Efficient Parameter Importance Analysis via Ablation with Surrogates

André Biedenkapp and Marius Lindauer and Katharina Eggensperger and Frank Hutter
University of Freiburg
{biedenka, lindauer, eggenspk, fh}@cs.uni-freiburg.de

Chris Fawcett and Holger H. Hoos
University of British Columbia
{fawcettc, hoos}@cs.ubc.ca

Abstract

To achieve peak performance, it is often necessary to adjust the parameters of a given algorithm to the class of problem instances to be solved; this is known to be the case for popular solvers for a broad range of AI problems, including AI planning, propositional satisfiability (SAT) and answer set programming (ASP). To avoid tedious and often highly sub-optimal manual tuning of such parameters by means of ad-hoc methods, general-purpose algorithm configuration procedures can be used to automatically find performance-optimizing parameter settings. While impressive performance gains are often achieved in this manner, additional, potentially costly parameter importance analysis is required to gain insights into what parameter changes are most responsible for those improvements. Here, we show how the running time cost of ablation analysis, a well-known general-purpose approach for assessing parameter importance, can be reduced substantially by using regression models of algorithm performance constructed from data collected during the configuration process. In our experiments, we demonstrate speed-up factors between 33 and 14 727 for ablation analysis on various configuration scenarios from AI planning, SAT, ASP and mixed integer programming (MIP).

Introduction

General-purpose algorithm configuration was used a lot in the last decade to substantially improve the performance of algorithms by optimizing their parameter configurations for the class of problem instances to be solved. Examples include the configuration of state-of-the-art solvers from propositional satisfiability solving (Hutter et al. 2017), AI planning (Fawcett et al. 2011) and mixed integer-programming (Hutter, Hoos, and Leyton-Brown 2010).

Most users of algorithm configuration procedures are not only interested in obtaining a well-performing parameter configuration, but they also want to understand why the performance of their algorithm improved over another configuration. To this end, different techniques have been proposed that score parameters based on their importance (Hutter, Hoos, and Leyton-Brown 2013; 2014; Siegmund et al. 2015; Fawcett and Hoos 2016). A common observation for hard-combinatorial problem solvers is that although automated algorithm configurators change nearly all parameters (i.e., between tens and hundreds of parameters, depending on the algorithm), as few as 1 to 3 parameter changes can be responsible for nearly all of the performance improvement.

To determine these important parameter changes between two configurations, Fawcett and Hoos (2016) proposed an ablation analysis procedure. However, this analysis is computationally quite expensive and often takes many days of CPU time, since it requires further algorithm runs to validate parameter importance. Here, we propose to use a model-based surrogate instead of expensive algorithm runs to dramatically speed up this ablation analysis.

During algorithm configuration, large amounts of performance data are generated by assessing the performance of many parameter configurations on different instances. We use these data to train a regression model to predict the performance (e.g., running time) of a parameter configuration on a given instance. Such models are called empirical performance models (EPMs; Leyton-Brown, Nudelman, and Shoham; Hutter et al. 2009; 2014b). By using these cheap-to-evaluate EPMs as a surrogate, we show in extensive experiments on prominent benchmarks from SAT, MIP, ASP and AI Planning that we can speed up ablation analysis by up to 4 orders of magnitude.

We fully integrated our surrogate-based ablation procedure into the prominent, general-purpose algorithm configuration system *SMAC* (Hutter, Hoos, and Leyton-Brown 2011b)[1].

Related Work

Hutter et al. (2014b) studied different approaches to train empirical performance models (EPMs). They showed that EPMs can be effectively used to predict the performance of algorithm configurations on a set of problem instances, in particular for $\mathcal{NP}$-hard problems, such as MIP, SAT and AI planning. EPMs are often used in modern algorithm configurators, such as *SMAC* (Hutter, Hoos, and Leyton-Brown 2011b) and *GGA*++ (Ansótegui et al. 2015), to guide the search to promising areas of the configuration space; a simplified version without consideration of instances is also used in hyperparameter optimization tools for machine learning algorithms based on Bayesian optimization (e.g.,

[1] In version 3; see `http://www.ml4aad.org/smac/`

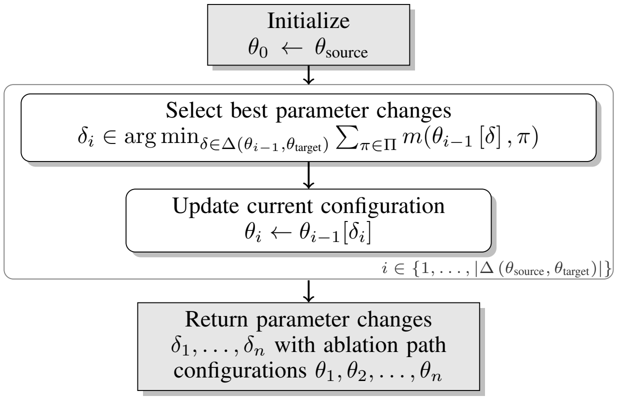
Initialize
$\theta_0 \leftarrow \theta_{\text{source}}$
Select best parameter changes
$\delta_i \in \arg\min_{\delta\in\Delta(\theta_{i-1},\theta_{\text{target}})} \sum_{\pi\in\Pi} m(\theta_{i-1}[\delta],\pi)$
Update current configuration
$\theta_i \leftarrow \theta_{i-1}[\delta_i]$
$i \in \{1,\ldots,|\Delta(\theta_{\text{source}},\theta_{\text{target}})|\}$
Return parameter changes
$\delta_1,\ldots,\delta_n$ with ablation path
configurations $\theta_1,\theta_2,\ldots,\theta_n$

Figure 1: Ablation Analysis

Shahriari et al. 2016). Before EPMs were used in algorithm configuration, they were already a key technique in per-instance algorithm selection (see, e.g., Rice 1976, Kotthoff 2014); furthermore, they have recently been used to define surrogate benchmarks for efficient benchmarking of hyper-parameter optimization tools (Eggensperger et al. 2015).

Other approaches for parameter importance analysis also make use of EPMs; an example is forward selection to determine which algorithm parameters are the most useful features to train an EPM that predicts the algorithm's performance (Hutter, Hoos, and Leyton-Brown 2013). A related approach is *fANOVA* (Hutter, Hoos, and Leyton-Brown 2014), which uses an EPM (based on random forests (Breiman 2001)) to analyze how much of the performance variance in the configuration space is explained by single parameters or combinations of few parameters. Both of these approaches follow a similar idea as used in this paper: during algorithm configuration, performance data is gathered on different parameter configurations and instances, and this data is later used to train an EPM, without any additional expensive algorithm runs. An exemplary implementation of this workflow is *SpySMAC* (Falkner, Lindauer, and Hutter 2015), an algorithm configuration toolkit supporting extensive post-configuration analysis. However, both approaches differ from ours in that they are limited to a global view of the configuration space and hence fail to capture important local effects responsible for the efficacy of top-performing configurations of a given algorithm. Furthermore, we are not aware of any work showing that such EPMs provide accurate enough predictions for local effects.

Background: Ablation

Ablation analysis is a recent approach that aims to answer more localized parameter importance questions. Fawcett and Hoos (2016) showed that algorithm performance improvements between two parameter configurations can often be explained by only a few parameter changes (typically at most 3); for example, a 473-fold performance speedup of the SAT solver *Spear* (Babić and Hutter 2007) on software verification instances was almost completely (i.e., 99.7%) explained by a single parameter change.

We now describe ablation analysis on a high level. Given a parameterised algorithm $\mathcal{A}$ with n configurable parameters defining a configuration space $\mathbf{\Theta}$, along with a *source configuration* θ_{source} and *target configuration* θ_{target} (e.g., a user-defined default configuration and one obtained using automated configuration), and a performance metric m (e.g., running time or solution quality), ablation analysis first computes the parameter setting differences $\Delta\left(\theta_{\text{source}},\theta_{\text{target}}\right)$ between the source and target configurations. Next, given a set Π of benchmark instances, an *ablation path* $\theta_{\text{source}},\theta_1,\theta_2,\ldots,\theta_{\text{target}}$ is iteratively constructed. In each iteration i with previous ablation path configuration θ_{i-1}, we consider all remaining parameter changes $\delta\in\Delta\left(\theta_{i-1},\theta_{\text{target}}\right)$ and apply the change to the previous ablation path configuration $\theta_{i-1}\left[\delta\right]$. Each parameter change δ is a modification of one parameter from its value in θ_{i-1} to its value in θ_{target}, along with any other parameter modifications that may be necessary due to conditionality constraints in $\mathbf{\Theta}$. The next configuration on the ablation path θ_i is the candidate $\theta_{i-1}\left[\delta\right]$ with the best performance m on the set Π. This performance may be directly measured by performing runs of $\mathcal{A}$, or approximated for efficiency reasons. A concise summary of ablation analysis is given in Figure 1.

Fawcett and Hoos (2016) introduced two variants of ablation analysis. The first, full ablation analysis, performs a full empirical analysis for each candidate configuration θ in a given ablation iteration, by running $\mathcal{A}\left[\theta\right]$ on all instances $\pi\in\Pi$. This can be computationally very expensive and, given that many configurations in each iteration perform poorly, quite wasteful of those computational resources. The second approach, racing-ablation analysis, reduces the computational burden by applying the statistical techniques introduced in F-race (Birattari et al. 2002; López-Ibáñez et al. 2016) to eliminate some candidate configurations as soon as there is enough statistical evidence on an iteratively increased subset of instances that they are outperformed by at least one other configuration. This procedure efficiently rejects poorly-performing candidate configurations after as few as 5 runs, dramatically reducing the computational costs of ablation analysis. However, there is a risk associated with these substantial savings: if the first instances considered favor a different parameter change than the entire set of instances would, then racing-ablation can fail to identify the best ablation path (we will show an example of this in our experiments). Also, if several candidate configurations in an ablation iteration have very similar performance (or remaining candidates have no effect on performance), no benefit is gained from racing.

We note that ablation has several advantages and disadvantages compared to the EPM-based methods of forward selection and *fANOVA* discussed in the related work section. The latter two produce parameter importance results based on a global view of the entire configuration space of the algorithm under consideration; since this can contain several regions of similarly-high performance, this global view may miss characteristics that are (only) important in the regions close to the optimal configuration. In contrast, the locality of ablation has the benefit of quantifying how much can be gained by simply changing a few parameters from an algorithm's tried-and-tested default configuration θ_{source}. However, ablation is limited to analyzing parameters that differ between θ_{source} and θ_{target}. Due to its iterative nature, it can also only partially identify interactions between parameters (e.g., by using a forward and backward pass of ablation). Therefore, ablation often yields results that complement the global analysis of forward selection and *fANOVA*. Finally, a key issue of ablation that complicates fast posthoc analyses is its reliance on new expensive algorithm runs; in this paper, we remove this issue by introducing an EPM-based ablation variant that does not require *any* new algorithm runs.

Efficient Ablation via Surrogates

As previously mentioned, ablation analysis is an expensive process due to the algorithm runs that have to be performed to gather enough empirical evidence to compare parameter configurations. For example, Fawcett and Hoos (2016) reported a running time of 5 CPU days for full ablation analysis and 1 CPU day for racing-ablation for the *Spear* example on software verification instances mentioned above. We note that this is only a moderately hard example with a cutoff of 300 seconds for each run of *Spear*.

To use ablation analysis in situations where CPU time is limited, we propose to accelerate the process of ablation by replacing expensive algorithm runs with cheap predictions obtained from a model-based surrogate. When selecting the best parameter changes δ_i (see Figure 1, second block), we replace the empirical performance $m : \mathbf{\Theta} \times \Pi \rightarrow \mathbb{R}$ by a predicted performance $\hat{m} : \mathbf{\Theta} \times \Pi \rightarrow \mathbb{R}$ for configurations $\theta \in \mathbf{\Theta}$ and instances $\pi \in \Pi$ using an EPM (Hutter et al. 2014b). Since typical machine learning models can produce predictions within fractions of a second, once an EPM is trained, our approach computes an ablation path virtually instantaneously. (See Table 3 in the last section for a comparison of its running time with racing.)

The application of our surrogate-ablation requires: i) Instance features as numerical representations of instances; these are available for many domains (Xu et al. 2008; 2011; Gebser et al. 2011; Fawcett et al. 2014) or can be generated in a domain-independent fashion (Loreggia et al. 2016); ii) Gathering sufficient performance data to train an EPM; iii) Obtaining reasonably accurate predictions from the EPM.

To address the latter two requirements, we discuss two points in the following subsections: i) How to construct EPMs that serve as good surrogates for ablation analysis, and ii) How to deal with uncertainty and prediction error induced by the EPM.

Fitting an Empirical Performance Model (EPM)

Our goal is to construct an EPM $\hat{m} : \mathbf{\Theta} \times \Pi \rightarrow \mathbb{R}$ and to use its predictions as a surrogate for expensive real algorithm runs. Running time is a common performance metric to optimize in algorithm configuration (in particular for solving hard combinatorial problems), and Hutter et al. (2014b) showed that random forests are currently the best known machine learning model for this task, with better results obtained when predicting log-transformed running time. Therefore, we follow this approach.

We now discuss how to efficiently gather data for the training of the EPM and how to compute accurate predictions of the commonly used performance metric of penalized average runtime.

Gathering Training Data As discussed above, parameter importance analysis is an important tool after applying algorithm configuration, since algorithm configuration tools often do not reveal which parameter changes lead to improved performance. Thus, we expect that our ablation analysis will mainly be used after algorithm configuration. In our surrogate-ablation workflow we reuse all algorithm performance data collected during configuration. These data consist of performance evaluations $m(\theta, \pi)$ of different configurations θ and instances π.

Since algorithm configuration procedures focus search in high-performance regions of the configuration space and thus gather more data there, these data are well suited to train our EPM. To apply surrogate-ablation analysis, we need an EPM that predicts well in these high-performance regions, because target configurations will likely stem from such regions, and we want to know which parameters have high impact on the performance within such regions.

An issue with the data from algorithm configuration procedures is that the data from algorithm runs that were prematurely terminated is right-censored, i.e., we only obtain a lower-bound on the true performance for the respective runs. This is a result of *adaptive capping* as used in *ParamILS* (Hutter et al. 2009) and *SMAC* (Hutter, Hoos, and Leyton-Brown 2011b): If the performance metric to be optimized is running time, the configurator does not require algorithm runs to use the entire running time cutoff when deciding which of two configurations perform better. Therefore we need to impute these censored data, e.g., by the methods of Schmee and Hahn (1979) or Hutter, Hoos, and Leyton-Brown (2011a).

Predicting Expected Penalized Running Time Using a running time cutoff for the target algorithm ensures that (i) an algorithm does not run forever, and that (ii) poorly-performing configurations do not waste too much time. Algorithm configurators therefore do not directly optimize running time but a censored version of it, known as penalized average running time (PARX). PARX penalizes each timed-out or crashed run as X times the running time cutoff.

Using PARX as a performance metric for EPM predictions is, however, not an easy task, because the EPM has to predict a large jump in performance on the boundary between successful runs and timed-out runs. Since the EPM (as all machine learning models) has some errors in its predictions, the prediction of a new pair of $\langle\theta, \pi\rangle$ close to this boundary could be on either side of the boundary. Therefore, mean predictions (e.g, of a random forest) can have a large error compared to the true performance value if the prediction is on the wrong side of the boundary.

To cope with this problem, we propose to predict *expected* penalized running time (EPARX) instead of penalized running time. The idea is that the prediction from our EPM is

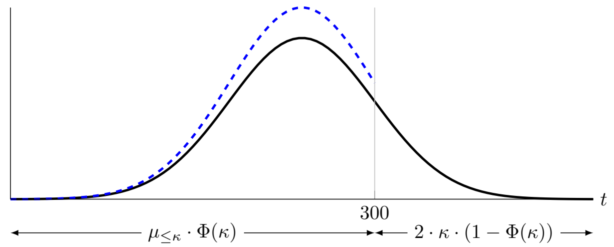

Figure 2: Visualization of expected PAR2 with $\kappa = 300$, black being the normal distribution $\mathcal{N}(\mu = 250, \sigma = 50)$ and dashed blue being the corresponding truncated normal distribution $\mathcal{N}_{\leq\kappa}$. The expected PAR2 would be 293.

not a single value for the running time, but a distribution over the predicted running time. E.g., a regression random forest (Breiman 2001) can be adapted to return a mean prediction and a variance estimate of a normal distribution (Hutter et al. 2014b). To predict EPARX for a distribution with probability density $p(t)$, we predict the expected running time below κ and the penalized score of $X \cdot \kappa$ weighted by the probability to be above κ as

$$\int_0^\kappa t \cdot p(t)\mathrm{d}t + \int_\kappa^\infty X \cdot \kappa \cdot p(t)\mathrm{d}t. \qquad (1)$$

If we approximate the distribution $p(t)$ by a normal distribution (e.g., using a random forest or a Gaussian process), we can solve Equation 1 by:

$$\mu_{\leq\kappa} \cdot \Phi(\kappa) + X \cdot \kappa \cdot (1 - \Phi(\kappa)), \qquad (2)$$

where κ is the running time limit, μ the predicted mean performance, σ^2 the predicted variance, Φ the CDF of a normal distribution $\mathcal{N}(\mu, \sigma^2)$ of the predicted mean μ and predicted variance σ^2, and $\mu_{\leq\kappa}$ the mean of the truncated normal distribution $\mathcal{N}(\mu, \sigma^2)_{\leq\kappa}$.

Figure 2 illustrates expected penalized average running time on an artificial example with penalization of $X = 2$ and a running time cutoff $\kappa = 300$. The black curve is the normal distribution $p(t)$ for a given mean prediction $\mu = 250$ and predicted standard deviation $\sigma = 50$. The probability of not being a timeout is $\Phi(\kappa) = 0.84$, the mean of the truncated normal distribution is $\mu_{\leq\kappa} = 235.62$. Using Equation 2, the expected penalized average running time is 293.43 in contrast to the predicted penalized average running time of 250.

Handling Uncertainty of EPMs

In contrast to the original ablation method, our method is much faster, but—like ablation—it also comes with the risk of wrong predictions. As shown by Hutter et al. (2014b) and Eggensperger et al. (2015), random forest models are well suited as EPMs, but we cannot guarantee that their predictions are always accurate enough.

Possible reasons EPMs having large errors include (i) uninformative instance features that do not permit us to discriminate well between instances; (ii) an insufficient amount of training data, e.g., due to a short algorithm configuration run that evaluated only few parameter configurations within a large configuration space; (iii) failure of the training data to capture the part of the configuration space spanned by the source and target configuration for ablation.

This can obviously lead to inaccurate ablation results; to quantify this risk, we need to assess the error made by the EPM used for surrogate-ablation. An approach to deal with the uncertainty of our random forests as EPMs is to use their variance predictions to indicate how certain they are about the predicted performance of a changed parameter. In contrast to the original ablation path, which can be visualized as a single curve in the space of parameter modifications and performance, in the following, we also compute the uncertainty in the predicted performance of each modification.

Experimental Study

To compare full ablation, racing-ablation and surrogate-ablation, we ran experiments on six algorithm configuration benchmarks from propositional satisfiability (SAT), mixed integer programming (MIP), answer set programming (ASP) and AI planning.

Setup

Our surrogate-ablation implementation in Python is freely available[2] and fully integrated into the algorithm configuration system *SMAC*. All experiments were executed on a compute cluster equipped with two Intel Xeon E5-2650v2 8-core CPUs, 20 MB L2 cache and 64 GB of (shared) RAM per node, running Ubuntu 14.04 LTS 64 bit.

As configuration benchmarks, we used the following six benchmarks from the algorithm configuration literature as implemented in AClib (Hutter et al. 2014a)[3]; details for these benchmarks are shown in Table 1:

SPEAR-QCP and SPEAR-SWV *Spear* (Babić and Hutter 2007) is a tree-search SAT solver. We configured *Spear* on quasigroup completion instances (QCP) (Gomes and Selman 1997) and software verification instances (Babić and Hu 2007), following Hutter et al. (2009).

CPLEX-RCW2 *CPLEX* is one of the most widely used MIP solvers. We configured *CPLEX* on the Red-cockaded Woodpecker (RCW2) instance set (Ahmadizadeh et al. 2010), following Hutter, Hoos, and Leyton-Brown (2010).

CLASP-WS *Clasp* (Gebser, Kaufmann, and Schaub 2012) is a state-of-the-art solver for ASP. We configured *Clasp* on weighted-sequence problems (WS) for database query optimization (Lierler et al. 2012), following Silverthorn, Lierler, and Schneider (2012).

LPG-SATELLITE and LPG-DEPOTS *LPG* (Gerevini, Saetti, and Serina 2003) is a local-search AI planning solver. We optimized *LPG* on the two domains DEPOTS and SATELLITE, following Fawcett and Hoos (2016).

For each benchmark, we performed configuration and ablation experiments. For the configuration part, we used 16 independent *SMAC* runs and picked the configuration with

[2] `http://www.ml4aad.org/efficient-ablation/`

[3] `www.aclib.net`

Benchmark	#P	κ	#Inst.	Budget	#Data
		[sec]	Train/Test	[h]	
SPEAR-QCP	26	5	976/2000	80	$200k$
SPEAR-SWV	26	300	302/302	768	$200k$
CPLEX-RCW2	76	10 000	495/495	768	$33k$
CLASP-WS	99	900	240/240	1536	$119k$
LPG-SATELLITE	66	300	2000/2000	768	$200k$
LPG-DEPOTS	66	300	2000/2000	768	$200k$

Table 1: Overview of used configuration benchmarks from AClib. #P is the number of parameters, κ is the running time cutoff, #Inst is the number of instances in the training and test set, Budget is the running time required to run *SMAC* 16 times, and #Data is the number of collected data points.

the best performance on the training instances as the ablation target θ_{target}. The performance metric was penalized average running time with a commonly used penalization factor of $X = 10$ (PAR10). Then, to obtain a ground truth for ablation, we ran a full ablation analysis using all training instances. Following Fawcett and Hoos (2016), this used the training instances to compute an ablation path from θ_{source} (the default parameter configuration) to θ_{target} and evaluated this path on the test set.

We next compared racing-ablation and surrogate-ablation against the ground truth obtained from the full analysis using the same θ_{source} and θ_{target}. Since full ablation is too expensive for many real-world benchmarks, and racing-ablation is currently the only computationally feasible method, we show that surrogate-ablation yields similarly good results as racing ablation, but does so much faster. In particular, while racing-ablation (like full ablation) requires new algorithm runs, surrogate-ablation only requires EPM predictions. In racing, following Fawcett and Hoos (2016), we set the maximum number of rounds to 200. To train our EPMs, for each benchmark, we used at most 200 000 performance data points, because in our (not memory-optimized) implementation, using more data points exhausted memory of 8 GB RAM. Since our goal is to run our surrogate-ablation analysis on a typical office machine, using more than 8 GB RAM would be unreasonable in this setting.

We will judge the similarity between each method's ablation path and the ground truth ablation paths by two metrics.[4] Both of these are related to the importance score that each modification receives for each path, i.e., the estimated performance improvement. We define a modification as *important* in a path if it is part of a prefix of the path each member of which has at least 5% performance improvement. Our two metrics are then:

- The similarity between the set of parameter modifications deemed important: We count true positives, false positives, and false negatives of racing-ablation and surrogate-ablation w.r.t. to the important parameters in the ground truth data. E.g., false positives refer to classifying unimportant parameters (w.r.t. the ground truth) as important.
- The speedup explained by changing the n first parameters, where n is w.r.t. the number of important parameters in the ground truth data. The n can be seen in Table 2 by adding the numbers of true positives and false negatives.

	TP/FP/FN		Speedup		
benchmark	R	S	F	R	S
SPEAR-QCP	1/1/1	1/1/1	2.4	2.6	1.9
SPEAR-SWV	0/2/1	1/1/0	332.6	1.0	332.6
CPLEX-RCW2	1/0/0	1/1/0	1.2	1.2	1.2
CLASP-WS	0/0/1	0/5/1	1.9	0.4	0.4
LPG-DEPOTS	1/0/0	1/0/0	23.0	23.0	23.0
LPG-SATELLITE	2/0/0	1/1/1	2.9	2.9	2.4

Table 2: Comparing racing-ablation and surrogate-ablation to a full ablation analysis ("F"). "R" corresponds to racing-ablation evaluated on all test instances, "S" to surrogate-ablation. "TP/FP/FN" shows the numbers of true positives, false positive and false negatives wrt the set of the most important parameters compared to the ground truth. "Speedup" shows the speedup explained after the first important parameter changes.

Results

Table 2 shows that, overall, surrogate-ablation performs similarly well as racing-ablation. Both methods correctly identified the most important parameters for most benchmarks, with few false positives and false negatives. In particular, surrogate-ablation explained the speedups caused by the important parameter changes well. Two of our six benchmarks showed some unexpected characteristics, i.e., SPEAR-SWV and CLASP-WS. On SPEAR-SWV, surrogate-ablation correctly identified the most important parameter change correctly and was able to explain a speedup of 332.6 between the default configuration and the optimized parameter configuration of *Spear*. However, SPEAR-SWV is an example for the previously explained flaw of racing: racing-ablation used an unrepresentative subset of instances in early iterations of the racing and hence, wrongly eliminated the most important parameter change. Since surrogate-ablation always reasons over the entire instance set, our new ablation method is not affected by such heterogeneous instance sets. On CLASP-WS, the training instances were also slightly heterogeneous and furthermore not representative of the test instances. Therefore, both ablation methods failed and even full ablation was not able to find a reasonable ablation path on the training set that generalizes well to the test instances.

To provide more detailed insight into our surrogate-ablation method, Figure 3 shows the first 5 parameter modifications of our surrogate-ablation and the ground truth. For LPG-SATELLITE, surrogate-ablation correctly identified the most important parameter (`cri_intermediate_levels`). For the second-most important, the full ablation chose a combined change of `vicinato` and `hpar_cut_neighb` due to conditional pa-

[4] We note that we cannot simply compute Spearman's correlation coefficients because we are mainly interested to identify the most important parameters and we are not interested in the order of the unimportant parameters.

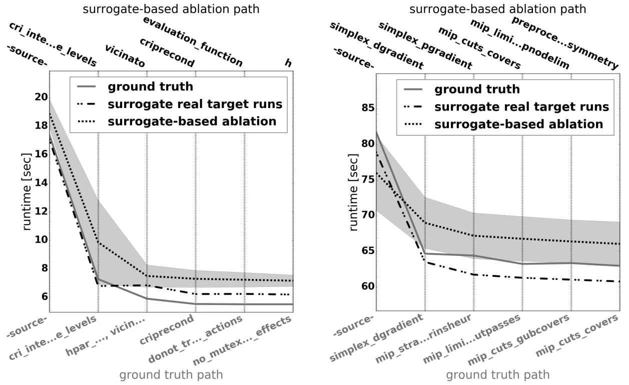

Figure 3: First five parameter changes in the ablation paths of the ground truth and surrogate-ablation for LPG-SATELLITE (left) and CPLEX-RCW2 (right). The grey area around the surrogate-ablation path is the estimated performance standard deviation.

rameters in the configuration space of *LPG*. Our surrogate-ablation has not learned the interaction between these two parameters and changed only `vicinato`. Although the third parameter is already nearly unimportant, our ablation method also found this one. For CPLEX-RCW2, the surrogate identified the most important parameter correctly (`simplex_dgradient`). However, it scored the importance of this modification lower than the ground truth.

Across all benchmarks, we observed that the predicted parameter importance scores along the surrogate-ablation paths drop smoother than in the ground truth, i.e., surrogate-ablation overestimated the importance of some parameters. This is consistent with the data in Table 2, since it has less false negatives than false positives. We believe that false positives are less severe in practice than false negatives because missing an important parameter distracts users more than looking into some more pretended important parameters.

Both figures also show the estimated standard deviation of the surrogate-ablation to illustrate the uncertainty of the parameter importance. The estimated standard deviation for LPG-SATELLITE is much smaller than for CPLEX-RCW2. Interestingly, the estimated standard deviation is nearly constant across the changes for CPLEX-RCW2. For LPG-SATELLITE, the EPM is much more certain at the source configuration (i.e., the default configuration of *LPG*) and more uncertain after the first parameter change. The reason is that during configuration, we evaluated θ_{source} and θ_{target} on more instances than the configurations in the middle of the ablation path.

Table 3 shows the running time of the different ablation approaches. For surrogate-ablation, we counted only the time to compute and validate the paths, but we did not include the time to gather the EPM training data or to train the EPM. Since we used *SMAC* as an algorithm configurator, which also fits an EPM to guide its search, we can simply save the EPM to disk and reuse it later such that we do not need to refit an EPM for the ablation analysis. Retraining the EPM would cost in our implementation on average 34 (±30) minutes, which is mostly due to the large amount of training samples (see Table 1) and the imputation of right-censored data. As expected, both original ablation methods needed between hours and days to compute and evaluate ablation paths, but our surrogate-ablation needed only a few minutes, making it between 33 and 14 727 times faster than racing-ablation.

	Full		Racing		Surro.	
benchmark	Train	Test	Train	Test	Train	Test
SPEAR-QCP	921	78	91	68	4	0.75
SPEAR-SWV	853	44	316	71	1	0.20
CPLEX-RCW2	121 279	11 639	21 290	11 552	2	0.23
CLASP-WS	159 799	8 266	57 689	8 323	6	0.50
LPG-DEPOTS	30 556	1 023	366	1 060	10	0.80
LPG-SATELLITE	113 126	6 533	5 783	6 162	21	1.17

Table 3: Comparing based on required time [CPU min] to do analysis. Train refers to the time required to compute the ablation paths on the training instances and test refers to the validation of the ablation paths on the test instances to derive parameter importance scores.

Discussion

By using performance predictions instead of actual target algorithm runs, our surrogate-ablation dramatically speeds up ablation analysis, while producing qualitatively similar results to racing-ablation. Up to now, running ablation analysis often required days or weeks of CPU time, and hence necessitated the use of sizeable compute clusters. With surrogate-ablation, comparable results are obtained within minutes on a typical desktop machine, while avoiding the risk of obtaining misleading results for heterogeneous instance sets that is inherent to all racing-based methods.

Since surrogate-ablation depends on the prediction accuracy of the underlying empirical performance model, our implementation provides uncertainty bounds in the ablation path, based on which users can assess the accuracy of the parameter importance information determined by ablation.

An open issue is handling of large amounts of data. Because of memory issues, we already had to subsample our EPM training data. Since deep neural networks have shown impressive results on big data, in the future we plan to study such networks as EPMs. Another direction for EPMs would be to use the random forest adaptation of *GGA*++ (Ansótegui et al. 2015) to obtain better predictions in high-performance regions. Furthermore, when applied to heterogeneous instance sets, ablation could be combined with *ISAC* (Kadioglu et al. 2010).

Acknowledgments K. Eggensperger, M. Lindauer and F. Hutter acknowledge funding by the DFG (German Research Foundation) under Emmy Noether grant HU 1900/2-1; K. Eggensperger also acknowledges funding by the State Graduate Funding Program of Baden-Württemberg. H. Hoos acknowledges funding through an NSERC Discovery Grant.

References

Ahmadizadeh, K.; Dilkina, B.; Gomes, C.; and Sabharwal, A. 2010. An empirical study of optimization for maximizing diffusion in networks. In *Proc. of CP'12*, 514–521.

Ansótegui, C.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2015. Model-based genetic algorithms for algorithm configuration. In *Proc. of IJCAI'15*, 733–739.

Babić, D., and Hu, A. 2007. Structural Abstraction of Software Verification Conditions. In *Proc. of CAV'07*, 366–378.

Babić, D., and Hutter, F. 2007. Spear theorem prover. Solver description, SAT competition.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A racing algorithm for configuring metaheuristics. In *Proc. of GECCO'02*, 11–18.

Breiman, L. 2001. Random forests. *MLJ* 45(1):5–32.

Eggensperger, K.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2015. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proc. of AAAI'15*, 1114–1120.

Falkner, S.; Lindauer, M.; and Hutter, F. 2015. SpySMAC: Automated configuration and performance analysis of SAT solvers. In *Proc. of SAT'15*, 1–8.

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

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Roger, G.; and Seipp, J. 2011. Fd-autotune: Domain-specific configuration using fast-downward. In *Proc. of ICAPS'11*.

Fawcett, C.; Vallati, M.; Hutter, F.; Hoffmann, J.; Hoos, H.; and Leyton-Brown, K. 2014. Improved features for runtime prediction of domain-independent planners. In *Proc. of ICAPS'14*. AAAI.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T.; Schneider, M.; and Ziller, S. 2011. A portfolio solver for answer set programming: Preliminary report. In *Proc. of LPNMR'11*, 352–357.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *AI* 187-188:52–89.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *JAIR* 20:239–290.

Gomes, C., and Selman, B. 1997. Problem structure in the presence of perturbations. In *Proc. of AAAI'97*, 221–226.

Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An automatic algorithm configuration framework. *JAIR* 36:267–306.

Hutter, F.; López-Ibánez, M.; Fawcett, C.; Lindauer, M.; Hoos, H.; Leyton-Brown, K.; and Stützle, T. 2014a. AClib: a benchmark library for algorithm configuration. In *Proc. of LION'14*.

Hutter, F.; Xu, L.; Hoos, H.; and Leyton-Brown, K. 2014b. Algorithm runtime prediction: Methods and evaluation. *AIJ* 206:79–111.

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

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. In *Proc. of CPAIOR'10*, 186–202.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011a. Bayesian optimization with censored response data. In *BayesOpt workshop at NIPS'11*.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011b. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION'11*, 507–523.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2013. Identifying key algorithm parameters and instance features using forward selection. In *Proc. of LION'13*. 364–381.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2014. An efficient approach for assessing hyperparameter importance. In *Proc. of ICML'14*, 754–762.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC - instance-specific algorithm configuration. In *Proc. of ECAI'10*, 751–756.

Kotthoff, L. 2014. Algorithm selection for combinatorial search problems: A survey. *AI Magazine* 48–60.

Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2009. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of ACM* 56(4).

Lierler, Y.; Smith, S.; Truszczynski, M.; and Westlund, A. 2012. Weighted-sequence problem: ASP vs CASP and declarative vs problem-oriented solving. In *Proc. of PADL*, 63–77.

López-Ibáñez, M.; Dubois-Lacoste, J.; Caceres, L. P.; Birattari, M.; and Stützle, T. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3:43–58.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. 2016. Deep learning for algorithm portfolios. In *Proc. of AAAI'16*, 1280–1286.

Rice, J. 1976. The algorithm selection problem. *Advances in Computers* 15:65–118.

Schmee, J., and Hahn, G. 1979. A simple method for regression analysis with censored data. *Technometrics* 21:417–432.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R.; and de Freitas, N. 2016. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE* 104(1):148–175.

Siegmund, N.; Grebhahn, A.; Apel, S.; and Kästner, C. 2015. Performance-influence models for highly configurable systems. In *Proc. of ESEC/FSE'15*, 284–294.

Silverthorn, B.; Lierler, Y.; and Schneider, M. 2012. Surviving solver sensitivity: An ASP practitioner's guide. In *Technical Communications of ICLP'12*, 164–175.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *JAIR* 32:565–606.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop at IJCAI'11*.