An Efficient Approach for Assessing Algorithm Parameter Importance

Keywords: Search, Satisfiability, Performance modelling, Statistical analysis

Abstract

Automated algorithm configuration tools can find good settings even for highly parametric algorithms. However, blind reliance on such tools can leave end users without insight into the relative importance of an algorithm's parameters, and ways in which these parameters interact. This paper describes efficient tools that can be used to gain such insight, leveraging predictive models of algorithm performance. We begin by introducing a novel, linear-time algorithm for computing marginals of random forest predictions. We show how to leverage this algorithm within a functional ANOVA framework to quantify the importance both of single parameters and of interactions between combinations of parameters. We validated our approach in experiments considering state-of-the-art algorithms for propositional satisfiability, mixed integer programming, answer set programming, and the traveling salesman problem, showing how to use our methods to identify small sets of parameters that explain large fractions of an algorithm's performance variation.

1 Introduction

Automated algorithm configuration (AC) procedures (see e.g., [Birattari *et al.*, 2002; Nannen and Eiben, 2007; Ansotegui *et al.*, 2009; Hutter *et al.*, 2009; 2011]) are becoming increasingly popular tools for optimizing the parameters of heuristic algorithms for many hard computational problems. Indeed, the growing efficacy of these procedures is leading algorithm designers to expose ever-more design choices as parameters, making highly parameterized algorithms more and more common. This trend has resulted in a wide range of improvements in the state of the art on solving many instance distributions of practical interest, demonstrating the limitations of the hand-tuning paradigm that preceded it. However, hand-tuning had one great advantage: it led algorithm designers to develop rich intuition about the interaction between their design choices and their algorithm's performance. An algorithm designer who instead relies upon a black-box AC procedure is likely to have trouble answering questions like the following: "How important is each of the parameters, and how do their values affect performance? Which parameter interactions matter? How do the answers to these questions differ across instance distributions?" There has been relatively little work on developing tools to help answer such questions. Notable exceptions in the literature include experimental design based on linear models [Ridge and Kudenko, 2006; Chiarandini and Goegebeur, 2010], an entropy-based measure [Nannen and Eiben, 2007], and visualization methods for interactive parameter exploration, such as contour plots [Bartz-Beielstein, 2006]. However, to the best of our knowledge, none of these methods has yet been applied to study the configuration spaces of state-of-the-art highly parametric solvers; indeed, their applicability is questionable, due to the high dimensionality of these spaces and the ubiquity of discrete parameters (which, e.g., linear models cannot handle gracefully). The only work we are aware of that *does* pertain to large configuration spaces applies a forward selection approach to determine important parameters [Hutter *et al.*, 2013]. In this paper, we follow a similar model-based approach, but leverage a new statistical analysis technique: *functional ANOVA*. Unlike forward selection, functional ANOVA has rich theoretical foundations, can assess interaction effects, and is computationally efficient (it requires constructing only a single model, while forward selection typically requires thousands).

This paper begins by introducing a new, linear-time algorithm for computing the marginals of random forest predictions (Section 2). We show how this algorithm can be leveraged to tractably identify main and (low-order) interaction effects within the functional ANOVA framework (Section 3). Finally, we demonstrate the power of this approach through an extensive experimental evaluation (Section 4). These experiments consider 8 state-of-the-art solvers and 12 benchmark distributions, spanning mixed integer programming, propositional satisfiability, answer set programming and the traveling salesperson problem. We investigate the extent to which performance variation can be explained through both main and interaction effects, compare model predictions to ground truth, and contrast two different performance measures (raw performance variation and variation in the improvement over a given default). Finally, we show that automated configuration in the restricted space of parameters that explain most performance variation can recover most of the gains of configuration in the full space, despite being a vastly easier problem.

2 Efficient Marginal Performance Predictions

An algorithm designer who wanted to manually assess parameter importance would be likely to investigate the local neighbourhood of a given parameter setting: vary one parameter at a time and measure how performance changes. Note that the only information obtained with this analysis is how different parameter values perform *in the context of a single instantiation* of the other parameters. Optimally, algorithm designers would like to know how their parameters affect performance in general, not just in the context of a single fixed instantiation of the remaining parameters but across all their instantiations. Unfortunately, performing algorithm runs for all these instantiations is infeasible in all but the easiest cases since there are D^k such instantiations of k discrete parameters with domain size D. (Continuous parameters are even worse, having infinitely many instantiations.) As it turns out, an approximate analysis based on predictive performance models can be used to solve this problem and quantify the performance of a parameter instantiation in the context of *all instantiations of the other parameters*. In this section, we will show that this *marginal performance* of a partial parameter instantiation can be predicted by computing the required infinite (or exponential) sum of predictions in linear time. We first cover some notation and define the problem formally. Then, we introduce an algorithm to predict this marginal performance using predictive models based on random forests and prove its correctness and linear time complexity.

2.1 Problem Definition

We begin with some basic definitions. Let A be an algorithm having n parameters with domains $\Theta_1, \ldots, \Theta_n$. We use integers to denote the parameters, and N to refer to the set $\{1, \ldots, n\}$ of all parameters of A.

Definition 1 (Configuration space $\boldsymbol{\Theta}$). *A's* configuration space *is* $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_n$.

Definition 2 (Parameter Instantiation). *A* complete instantiation *of an algorithm's n parameters is a vector $\boldsymbol{\theta} = \langle \theta_1, \ldots, \theta_n \rangle$ with $\theta_i \in \Theta_i$. We also refer to complete parameter instantiations as* parameter configurations. *A* partial instantiation *of a subset $U = \{u_1, \ldots, u_m\} \subseteq N$ of A's parameters is a vector $\boldsymbol{\theta}_U = \langle \theta_{u_1}, \ldots, \theta_{u_m} \rangle$ with $\theta_{u_i} \in \Theta_{u_i}$.*

It has recently been shown that it is possible to fit machine learning models that predict the performance of an algorithm with new parameter configurations and on new problem instances [Brewer, 1995; Xu *et al.*, 2008; Gagliolo and Legrand, 2010; Smith-Miles and Lopes, 2012]. In an extensive computational study, Hutter et al. [2012] compared a variety of different machine learning models for predicting the performance of highly parametric solvers for NP-hard problems, concluding that random forests [Breiman, 2001] yielded the best performance. Random forests are ensembles that average across several regression trees. Regression trees are tree-based predictors similar to decision trees, but (as regression models) their leaves hold continuous response values rather than class labels. A regression tree partitions its input space through sequences of branching decisions that lead to each of its leaves. It will be useful in what follows to define this formally. Denote the partitioning as $\mathcal{P}$; each equivalence class $P_i \in \mathcal{P}$ is associated with a leaf of the regression tree. Let $\Theta_j^{(i)} \subset \Theta_j$ denote the subset of domain values of parameter j that is consistent with the branching decisions leading to the leaf associated with P_i. Then, P_i is simply the Cartesian product

$$P_i = \Theta_1^{(i)} \times \cdots \times \Theta_n^{(i)}, \tag{1}$$

and has an associated constant prediction c_i. We can use this definition to formally define the predictor $\hat{y} : \boldsymbol{\Theta} \mapsto \mathbb{R}$ encoded by a regression tree:

$$\hat{y}(\boldsymbol{\theta}) = \sum_{P_i \in \mathcal{P}} \mathbb{I}(\boldsymbol{\theta} \in P_i) \cdot c_i, \tag{2}$$

where $\mathbb{I}(x)$ is the indicator function.

The extension set for a partial parameter instantiation is the set of parameter instantiations that are consistent with it.

Definition 3 (Extension Set). *Let $\boldsymbol{\theta}_U$ be a partial instantiation of the parameters $U = \{u_1, \ldots, u_m\} \subseteq N$. We define $X(\boldsymbol{\theta}_U)$, the* extension set of *$\boldsymbol{\theta}_U = \langle \theta'_{u_1}, \ldots, \theta'_{u_m} \rangle$, as the set of parameter instantiations $\boldsymbol{\theta}_{N|U} = \langle \theta_1, \ldots, \theta_n \rangle$ such that $\forall j (j = u_k \Rightarrow \theta_j = \theta'_{u_k})$.*

Definition 4 (Range size). *The* range size *$||S||$ of a set S is its arity $|S|$ if S is finite; for closed intervals $S = [l, u] \subset \mathbb{R}$ we define $||S|| = u - l$. For cross-products $S = S_1 \times \cdots \times S_k$, $||S|| = \prod_{i=1}^{k} ||S_i||$.*

The marginal (predicted) performance of a partial instantiation $\boldsymbol{\theta}_U$ is the expected (predicted) performance of A over the settings in $\boldsymbol{\theta}_U$'s extension set.

Definition 5 (Marginal performance). *Let A's (true) performance be $y : \boldsymbol{\Theta} \mapsto \mathbb{R}$, $U \subseteq N$, and $T = N \setminus U$. The* marginal performance *$a_U(\boldsymbol{\theta}_U)$ of A with parameters U instantiated to $\boldsymbol{\theta}_U$ is then defined as the expectation over the performance of the different parameter settings in $\boldsymbol{\theta}_U$'s extension set:*

$$\begin{aligned} a_U(\boldsymbol{\theta}_U) &= \mathbb{E}[y(\boldsymbol{\theta}_{N|U}) \mid \boldsymbol{\theta}_{N|U} \in X(\boldsymbol{\theta}_U)] \\ &= \frac{1}{||\boldsymbol{\Theta}_T||} \int y(\boldsymbol{\theta}_{N|U}) d\boldsymbol{\theta}_T. \end{aligned}$$

Similarly, the marginal predicted *performance $\hat{a}_U(\boldsymbol{\theta}_U)$ under a model $\hat{y} : \boldsymbol{\Theta} \mapsto \mathbb{R}$ is*

$$\hat{a}_U(\boldsymbol{\theta}_U) = \frac{1}{||\boldsymbol{\Theta}_T||} \int \hat{y}(\boldsymbol{\theta}_{N|U}) d\boldsymbol{\theta}_T. \tag{3}$$

Note that if the performance model $\hat{y}$ has low predictive error on average across the configuration space, the difference between predicted and true marginal performance will also be low.

2.2 Efficient Computation of Marginal Predictions

In this section, we show that when using random forests predictors $\hat{y}$, the marginal predicted performance $\hat{a}_U(\boldsymbol{\theta}_U)$ defined in Eq. 3 can be computed exactly in linear time. This is important, since random forests have been shown to yield state-of-the-art performance predictions for a wide range of instance distributions from various computational problems, and for several highly parameterized algorithms [Hutter *et al.*, 2012].

Our approach works in two phases: a preprocessing phase that has to be carried out only once and a prediction phase that has to be carried out once for each requested marginal prediction. Both phases require only linear time given a random forest model as input (constructing the random forest is a separate problem, but is also cheap: quadratic in the number of data data points N in the worst case, and proportional to $N \log N$ in the more realistic best case [Hutter *et al.*, 2012]).

The key idea behind our algorithm is to exploit the fact that each of the regression trees in the random forest defines a partitioning $\mathcal{P}$ of the configuration space $\mathbf{\Theta}$, with piecewise constant predictions c_i in each $P_i \in \mathcal{P}$ (see Eq. 1 and 2), and that the problem of computing sums over an arbitrary number of configurations thus reduces to the problem of identifying how many of the configurations fall into which partition. The random forest prediction then simply averages the individual tree predictions.

We first show that, given a partitioning $\mathcal{P}$, we can compute marginal predictions as a linear sum over entries in the leaves.

Theorem 6. *Given the partitioning $\mathcal{P}$ of a regression tree $\mathcal{T}$ that defines a predictor $\hat{y} : \mathbf{\Theta} \mapsto \mathbb{R}$, and any partial instantiation $\boldsymbol{\theta}_U$ of $\mathbf{\Theta}$'s parameters N, the marginal predicted performance $\hat{a}_U(\boldsymbol{\theta}_U)$ can be computed as*

$$\hat{a}_U(\boldsymbol{\theta}_U) = \sum_{P_i \in \mathcal{P}} \frac{||\mathbf{\Theta}^{(i)}_{N \setminus U}||}{||\mathbf{\Theta}_{N \setminus U}||} \, \mathbb{I}(\boldsymbol{\theta}_U \in \mathbf{\Theta}^{(i)}_U) \cdot c_i.$$

We can compute this equation by a simple iteration over the partitions, looping over the variables in U to check whether $\mathbf{\Theta}_U$ is consistent with the current partition. However, regression trees do not normally provide explicit access to these partitions; they are defined implicitly through the leaves. Since we need to represent the partitioning explicitly, one may worry about space complexity. Indeed, if we stored the values $\mathbf{\Theta}^{(i)}_j$ for each leaf i and categorical parameter j (and lower and upper bounds for continuous parameters), for a random forest with B trees of L leaves each, and n categorical parameters with maximal categorical domain size D, we would end up with space complexity of $\Theta(B \cdot L \cdot n \cdot D)$. In the largest of the practical scenarios we consider later in this work (random forests with $B = 10$ trees of up to $L = 100\,000$ leaves, configuration spaces with up to $n = 76$ parameters and domain sizes up to $D = 20$) this would have been infeasible. Instead, we show that Algorithm 1 can compute the partitioning using a pointer-based data structure, reducing the space complexity to $O(B \cdot L \cdot (D + n))$.[1]

Theorem 7. *Given a regression tree $\mathcal{T}$ with L leaves and a configuration space $\mathbf{\Theta}$ with n parameters of maximal categorical domain size D, Algorithm 1 computes $\mathcal{T}$'s partitioning of $\mathbf{\Theta}$ in time and space $O(L \cdot D + L \cdot n)$. We can use that partitioning to compute arbitrary marginal predictions using additional space $O(1)$ and time $O(L \cdot n \log D)$.*

Corollary 8. *Given a random forest F with B trees of up to L leaves that defines a predictor $\hat{y} : \mathbf{\Theta} \mapsto \mathbb{R}$ for a configuration space with n parameters and maximal categorical domain size D, the time and space complexity of computing a single marginal prediction of F is $O(B \cdot L \cdot \max\{D + n, n \log D\})$. Additional marginal predictions cost additional space $O(1)$ and time $O(B \cdot L \cdot n \log D)$.*

[1]Alternatively, if space were *not* a concern, the partitioning could be represented using a simple bit mask, replacing $O(\log(D))$ member queries with $O(1)$ operations and thus reducing its complexity (for single trees) from $O(L \cdot n \log D)$ to $O(L \cdot n)$.

Algorithm 1: ComputePartitioning($\mathbf{\Theta}$, $\mathcal{T}$, i, $\mathbf{\Theta}^{(i)}$)

```
Input  : $\mathbf{\Theta} = \Theta_1 \times \cdots \times \Theta_n$, a parameter configuration space;
         $\mathcal{T}$, a regression tree partitioning $\mathbf{\Theta}$; $i$, a node;
         $\mathbf{\Theta}^{(i)} = \Theta^{(i)}_1 \times \cdots \times \Theta^{(i)}_n$, $i$'s partition of $\mathbf{\Theta}$
Output : Partitioning $\mathcal{P} = \{P_1, ..., P_k\}$ of $\mathbf{\Theta}^{(i)}$
1 if If node $i$ is a leaf then return $\{\mathbf{\Theta}^{(i)}\}$
2 else
3     Let $v$ be the parameter that node $i$ splits on
4     Follow the splitting rule defined by node $i$ to partition $\Theta^{(i)}_v$
      into newly created sets $\Theta^{(l)}_v$ and $\Theta^{(r)}_v$ for its left and right
      child $l$ and $r$, respectively
5     $\mathcal{P}_l \leftarrow$ ComputePartitioning($\mathbf{\Theta}$, $\mathcal{T}$, $l$,
      $\Theta^{(i)}_1 \times \cdots \Theta^{(i)}_{v-1} \times \Theta^{(l)}_v \times \Theta^{(i)}_{v-1} \times \cdots \times \Theta^{(i)}_n$)
6     $\mathcal{P}_r \leftarrow$ ComputePartitioning($\mathbf{\Theta}$, $\mathcal{T}$, $r$,
      $\Theta^{(i)}_1 \times \cdots \Theta^{(i)}_{v-1} \times \Theta^{(r)}_v \times \Theta^{(i)}_{v-1} \times \cdots \times \Theta^{(i)}_n$)
7     return $\mathcal{P}_l \cup \mathcal{P}_r$
```

3 Efficient Decomposition of Variance

In this section, we review functional analysis of variance and demonstrate how we can use our efficient marginal predictions with this technique to quantify the importance of each algorithm parameter, as well as the importance of low-order parameter interactions.

3.1 Functional Analysis of Variance (ANOVA)

The machinery we use is that of functional analysis of variance (functional ANOVA), a data analysis method from the statistical literature [Stone, 1994; Huang, 1998; Hooker, 2007]. While this method has not received much attention in the AI community so far, we believe that it offers great value. In a nutshell, ANOVA partitions the observed variation of a response value (here: algorithm performance) into components due to each of its inputs (here: algorithm parameters). *Functional* ANOVA decomposes a function $\hat{y} : \Theta_1 \times \cdots \times \Theta_n \mapsto \mathbb{R}$ into additive components that only depend on subsets of its parameters N:

$$\hat{y}(\boldsymbol{\theta}) = \sum_{U \subseteq N} f_U(\boldsymbol{\theta}_U).$$

The components $f_U(\boldsymbol{\theta}_U)$ are defined as follows:

$$f_U(\boldsymbol{\theta}_U) = \begin{cases} \frac{1}{||\mathbf{\Theta}||} \int \hat{y}(\boldsymbol{\theta}) d\boldsymbol{\theta} & \text{if } U = \emptyset. \\ a_U(\boldsymbol{\theta}_U) - \sum_{W \subsetneq U} f_W(\boldsymbol{\theta}_W) & \text{otherwise.} \end{cases} \quad (4)$$

The constant $f_\emptyset$ is the predictor's mean across its domain. The unary functions $f_{\{j\}}(\boldsymbol{\theta}_j)$ are called *main effects* and capture the effect of varying parameter j, averaging across all instantiations of all other parameters. The functions $f_U(\boldsymbol{\theta}_U)$ for $|U| > 1$ capture exactly the interaction effects between all variables in U (excluding all lower-order main and interaction effects of $W \subsetneq U$).

The variance of $\hat{y}$ across its domain $\mathbf{\Theta}$ is

$$\mathbb{V} \;=\; \frac{1}{||\mathbf{\Theta}||} \int (\hat{y}(\boldsymbol{\theta}) - f_\emptyset)^2 d\boldsymbol{\theta}, \quad (5)$$

and functional ANOVA decomposes this variance into contributions by all subsets of variables (see, e.g., Hooker [2007]

Algorithm 2: QuantifyParameterImportance($\mathbf{\Theta}$, $\mathcal{T}$, K)

```
Input  : $\mathbf{\Theta}$, a configuration space with parameters $N$; $\mathcal{T}$, a regression tree; $K$, the maximal order of interaction effects to be computed
Output : $\{\mathbb{F}_U \mid U \subseteq N, |U| \leq K\}$, the fraction of variance contributed by all parameter subsets up to size $K$
1  $\mathcal{P} \leftarrow$ ComputePartitioning($\mathbf{\Theta}$, $\mathcal{T}$, 1, $\mathbf{\Theta}$)   // 1 is $\mathcal{T}$'s root's index
2  $f_\emptyset \leftarrow \sum_{P_i \in \mathcal{P}} \left(\prod_{j \in N} ||\mathbf{\Theta}_j^{(i)}||/||\mathbf{\Theta}_j||\right) \cdot c_i$.
3  $\mathbb{V} \leftarrow \sum_{P_i \in \mathcal{P}} \left(\prod_{j \in N} ||\mathbf{\Theta}_j^{(i)}||/||\mathbf{\Theta}_j||\right) \cdot (c_i - f_\emptyset)^2$.
4  for $k = 1, \ldots, K$ do
5      for all $U \in \{U' \subset N, |U'| = k\}$ do
6          $\mathbb{V}_U \leftarrow 0$
7          for all $\boldsymbol{\theta}_U \in \mathbf{\Theta}_U$ do
8              $a_U(\boldsymbol{\theta}_U) \leftarrow \sum_{P_i \in \mathcal{P}} \frac{||\mathbf{\Theta}_{N \setminus U}^{(i)}||}{||\mathbf{\Theta}_{N \setminus U}||} \mathbb{I}(\boldsymbol{\theta}_U \in \mathbf{\Theta}_U^{(i)}) \cdot c_i$
9              $f_U(\boldsymbol{\theta}_U) \leftarrow a_U(\boldsymbol{\theta}_U) - \sum_{W \subsetneq U} f_W(\boldsymbol{\theta}_W)$
10             $\mathbb{V}_U \leftarrow \mathbb{V}_U + 1/||\mathbf{\Theta}_U|| \cdot f_U(\boldsymbol{\theta}_U)^2$
11         $\mathbb{F}_U \leftarrow \mathbb{V}_U / \mathbb{V}$
12 return $\{\mathbb{F}_U \mid U \subseteq N, |U| \leq K\}$
```

for a derivation):

$$\mathbb{V} \quad = \quad \sum_{U \subset N} \frac{1}{||\mathbf{\Theta}_U||} \int f_U(\boldsymbol{\theta}_U)^2 d\boldsymbol{\theta}_U. \qquad (6)$$

The importance of all main and interaction effects f_U can thus be quantified by the fraction of variance they explain:

$$\mathbb{F}_U \quad = \quad \frac{1}{\mathbb{V}} \left(\frac{1}{||\mathbf{\Theta}_U||} \int f_U(\boldsymbol{\theta}_U)^2 d\boldsymbol{\theta}_U \right). \qquad (7)$$

3.2 Variance Decomposition in Random Forests

In Algorithm 2, we use our efficient marginal predictions from Section 2.2 to quantify the importance of main and interaction effects in the functional ANOVA framework of Eq. 7.

Theorem 9. *Given a configuration space $\mathbf{\Theta}$ consisting of n categorical*[2] *parameters of maximal domain size D and a regression tree $\mathcal{T}$ with L leaves that defines a predictor $\hat{y} : \mathbf{\Theta} \mapsto \mathbb{R}$, Algorithm 2 exactly computes the fractions of variance explained by all subsets U of $\mathbf{\Theta}$'s parameters N of arity up to K, with space complexity $O(L \cdot D + L \cdot n)$ and time complexity $O\left(L \cdot D + \sum_{k=1}^{K} \binom{n}{k} \cdot D^k (L \cdot n \log d + 2^k)\right)$.*

To compute parameter importance in random forests, we simply apply Algorithm 2 for each tree, and compute means and standard deviations across the results. The theoretical time complexity shown above suggests that we should be able to compute all main interaction effects and low-order interaction effects in reasonable time. As we will demonstrate in our experiments, in most domains we can indeed compute all main effects in seconds and all pairwise interaction effects in a few minutes; for a prominent algorithm with only 4 parameters we can compute *all* effects in less than a second.

[2]For continuous parameters j with $\mathbf{\Theta}_j = [l_j, u_j]$, we have to sum over all intervals of $[l_j, u_j]$ defined by the split points in $\bigcup_{P_i \in \mathcal{P}} \{\min \mathbf{\Theta}_j^{(i)}, \max \mathbf{\Theta}_j^{(i)}\}$. The number of such intervals can in principle grow as large as the number of leaves, leading to an increased worst-case time complexity $O\left(L \cdot D + \sum_{k=1}^{K} \binom{n}{k} \cdot L^k (L \cdot n \log d + 2^k)\right)$.

Problem	Algorithm	# Params	Total #configs	Reference
MIP	CPLEX	76	1.90×10^{47}	[IBM Corp., 2012]
ASP	CLASP	85	1.42×10^{49}	[Gebser *et al.*, 2007]
TSP	LK-H	23	8.06×10^{14}	[Helsgaun, 2009]
SAT	SPEAR	26	8.34×10^{17}	[Babić and Hutter, 2007]
SAT	CRYPTOMINISAT	36	5.0×10^{13}	[Soos, 2010]
SAT	SPARROW	4	2 800	[Balint *et al.*, 2011]
SAT	CAPTAINJACK	33	4.98×10^{30}	[Tompkins *et al.*, 2011]
SAT	SATENSTEIN	51	1.38×10^{34}	[KhudaBukhsh *et al.*, 2009]

Table 1: Algorithms and their parameter configuration spaces.

Problem	Benchmark	Application	Reference
MIP	RCW	Wildlife conservation	[Ahmadizadeh *et al.*, 2010]
MIP	CORLAT	Wildlife conservation	[Gomes *et al.*, 2008]
MIP	Regions200	Combinatorial auctions	[Leyton-Brown *et al.*, 2000]
MIP	CLS	Capacitated lot-sizing	[Atamtürk and Muñoz, 2004]
ASP	WeightSeq	Database query optimization	[Silverthorn *et al.*, 2012]
ASP	Riposte	Software Verification	[Silverthorn *et al.*, 2012]
SAT	BMC	Bounded model checking	[Zarpas, 2005]
SAT	SWV	Software verification	[Babić and Hu, 2007]
SAT	3SAT1k	Unif. random 3-SAT	[Tompkins *et al.*, 2011]
SAT	5SAT500	Unif. random 5-SAT	[Tompkins *et al.*, 2011]
TSP	RUE	Random uniform TSP	[Johnson, 2011]
TSP	RCE	Random clustered TSP	[Johnson, 2011]

Table 2: Instance distributions.

4 Empirical Evaluation

We describe an empirical study in which we applied our methods for assessing parameter importance to state-of-the-art solvers for propositional satisfiability (SAT), mixed integer programming (MIP), answer set programming (ASP) and the travelling salesman problem (TSP). Tables 1 and 2 summarize the solvers and benchmark instance distributions used in our experiments; in all cases, we chose solver parameterizations found in previous work on algorithm configuration. We ran CPLEX 12.1 on all four MIP benchmarks, CLASP 2.1 on both ASP benchmarks, LK-H 2.0 on both TSP benchmarks, the industrial SAT solvers SPEAR 1.2.1 and CRYPTOMINISAT 2.9.5 on BMC and SWV, and the stochastic local search solvers SPARROW, CAPTAINJACK, and SATENSTEIN on the (satisfiable) k-SAT benchmarks.

To gather training data for a performance model, in each case, we ran the solver with 10 000 uniformly sampled parameter configurations on one randomly sampled instance each. We merged this data with the data gathered by executing (at least) 25 runs of the algorithm configuration procedure SMAC [Hutter *et al.*, 2011] and constructed random forest predictors $\hat{y}' : \mathbf{\Theta} \times \mathcal{I} \mapsto \mathbb{R}$ on the resulting data sets that predict the performance for combinations of parameter configurations $\boldsymbol{\theta} \in \mathbf{\Theta}$ and instances $i \in \mathcal{I}$. Note that our analysis procedures are only defined for random forests $\hat{y} : \mathbf{\Theta} \mapsto \mathbb{R}$ that predict the performance of parameter configurations, aggregated across instances. To construct such forests, we first used $\hat{y}'$ to predict the marginal performance $m_{\boldsymbol{\theta}} = \mathbb{E}_{i \in \mathcal{I}}[\hat{y}'(\boldsymbol{\theta}, i)]$ of the training configurations across instances. Then, we built random forest predictors $\hat{y} : \mathbf{\Theta} \mapsto \mathbb{R}$ using tuples $(\boldsymbol{\theta}, m_{\boldsymbol{\theta}})$ as training data. In preliminary experiments (omitted due to lack of space), we confirmed that these models predicted well

Scenario	Raw Performance		Improvement over default	
	Main	Pairwise	Main	Pairwise
CPLEX-RCW	70% (0.7s)	13% (110s)	5% (0.3s)	12% (32s)
CPLEX-CORLAT	35% (6.3s)	8% (968s)	45% (4.6s)	20% (709s)
CPLEX-Regions200	47% (15s)	29% (2 230s)	12% (5.7s)	21% (725s)
CPLEX-CLS	68% (45s)	11% (6 119s)	3% (12s)	10% (1 495s)
CLASP-WeightedSeq	63% (5.7s)	12% (990s)	45% (4.5s)	18% (714s)
CLASP-Riposte	79% (26s)	14% (4 205s)	26% (21.8s)	5% (3 764)
SPEAR-BMC	93% (0.3s)	1% (20s)	34% (0.1s)	13% (7.0s)
SPEAR-SWV	87% (1.6s)	5% (92s)	83% (1.2s)	8% (73s)
CRYPTOMINISAT-BMC	36% (0.4s)	21% (14s)	4% (0.04s)	10% (1.0s)
CRYPTOMINISAT-SWV	32% (1.0s)	29% (59s)	23% (0.7s)	34% (28s)
SPARROW-3SAT1k	84% (0.03s)	11% (0.08s)	42% (0.008s)	36% (0.1s)
SPARROW-5SAT500	73% (0.03s)	22% (0.05s)	73% (0.04s)	22% (0.06s)
CAPTAINJACK-3SAT1k	61% (2.0s)	13% (286s)	63% (1.5s)	10% (208s)
CAPTAINJACK-5SAT500	48% (1.3s)	13% (184s)	49% (1.2s)	13% (152s)
SATENSTEIN-3SAT1k	53% (2.1s)	39% (203s)	36% (0.9s)	36% (112s)
SATENSTEIN-5SAT500	55% (2.7s)	34% (227s)	51% (1.5s)	36% (175s)
LKH-RUE	83% (0.76s)	10% (29s)	3% (0.06s)	6% (1.8s)
LKH-RCE	77% (1.1s)	12% (35s)	22% (0.3s)	24% (9.8s)

Table 3: Fractions of variance explained by main effects and pairwise interaction effects, and time required to compute them. Left: effects for explaining raw performance; Right: effects for explaining improvements over the default.

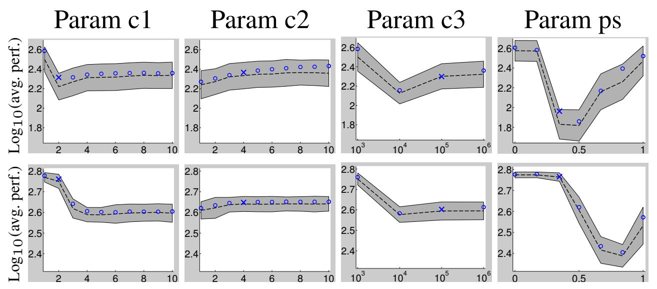


Figure 1: Main effects of Sparrow's 4 parameters on 3SAT-1k (top) and 5SAT-500 (bottom). Each plot shows the effect of one parameter (identified by the column header); the y-axis in each plot shows $\log_{10}$ of marginal performance (across all instantiations of all other parameters) when varying that single parameter across the x-axis. Blue circles and crosses denote true marginals, with the cross also denoting the default value of each parameter. The black line and grey-shaded area indicate predicted means $\pm$ one predictive standard deviation.

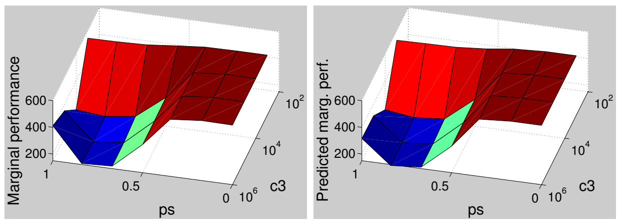


Figure 2: Most important interaction effect between 2 Sparrow parameters on 5SAT500. Left: true marginal performance; right: predicted marginal performance.

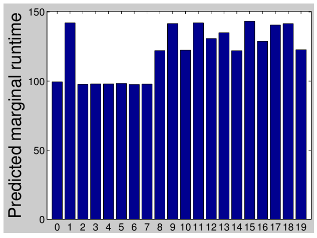
across the configuration space, and particularly well in good parts of the space, as identified by configuration runs using both SMAC and ParamILS.

Table 3 (left side) quantifies main and interaction effects for all combinations of algorithm and instance distribution we studied. We note that in all cases, the main effects accounted for a substantial fraction of the overall performance variation, between 32% and 93%. Since single parameter effects are easier to understand for human algorithm designers or analysts than complex interaction effects, this is an encouraging finding. These main effects could also usually be computed within seconds, allowing algorithm designers to use our approach interactively. Pairwise interaction effects were important in several cases, explaining up to 39% of the performance variation. Their computational cost depended on the number of algorithm parameters and the size of the regression trees (which grows linearly in the number of training data points); it ranged from 0.05 seconds to 102 minutes.

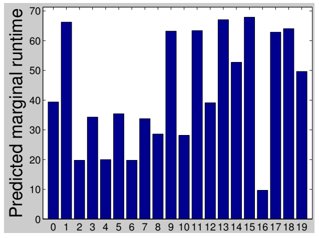
To confirm that our model predictions indeed capture the performance variation over a given configuration space, we gathered performance data for all 2 800 parameter configurations of the Sparrow solver, allowing us to compute actual (as opposed to predicted) marginals across its entire configuration space, as well as arbitrary interaction effects. Obviously, this type of analysis would be infeasible for most larger configuration spaces; on the other hand, as we will see later, the predictions we obtain based on our models *do* scale to larger configuration spaces. For each of two instance distributions, Figure 1 shows true and predicted main effects of Sparrow's 4 parameters, that is, the performance of all possible values for one single parameter, marginalized across all values of all other parameters. We note that our marginal predictions agree well with the true marginals. Figure 2 shows true and predicted marginals for the most important interaction effect on one distribution (explaining 9.7% of the total variance). As the plot shows, large values of the smoothing probability ps and small values of the inverse multiplier for variable age $c3$ are generally preferable, but setting both parameters to their respective limits yields poor performance. Again, our model's mean predictions closely resemble the true marginals.

One particularly interesting case in Table 3 is SPEAR's performance on bounded model checking (BMC) instances. Here, 92.7% of the variance was explained by a single parameter, SPEAR's variable selection heuristic. Figure 3(a) shows that several standard activity heuristics performed well for this dataset. In contrast, for SPEAR's performance on software verification (SWV) instances (see Figure 3(b)), we noticed that one of the heuristics that was thought to perform poorly in fact performed very well. Before seeing these results, SPEAR's developer did not have any intuition about which variable selection heuristic would work well for SWV; in particular, he was not sure whether selecting variables in the order they were created (option 16, clearly the best choice) or in reverse order (option 17, one of the worst choices) would be preferable (personal communication).

Considering once more the SPEAR-BMC scenario from Figure 3(a), we note that much of the predictor's variance was caused by choices for the variable selection heuristic that perform much worse than the default (option 0). The notion of parameter importance as the parameter's contribution to performance variations across the whole space will inherently be influenced strongly by such poorly performing values. In practice, one might be more interested in good settings of a parameter than in bad ones. We can formalize this notion by asking only about variation across parts of the configuration



(a) SPEAR-BMC



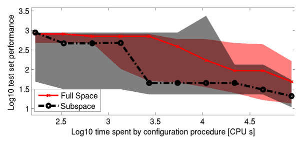
(b) SPEAR-SWV

Figure 3: Main effect of SPEAR's variable selection heuristic for two different instance distributions. The figure shows predicted marginal performance for each of SPEAR's 20 options for variable selection. The first 8 options are activity heuristics (heuristic 1 prefers *less* active variables, which performs poorly).

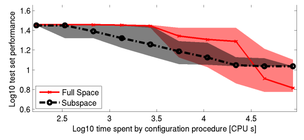
space in which performance is better than the default. Our analysis methods can be adapted to support this notion of importance by adapting the random forest they operate on, by changing its input data: we simply cap the performance values of each training configuration at the performance of the default configuration. The consequence is that no configuration will be predicted to be worse than the default, and thus all remaining performance variation will indicate regions of the space with performance better than the default. By assessing which parameters and interactions cause this variation, we directly capture parameters that are interesting to change from their defaults.

To study this alternative notion of importance, we again show main and interaction effects for each combination of algorithm and instance distribution, now for quantifying variation in the performance metric "improvement over the default" (see the right columns in Table 3). The differences to main and interaction effects for explaining variation in *raw* performance (left columns in the same table) are marked; in most cases, single parameters explain improvements over the default to a much smaller degree than they explain overall performance variation. An extreme example is CPLEX-RCW: while 70% of the variance in raw performance was explained by main effects, this was the case for only 5.2% of the variance in improvements over the default. Investigating this in more detail, we found that most of the former variance (58.7%) was explained by the single parameter `preprocessing_reduce` (which determines which kind of reductions CPLEX performs during preprocessing): similar to the SPEAR-BMC example above, this parameter has two settings yielding extremely poor performance. However, for explaining improvements over the default this parameter is useless, since none of its values achieves such improvements.

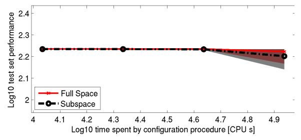
Based on this insight, we can define *reduced parameter configuration spaces* that capture most of the potential of improving over the default. To verify that by doing so we indeed capture the potential for improvement, for each of the four MIP instance distributions we defined a CPLEX subspace with the 10 parameters with largest main effects for explaining the variance in improvements over the default. We then ran the automated configuration procedures ParamILS and SMAC both on the original CPLEX space (with 76 parameters) and on the reduced space to assess the true importance of the selected parameters. As shown in Figure 4, both ParamILS and SMAC



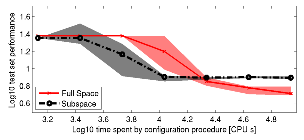
(a) ParamILS on CPLEX-CORLAT



(b) ParamILS on CPLEX-Regions200



(c) SMAC on CPLEX-RCW



(d) SMAC on CPLEX-CLS

Figure 4: Algorithm configuration results on full and reduced parameter configuration spaces. SMAC and ParamILS made faster progress initially when configuring on the smaller important subspace, but when given more time, also optimizing less important parameters yielded further progress.

found good configurations in the reduced spaces and made progress faster than in the original CPLEX space. However, while the selected parameters captured a substantial part of the performance variation, additionally considering the remaining parameters allowed the configuration process in the full space to produce better results given sufficient time.

5 Conclusion

In this work, we introduced an efficient approach for assessing the importance of algorithm parameters. We first derived a novel linear-time algorithm for computing marginal predictions over arbitrary input dimensions in random forests and then showed how this algorithm can be used to quantify the importance of main effects and interaction effects through a functional ANOVA framework. We empirically validated our approach using performance data of several state-of-the-art solvers for SAT, MIP, ASP, and TSP. We showed that our model-based approach accurately predicted the performance variation over parameter settings by checking against ground truth data for the entire configuration space of a solver with only four parameters. We then demonstrated that the performance variability of seven highly parametric solvers on 12 instance distributions is caused to a large degree by a small number of parameters, and that pairwise interactions between parameters can also play an important role. Finally, we illustrated how our approach can be used to gain insights into a solver's most important parameters and verified the true importance of the selected parameters using algorithm configuration experiments on the subspace induced by these parameters for the highly parametric and widely used CPLEX solver.

We believe that the methods introduced in this work offer a principled, scientific way for algorithm designers and users to gain deeper insights into the way in which design choices controlled by parameters affect the overall performance of a solver. In future work, we plan to extend our approach to detect dominated parameter values and interactions between instance characteristics and parameter settings. We also plan to exploit the insight that in many cases a relatively small number of parameters is responsible for the major part of the performance variation exhibited by highly parametric solvers in the development of new algorithm configuration methods.

References

[Ahmadizadeh *et al.*, 2010] K. Ahmadizadeh, B. Dilkina, C.P. Gomes, and A. Sabharwal. An empirical study of optimization for maximizing diffusion in networks. In *Proc. of CP-10*, pages 514–521, 2010.

[Ansotegui *et al.*, 2009] C. Ansotegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of solvers. In *Proc. of CP-09*, pages 142–157, 2009.

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

[Babić and Hu, 2007] D. Babić and A. J. Hu. Structural abstraction of software verification conditions. pages 366–378, 2007.

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

[Balint *et al.*, 2011] A. Balint, A. Fröhlich, D. A. D. Tompkins, and H. H. Hoos. Sparrow2011. Solver description, SAT competition 2011, 2011.

[Bartz-Beielstein, 2006] T. Bartz-Beielstein. *Experimental research in evolutionary computation: the new experimentalism*. Natural Computing Series. Springer Verlag, 2006.

[Birattari *et al.*, 2002] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proc. of GECCO-02*, pages 11–18, 2002.

[Breiman, 2001] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Brewer, 1995] E. A. Brewer. High-level optimization via automated statistical modeling. In *Proc. of PPOPP-95*, pages 80–91, 1995.

[Chiarandini and Goegebeur, 2010] M. Chiarandini and Y. Goegebeur. Mixed models for the analysis of optimization algorithms. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 225–264. Springer Verlag, 2010.

[Gagliolo and Legrand, 2010] M. Gagliolo and C. Legrand. Algorithm survival analysis. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 161–184. Springer Verlag, 2010.

[Gebser *et al.*, 2007] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proc. of IJCAI-07*, pages 386–392, 2007.

[Gomes *et al.*, 2008] C.P. Gomes, W. van Hoeve, and A. Sabharwal. Connections in networks: a hybrid approach. In *Proc. of CPAIOR-08*, pages 303–307, 2008.

[Helsgaun, 2009] K. Helsgaun. General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119–163, 2009.

[Hooker, 2007] G. Hooker. Generalized functional ANOVA diagnostics for high dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3), 2007.

[Huang, 1998] J. Z. Huang. Projection estimation in multiple regression with application to functional anova models. *The Annals of Statistics*, 26(1):242–272, 1998.

[Hutter *et al.*, 2009] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *JAIR*, 36:267–306, October 2009.

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

[Hutter *et al.*, 2012] F. Hutter, L. Xu, H.H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: The state of the art. *CoRR*, abs/1211.0906, 2012.

[Hutter *et al.*, 2013] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In *Proc. of LION-7*, 2013.

[IBM Corp., 2012] IBM Corp. IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, 2012. Last accessed on October 27, 2012.

[Johnson, 2011] D. S. Johnson. Random TSP generators for the DIMACS TSP challenge. http://www2.research.att.com/~dsj/chtsp/codes.tar, 2011. Last accessed on May 16, 2011.

[KhudaBukhsh *et al.*, 2009] A. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown. SATenstein: Automatically building local search SAT solvers from components. In *Proc. of IJCAI-09*, pages 517–524, 2009.

[Leyton-Brown *et al.*, 2000] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proc. of EC-00*, pages 66–76, 2000.

[Nannen and Eiben, 2007] V. Nannen and A.E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proc. of IJCAI-07*, pages 975–980, 2007.

[Ridge and Kudenko, 2006] E. Ridge and D. Kudenko. Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In *Proc. of PPSN-06*, pages 27–34, 2006.

[Silverthorn *et al.*, 2012] B. Silverthorn, Y. Lierler, and M. Schneider. Surviving solver sensitivity: An ASP practitioner's guide. In *Proc. of ICLP-LIPICS-12*, pages 164–175, 2012.

[Smith-Miles and Lopes, 2012] K. Smith-Miles and L. Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers and Operations Research*, 39(5):875–889, 2012.

[Soos, 2010] M. Soos. CryptoMiniSat 2.5.0. Solver description, SAT Race 2010, 2010.

[Stone, 1994] C. J. Stone. The use of polynomial splines and their tensor products in multivariate function estimation. *The Annals of Statistics*, 22(1):118–171, 1994.

[Tompkins *et al.*, 2011] D. A. D. Tompkins, A. Balint, and H. H. Hoos. Captain jack: new variable selection heuristics in local search for sat. In *Proc. of SAT-11*, pages 302–316, 2011.

[Xu *et al.*, 2008] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *JAIR*, 32:565–606, June 2008.

[Zarpas, 2005] E. Zarpas. Benchmarking SAT solvers for bounded model checking. In *Proc. of SAT-05*, pages 340–354. Springer Verlag, 2005.