCDF) plots across the instance set (Fig. 2b) and algorithm footprint plots [14] (Fig. 3a).

What have we learned about spear? Figure 2 shows the scatter plot and the eCDF for *spear* on software verification instances. From these plots, we learn that the performance of *spear* was not improved on all instances, but on many of them, with large speedups on some. The optimized configuration solved all instances in at most 20 seconds, while the default led to many timeouts. Based on the

[4] http://ml.informatik.uni-freiburg.de/papers/18-LION12-CAVE.pdf
[5] The complete generated report can be found at http://ml.informatik.uni-freiburg.de/~biedenka/cave.html



(a) Scatter Plots

(b) Empirical Cumulative Distribution Function (eCDF) Plots

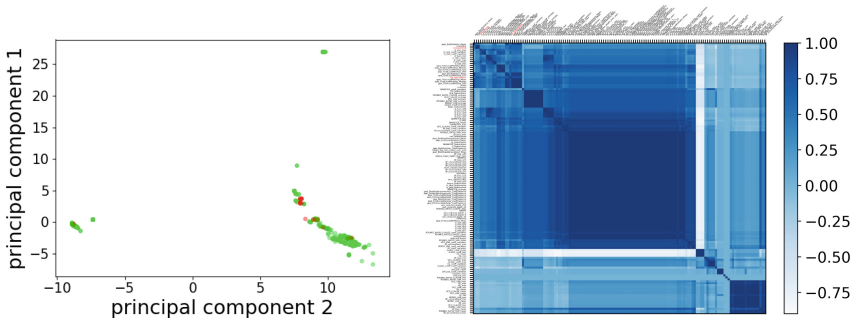
Fig. 2. Comparison of the empirical performance of the default and final incumbent.

eCDF plot, the optimized configuration solved all instances whereas the default configuration solved only 80% of the instances. Furthermore, if the cutoff time of *spear* is larger than 0.8 seconds, the optimized configuration performed better; the blue curve is above the red curve consistently. The algorithm footprint plot (Fig. 3a) shows that the incumbent performed well on different types of instances. Compared to the scatter plot, we expected to see more red points in the footprint plots (instances where the default outperforms the incumbent). Looking more deeply into the data revealed that several of these are overlapping each other because the instances features are missing for these instances.

4.2 Instance and Instance Feature Analysis

If instances are characterized by instance features (as done for model-based configurators), these features can be studied to better understand an instance set. To obtain instance features for the SAT instances at hand, we used the instance feature generator accompanied by the algorithm selection tool *SATzilla* [29,34]. In particular, the feature analysis part of *CAVE* consists of box and violin plots for each instance feature, clustering plots based on a PCA into the 2-dimensional feature space, correlation heatmaps for each pair of features (Fig. 3b), and feature importance plots based on greedy forward selection [15].

What have we learned about the software verification instances? Based on these plots, we learned that there are at least three instance clusters mixed together; knowing the source of these instances (also reflected in the instance names), we can verify that software verification for four different software tools were encoded in this instance set and that clustering approximately recovered the sources (merging two sources into one cluster). Since the PCA plot and the footprint plot (Fig. 3a) indicate that the instance set is heterogeneous [35], using algorithm configuration on each individual software tool or per-instance algorithm



(a) Algorithm Footprint (b) Feature Correlation

Fig. 3. (a) Green dots indicate instances where the incumbent configuration performed well i.e., at most 5% worse than the oracle performance of default and incumbent. All other instances are plotted as red dots, indicating that the default configuration performed well. Instances might be mapped closely together in the 2D reduced space. (b) The correlation matrix for all features is shown.

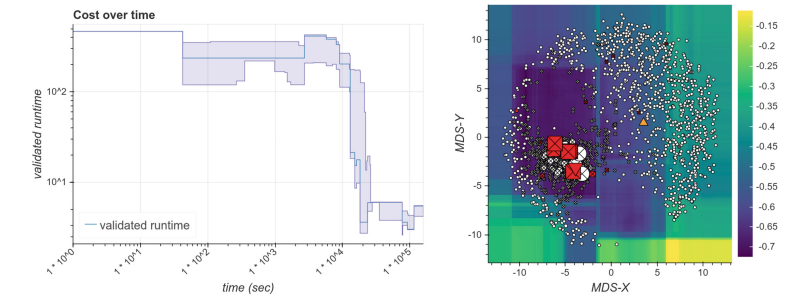
configuration [36,37] could improve the performance of *spear* even further. From the feature correlation plot, we see that roughly half of the features are highly correlated and some of the features could be dropped potentially.

4.3 Configurator Behavior

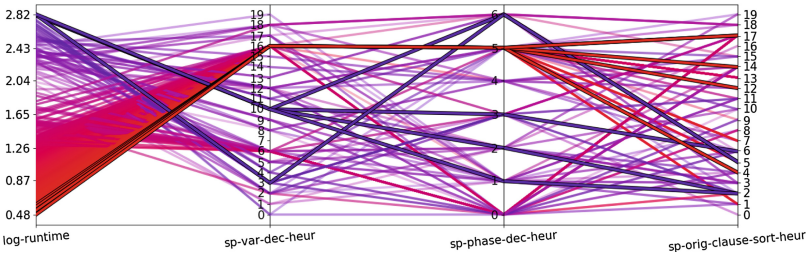
Besides insights into the algorithm and instances at hand, the trajectory and the runhistory returned by a configurator also allow for insights into how the configurator tried to find a well-performing configuration. This may lead to insights into how the optimization process could be adjusted. In particular, the configurator behavior analysis consists of a plot showing the performance of the target algorithm over the time spend by the configurator (Fig. 4a) and parallel coordinate plots showing the interactions between parameter settings and performance of the target algorithm [26] (Fig. 4c).

Configurator Footprint. As a novel approach, we propose in this paper to study how a configurator iteratively sampled parameter configurations, i.e., all configurations in the runhistory, see Algorithm 1 and exemplary Fig. 4b. It is based on a similarity metric for parameter configurations[6] which is used in a multi-dimensional scaling (MDS) [38] based on the *SMACOF* algorithm [39] to obtain a non-linear mapping into 2-dimensions [20]. We extend this analysis by highlighting incumbent configurations in the trajectory and by scaling the dots (parameter configurations) wrt. the number of instances they were evaluated

[6] In contrast to Xu et al. [20], we normalize the relabelling cost of continuous parameters to $[0, 1]$ since otherwise relabelling of continuous parameters would dominate the similarity metric compared to relabelling of discrete parameters.



(a) Cost-over-Time (b) Configurator Footprint



(c) Parallel Coordinate Plot

Fig. 4. (a) depicts the predicted performance over time. The blue area gives the first and third quantiles over three configurator runs. (b) Dots represent sampled configurations, where the size represents the amount of times a configuration was evaluated. The background shows the predicted performance in the 2D space using an MDS. Incumbents are plotted as red squares, the default as orange triangle and the final incumbent as red inverted triangle. Configurations sampled from an acquisition function are marked with an X, all other configurations were purely randomly sampled. (c) Parallel Coordinate Plot for the three most important parameters with a subsampled set of 500 configurations. The best performing configuration has the brightest shade of red, whereas the darkest shade of blue depicts the worst observed configuration.

on. For racing-based configurators, this corresponds to how well a configuration performed compared to the current best found configuration. Furthermore, we use an EPM in the 2D space to highlight promising parts of the configuration space. Finally, the figures in the html-report also have a mouse-over-effect that allows to see the parameter configuration corresponding to each dot in the figure.

What have we learned about the configurator and spear? From the cost-over-time plot (Fig. 4a), we learned that *SMAC* already converged after 40,000 seconds and investing further time did not improve *spear*'s performance further. From the configurator footprints (Fig. 4b), we see that *SMAC* covered most parts of the space (because every second parameter configuration evaluated by *SMAC* is a random configuration) but it also focused on promising areas in the space. The fraction of good configurations is fairly large, which also explains why *SMAC* was

Algorithm 1: Configurator Footprint (Visualization of a runhistory)

Input: Runhistory $\mathcal{H} = \{\theta_j, \pi_j, c_j\}_j^N$; trajectory $\mathcal{T} = \langle \hat{\theta}_k, t_k \rangle_k$; Instance set Π
For each pair $\langle \theta_i, \theta_j \rangle$, compute similarity $s(\theta_i, \theta_j)$ [20];
Fit 2D MDS based on similarities $s(\theta_i, \theta_j)$;
Replace each θ in $\mathcal{H}$ by 2D projection $MDS(\theta)$;
Plot each θ in 2D space $MDS(\theta)$ with size proportional to #entries in $\mathcal{H}$;
Highlight incumbents $\hat{\theta}$ of trajectory $\mathcal{T}$;
Fit EPM $\hat{c} : \mathbb{R}^2 \times \Pi \rightarrow \mathbb{R}$ based on $\mathcal{H}$;
Plot heatmap in background based on $\frac{1}{|\Pi|} \sum_{\pi \in \Pi} \hat{c}(MDS(\theta), \pi)$

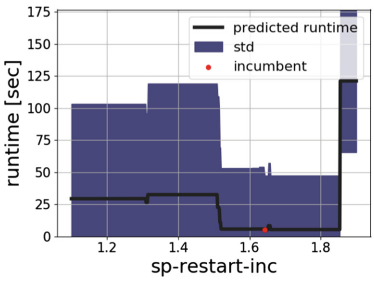
able to find a well-performing configuration early on. The parallel coordinate plot (Fig. 4c) reveals that sp-var-dec-heur should be set to 16 for peak performance; however, the runtime also depends on other parameters such as sp-phase-dec-heur.

4.4 Parameter Importance

Besides obtaining a well-performing parameter configuration, the most frequently asked question by developers is which parameters are important to achieve better performance. This question is addressed in the field of parameter importance. Existing approaches for parameter importance analysis (used in AC) include greedy forward selection [15], ablation analysis [17,18] and functional ANOVA [16].

Parameter	fANOVA	Ablation	LPI
sp-var-dec-heur	62.052	93.224	93.351
sp-phase-dec-heur	7.535	< 0.01	0.508
sp-orig-clause-sort-heur	0.764	< 0.01	< 0.01
sp-learned-clause-sort-heur	0.239	< 0.01	< 0.01
sp-restart-inc	< 0.01	4.239	3.473
sp-clause-activity-inc	< 0.01	2.406	< 0.01
sp-clause-decay	< 0.01	1.097	< 0.01
sp-variable-decay	< 0.01	< 0.01	1.005

(a) Parameter Importance



(b) LPI on sp-restart-inc

Fig. 5. (a) Relative parameter importance values for the most important parameters of each method. The parameters are ordered by fANOVA's importance values. (b) Exemplary LPI plot. The shaded area is the model uncertainty.

Local Parameter Importance (LPI). In addition to the existing approaches, we propose a new parameter importance analysis approach, which we dub LPI. It is inspired by the human strategy to look for further improved parameter

configurations or to understand the importance of parameter changes in the neighborhood of a parameter configuration. For example, most users are interested in understanding which parameters in optimized parameter configurations are crucial for the achieved performance.

Using an EPM, we study performance changes of a configuration along each parameter. To quantify the importance of a parameter value θ_p, we compute the variance of all cost values by changing θ_p and then compute the fraction of all variances. Given the parameter space of the target algorithm $\mathbf{\Theta}$, a set of parameters P and an EPM $\hat{c}\colon \mathbf{\Theta} \to \mathbb{R}$ marginalized over Π, the local importance of parameter $p \in P$ with domain $\mathbf{\Theta}_p$ is given by the fraction of variance caused by p over the sum of all variances

$$LPI(p \mid \theta) = \frac{\mathrm{Var}_{v \in \mathbf{\Theta}_p} \hat{c}(\theta\,[\theta_p = v])}{\sum_{p' \in P} \mathrm{Var}_{w \in \mathbf{\Theta}_{p'}} \hat{c}(\theta\,[\theta_{p'} = w])}$$

Compared to an ablation analysis, our new analysis is even more local since it solely focuses on one parameter configuration. Nevertheless, it is also similar to fANOVA since we quantify the importance of a parameter by studying the variance of performance changes. However the marginalization across all other parameter settings is not a part of LPI.

What have we learned about spear? Figure 5a gives an example of LPI estimated parameter importance and contrasts it to fANOVA and ablation results. The ablation analysis reveals that the most important parameter change was setting *sp-var-dec-heur* to 16 instead of the default of 0. Only a few of the 25 other parameters were important for the performance improvement. Overall, fANOVA and LPI agree with this assessment. However, LPI and ablation give a much larger importance to *sp-var-dec-heur* than fANOVA does; this is in part due to the fact that fANOVA (in contrast to LPI and Ablation) considers higher-order interaction effects as important, but also in part due to fANOVA being a global method and LPI and Ablation being local methods.

5 Exemplary Study of Parameter Importance using *CAVE*

In this section we will show that *CAVE* enables comprehensive studies. We will study two assumptions in the field of algorithm configuration regarding parameter importance:

Q1 Does the set of important parameters change depending on the instance set? If that is false, parameter importance of a given algorithm has only to be studied once and warmstarting methods of configurators [40] should perform quite well in general. Alternatively, parameter importance studies would be required on each new instance set again and warmstarting methods should perform quite poorly.

Q2 Do local and global parameter importance approaches agree on the set of important parameters? The common assumption is that local and global parameter importance analysis are complementary. A globally important parameter may not be important in a local analysis since it might already be set to a well-suited parameter value in a specific configuration.

Setup: To study these two questions, we run (capped) fANOVA [16][7], ablation analysis [18] and our newly proposed LPI analysis, through *CAVE* for different algorithms from different domains on several instance sets, see Table 1. All these benchmarks are part of the algorithm configuration library [33][8]. We note that *clasp* and *probSAT* were the respective winners on these instance sets in the configurable SAT solver challenge 2014 [6]. To collect the cost data for training the required EPMs, we ran *SMAC* (v3 0.7.1)[9] ten times on each of the benchmarks using a compute cluster with nodes equipped with two Intel Xeon E5-2630v4 and 128GB memory running CentOS 7.

Table 1. Algorithm Configuration Library benchmarks, with #P the number of parameters of each algorithm.

Algorithm	Domain	#P	Instance sets
lpg[41]	AI plan.	65	satellite, zenotravel, depots
clasp(-ASP)[42]	ASP	98	ricochet, riposte, weighted-sequence
CPLEX	MIP	74	cls, corlat, rcw2, regions200
SATenstein[43]	SAT	49	cmbc, factoring, hgen2-small, k3-r4 26-v600, qcp, swgcp
clasp(-HAND)	SAT	75	GI, LABS, queens
clasp(-RAND)	SAT	75	3cnf-v350, K3, unsat-unif-k5
probSAT[44]	SAT	9	3SAT1k, 5SAT500, 7SAT90

Metric: For both questions, we are interested in the sets of important parameters. To this end, we define a parameter to be important if it is responsible for at least 5% cost improvement (ablation) or explains at least 5% of the variance in the cost space (fANOVA, LPI). To compare the sets of important parameters for two instance sets (Q1) or for ablation, fANOVA and LPI (Q2), we report the fraction of the intersection and the union of the two sets. For example, if both sets are disjoint, this metric would return 0%; and if they are identical, the score would be 100%. In Tables 2 & 3, we show the averaged results for each solver across all instance sets; for Q1 we averaged over all pairs of instance sets and for Q2 we averaged over all instance sets.

[7] In capped fANOVA, all cost values to train a marginalized EPM are capped at the cost of the default configuration θ_{def}: $c(\theta) := \min(c(\theta_{\text{def}}), c(\theta))$.

[8] http://aclib.net/

[9] https://github.com/automl/SMAC3

Table 2. Q1: Comparison of fANOVA/ablation/LPI results across different instance sets. The values show the percentage of how often a method determined the same parameters to be important on a pair of instance sets.

	ablation		fANOVA		fANOVA_c		LPI	
Algorithm	μ	σ	μ	σ	μ	σ	μ	σ
clasp(-ASP)	8.33	5.89	41.67	18.00	21.92	27.95	31.30	4.54
clasp(-HAND)	0.00	0.00	50.00	13.61	13.61	11.34	24.94	10.24
clasp(-RAND)	13.61	4.83	11.11	15.71	2.02	1.89	27.51	4.86
CPLEX	4.03	6.29	15.83	20.56	0.00	0.00	36.37	8.44
lpg	16.19	11.97	30.00	14.14	33.33	47.14	37.63	12.44
probSAT	46.67	37.71	31.67	13.12	30.95	14.68	60.65	19.12
SATenstein	14.90	28.18	26.33	13.06	15.45	27.98	26.99	16.85

Table 3. Q2: Comparison of results obtained with different importance methods, on the same instance sets. The values show the percentage of how often two methods agreed on the set of most important parameters.

	fANOVA				fANOVA_c				ablation	
	vs. ablation		vs. LPI		vs. ablation		vs. LPI		vs. LPI	
Algorithm	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
clasp(-ASP)	8.33	11.79	5.87	1.54	3.48	2.47	25.71	19.87	12.36	4.02
clasp(-HAND)	6.67	9.43	9.98	4.06	4.88	6.90	20.23	7.53	21.97	19.15
clasp(-RAND)	38.10	44.16	13.35	3.16	35.86	45.46	13.89	7.88	31.60	15.92
CPLEX	6.86	8.59	6.76	2.64	0.82	1.64	7.12	14.25	12.95	6.18
lpg	42.86	10.10	37.90	3.90	27.78	20.79	20.97	11.76	38.91	2.83
probSAT	4.17	5.89	21.94	8.62	23.81	33.67	33.06	23.96	32.26	21.51
SATenstein	11.58	10.42	12.96	6.22	15.63	18.56	13.89	15.94	34.38	15.51

Q1: Parameter Importance across different Instance Sets. As shown in the right part of Table 2, the overlap of important parameters on pairs of instance sets is often quite small. Hence it depends on the instance set whether a parameter is important or not. Surprisingly, the results of ablation and fANOVA are similar in this respect. This indicates that for some algorithms (e.g. *probSAT*), a subset of the important parameters is constant across all considered instance sets. This supports the results on warmstarting of configurators [40], but also shows that warmstarting will potentially fail for some algorithms, e.g., *clasp*-(RAND), *CPLEX* and *SATenstein*.

Q2: Comparison of Local and Global Parameter Importance. As shown in Table 3, fANOVA and Ablation do not agree on the set of important parameters for most algorithms. Only for *lpg* and *clasp*(-RAND), both parameter importance approaches return some overlapping parameters, i.e., more than a third of the

parameters are on average important according to both approaches. LPI results tend to agree more with ablation results, but there is also some overlap with capped fANOVA. Thus, local and global parameter importance analysis are not redundant and indeed provide a different view on the importance of parameters.

6 Discussion and Conclusion

Algorithm configurators generate plenty of data that is full of potential to learn more about the algorithm or instance set at hand as well as the configurator itself. However, this potential so far remains largely untapped. *CAVE* provides users with the opportunity to broaden their understanding of the algorithm they want to inspect, by automatically generating comprehensive reports as well as insightful figures. We also introduced two new analysis approaches: configurator footprints and local parameter importance analysis.

We demonstrated the usefulness of such an automatic tool by using it to verify the assumption that local and global parameter importance are complementary and to demonstrate that important parameters depend on the examined set of instances.

CAVE could be further extended in many ways. In particular, we plan to analyze instance sets for their homogeneity [35]; to this extent, *CAVE* could recommend users to use per-instance algorithm configuration methods [36,45] instead of conventional configurators if the instance set is strongly heterogeneous. We also plan to improve the uncertainty estimates of *CAVE*'s EPM by replacing the random forest models by quantile regression forests [46] as shown by Eggensperger et al. [32].

Acknowledgments. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG and the Emmy Noether grant HU 1900/2-1.

References

1. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR **36**, 267–306 (2009)
2. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04244-7_14
3. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello Coello, C.A. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
4. Ansótegui, C., Malitsky, Y., Sellmann, M., Tierney, K.: Model-based genetic algorithms for algorithm configuration. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of IJCAI'15, pp. 733–739 (2015)

5. López-Ibáñez, M., Dubois-Lacoste, J., Caceres, L.P., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
6. Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). AIJ **243**, 1–25 (2017)
7. Fawcett, C., Helmert, M., Hoos, H., Karpas, E., Roger, G., Seipp, J.: Fd-autotune: domain-specific configuration using fast-downward. In: Helmert, M., Edelkamp, S. (eds.) Proceedings of ICAPS'11 (2011)
8. Mu, Z., Hoos, H.H., Stützle, T.: The impact of automated algorithm configuration on the scaling behaviour of state-of-the-Art inexact TSP solvers. In: Festa, P., Sellmann, M., Vanschoren, J. (eds.) LION 2016. LNCS, vol. 10079, pp. 157–172. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50349-3_11
9. Wagner, M., Friedrich, T., Lindauer, M.: Improving local search in a minimum vertex cover solver for classes of networks. In: Proceedings of IEEE CEC, pp. 1704–1711. IEEE (2017)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 186–202. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13520-0_23
11. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Proceedings of NIPS'12, pp. 2960–2968 (2012)
12. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: Proceedings of ICLR'17 (2017)
13. Bischl, B., et al.: ASlib: a benchmark library for algorithm selection. AIJ 41–58 (2016)
14. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Comput. OR **45**, 12–24 (2014)
15. Hutter, F., Hoos, H., Leyton-Brown, K.: Identifying key algorithm parameters and instance features using forward selection. In: Proceedings of LION'13, pp. 364–381 (2013)
16. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Proceedings of ICML'14, pp. 754–762 (2014)
17. Fawcett, C., Hoos, H.: Analysing differences between algorithm configurations through ablation. J. Heuristics **22**(4), 431–458 (2016)
18. Biedenkapp, A., Lindauer, M., Eggensperger, K., Fawcett, C., Hoos, H., Hutter, F.: Efficient parameter importance analysis via ablation with surrogates. In: Proceedings of AAAI'17, pp. 773–779 (2017)
19. Babić, D., Hutter, F.: Spear theorem prover. Solver description. SAT Competition (2007)
20. Xu, L., KhudaBukhsh, A.R., Hoos, H.H., Leyton-Brown, K.: Quantifying the similarity of algorithm configurations. In: Festa, P., Sellmann, M., Vanschoren, J. (eds.) LION 2016. LNCS, vol. 10079, pp. 203–217. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50349-3_14
21. Bussieck, M., Drud, A.S., Meeraus, A., Pruessner, A.: Quality assurance and global optimization. In Bliek, C., Jermann, C., Neumaier, A. (eds.) Proceedings of GOCOS. Lecture Notes in Computer Science, vol. 2861. Springer (2003) 223–238
22. Bussieck, M., Dirkse, S., Vigerske, S.: PAVER 2.0: an open source environment for automated performance analysis of benchmarking data. J. Glob. Optim. **59**(2–3), 259–275 (2014)
23. Rice, J.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)

24. Nell, C., Fawcett, C., Hoos, H.H., Leyton-Brown, K.: HAL: a framework for the automated analysis and design of high-performance algorithms. In: Coello Coello, C.A. (ed.) LION 2011. LNCS, vol. 6683, pp. 600–615. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_47
25. Falkner, S., Lindauer, M., Hutter, F.: SpySMAC: automated configuration and performance analysis of SAT solvers. In: Proceedings of SAT'15, pp. 1–8 (2015)
26. Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google vizier: a service for black-box optimization. In: Proceedings of KDD, pp. 1487–1495. ACM (2017)
27. Heinrich, J., Weiskopf, D.: State of the art of parallel coordinates. In: Proceedings of Eurographics, Eurographics Association, pp. 95–116 (2013)
28. Lloyd, J., Duvenaud, D., Grosse, R., Tenenbaum, J., Ghahramani, Z.: Automatic construction and natural-language description of nonparametric regression models. In: Proceedings of AAAI'14, pp. 1242–1250 (2014)
29. Hutter, F., Xu, L., Hoos, H., Leyton-Brown, K.: Algorithm runtime prediction: methods and evaluation. AIJ **206**, 79–111 (2014)
30. Breimann, L.: Random forests. MLJ **45**, 5–32 (2001)
31. Nudelman, E., Leyton-Brown, K., Hoos, H.H., Devkar, A., Shoham, Y.: Understanding random sat: beyond the clauses-to-variables ratio. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 438–452. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30201-8_33
32. Eggensperger, K., Lindauer, M., Hoos, H., Hutter, F., Leyton-Brown, K.: Efficient benchmarking of algorithm configuration procedures via model-based surrogates. Mach. Learn. (2018) (To appear)
33. Hutter, F., et al.: AClib: a benchmark library for algorithm configuration. In: Pardalos, P.M., Resende, M.G.C., Vogiatzis, C., Walteros, J.L. (eds.) LION 2014. LNCS, vol. 8426, pp. 36–40. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09584-4_4
34. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. JAIR **32**, 565–606 (2008)
35. Schneider, M., Hoos, H.H.: Quantifying homogeneity of instance sets for algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, pp. 190–204. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34413-8_14
36. Xu, L., Hoos, H., Leyton-Brown, K.: Hydra: automatically configuring algorithms for portfolio-based selection. In: Proceedings of AAAI'10, pp. 210–216 (2010)
37. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC - instance-specific algorithm configuration. In: Proceedings of ECAI'10, pp. 751–756 (2010)
38. Kruskal, J.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika **29**(1), 1–27 (1964)
39. Groenen, P., van de Velden, M.: Multidimensional scaling by majorization: A review. J. Stat. Softw. 73(8) (2016)
40. Lindauer, M., Hutter, F.: Warmstarting of model-based algorithm configuration. In: Proceedings of the AAAI conference (2018) (To appear)
41. Gerevini, A., Serina, I.: LPG: a planner based on local search for planning graphs with action costs. In: Proceedings of AIPS'02, pp. 13–22 (2002)
42. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: from theory to practice. AI **187–188**, 52–89 (2012)
43. KhudaBukhsh, A., Xu, L., Hoos, H., Leyton-Brown, K.: SATenstein: automatically building local search SAT solvers from components. In: Proceedings of IJCAI'09, pp. 517–524 (2009)

44. Balint, A., Schöning, U.: Choosing probability distributions for stochastic local search and the role of make versus break. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 16–29. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_3
45. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm selection and scheduling. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 454–469. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_35
46. Meinshausen, N.: Quantile regression forests. JMLR **7**, 983–999 (2006)