Identifying Key Algorithm Parameters and Instance Features Using Forward Selection

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown(✉)

University of British Columbia, 2366 Main Mall, Vancouver BC, V6T 1Z4, Canada
{hutter,hoos,kevinlb}@cs.ubc.ca

Abstract. Most state-of-the-art algorithms for large-scale optimization problems expose free parameters, giving rise to combinatorial spaces of possible configurations. Typically, these spaces are hard for humans to understand. In this work, we study a model-based approach for identifying a small set of both algorithm parameters and instance features that suffices for predicting empirical algorithm performance well. Our empirical analyses on a wide variety of hard combinatorial problem benchmarks (spanning SAT, MIP, and TSP) show that—for parameter configurations sampled uniformly at random—very good performance predictions can typically be obtained based on just two key parameters, and that similarly, few instance features and algorithm parameters suffice to predict the most salient algorithm performance characteristics in the combined configuration/feature space. We also use these models to identify settings of these key parameters that are predicted to achieve the best overall performance, both on average across instances and in an instance-specific way. This serves as a further way of evaluating model quality and also provides a tool for further understanding the parameter space. We provide software for carrying out this analysis on arbitrary problem domains and hope that it will help algorithm developers gain insights into the key parameters of their algorithms, the key features of their instances, and their interactions.

1 Introduction

State-of-the-art algorithms for hard combinatorial optimization problems tend to expose a set of parameters to users to allow customization for peak performance in different application domains. As these parameters can be instantiated independently, they give rise to combinatorial spaces of possible parameter configurations that are hard for humans to handle, both in terms of finding good configurations and in terms of understanding the impact of each parameter. As an example, consider the most widely used mixed integer programming (MIP) software, IBM ILOG CPLEX, and the manual effort involved in exploring its 76 optimization parameters [1].

By now, substantial progress has been made in addressing the first sense in which large parameter spaces are hard for users to deal with. Specifically, it has been convincingly demonstrated that methods for *automated algorithm*

DOI: 10.1007/978-3-642-44973-4_40, © Springer-Verlag Berlin Heidelberg 2013

configuration [2–7] are able to find configurations that substantially improve the state of the art for various hard combinatorial problems (e.g., SAT-based formal verification [8], mixed integer programming [1], timetabling [9], and AI planning [10]). However, much less work has been done towards the goal of explaining to algorithm designers which parameters are important and what values for these important parameters lead to good performance. Notable exceptions in the literature include experimental design based on linear models [11,12], an entropy-based measure [2], and visualization methods for interactive parameter exploration, such as contour plots [13]. However, to the best of our knowledge, none of these methods has so far been applied to study the configuration spaces of state-of-the-art highly parametric solvers; their applicability is unclear, due to the high dimensionality of these spaces and the prominence of discrete parameters (which, e.g., linear models cannot handle gracefully).

In the following, we show how a generic, model-independent method can be used to:

- identify key parameters of highly parametric algorithms for solving SAT, MIP, and TSP;
- identify key instance features of the underlying problem instances;
- demonstrate interaction effects between the two; and
- identify values of these parameters that are predicted to yield good performance, both unconditionally and conditioned on instance features.

Specifically, we gather performance data by randomly sampling both parameter settings and problem instances for a given algorithm. We then perform *forward selection*, iteratively fitting regression models with access to increasing numbers of parameters and features, in order to identify parameters and instance features that suffice to achieve predictive performance comparable to that of a model fit on the full set of parameters and instance features. Our experiments show that these sets of sufficient parameters and/or instance features are typically very small—often containing only two elements—even when the candidate sets of parameters and features are very large. To understand what values these key parameters should take, we find performance-optimizing settings given our models, both unconditionally and conditioning on our small sets of instance features. We demonstrate that parameter configurations that set as few as two key parameters based on the model (and all other parameters at random) often substantially outperform entirely random configurations (sometimes by up to orders of magnitude), serving as further validation for the importance of these parameters. Our qualitative results still hold for models fit on training datasets containing as few as 1 000 data points, facilitating the use of our approach in practice. We conclude that our approach can be used out-of-the-box by algorithm designers wanting to understand key parameters, instance features, and their interactions. To facilitate this, our software (and data) is available at http://www.cs.ubc.ca/labs/beta/Projects/EPMs.

2 Methods

Ultimately, our forward selection methods aim to identify a set of the k_{max} most important algorithm parameters and m_{max} most important instance features (where k_{max} and m_{max} are user-defined), as well as the best values for these parameters (both on average across instances and on a per-instance basis). Our approach for solving this problem relies on predictive models, learned from given algorithm performance data for various problem instances and parameter configurations. We identify important parameters and features by analyzing which inputs suffice to achieve high predictive accuracy in the model, and identify good parameter values by optimizing performance based on model predictions.

2.1 Empirical Performance Models

Empirical Performance Models (EPMs) are statistical models that describe the performance of an algorithm as a function of its inputs. In the context of this paper, these inputs comprise both features of the problem instance to be solved and the algorithm's free parameters. We describe a problem instance by a vector of m features $\boldsymbol{z} = [z_1, \ldots, z_m]^\intercal$, drawn from a given *feature space* $\mathcal{F}$. These features must be computable by an automated, domain-specific procedure that efficiently extracts features for any given problem instance (typically, in low-order polynomial time w.r.t. the size of the given problem instance). We describe the *configuration space* of a parameterized algorithm with k parameters $\theta_1, \ldots, \theta_k$ and respective domains $\Theta_1, \ldots, \Theta_k$ by a subset of the cross-product of parameter domains: $\boldsymbol{\Theta} \subseteq \Theta_1 \times \cdots \times \Theta_k$. The elements of $\boldsymbol{\Theta}$ are complete instantiations of the algorithm's k parameters, and we refer to them as *configurations*. Taken together, the configuration and the feature space define the *input space* $\mathcal{I} := \boldsymbol{\Theta} \times \mathcal{F}$.

EPMs for predicting the "empirical hardness" of instances have their origin over a decade ago [14–17] and have been the preferred core reasoning tool of early state-of-the-art methods for the algorithm selection problem (which aim to select the best algorithm for a given problem, dependent on its features [18–20]), in particular of early iterations of the `SATzilla` algorithm selector for SAT [21]. Since then, these predictive models have been extended to model the dependency of performance on (often categorical) algorithm parameters, to make probabilistic predictions, and to work effectively with large amounts of training data [11,12,22,23].

In very recent work, we comprehensively studied EPMs based on a variety of modeling techniques that have been used for performance prediction over the years, including ridge regression [17], neural networks [24], Gaussian processes [22], regression trees [25], and random forests [23]. Overall, we found random forests and approximate Gaussian processes to perform best. Random forests (and also regression trees) were particularly strong for very heterogeneous benchmark sets, since their tree-based mechanism automatically groups similar inputs together and does not allow widely different inputs to interfere with the predictions for a given group. Another benefit of the tree-based methods is apparent from the fact that hundreds of training data points could be shown to suffice to yield competitive performance predictions in joint input spaces induced by as many as 76 algorithm parameters and 138 instance features [23]. This strong performance suggests that the functions being modeled must be relatively simple, for example, by depending at most very weakly on most inputs. In this paper, we ask whether this is the case, and to the extent that this is so, aim to identify the key inputs.

2.2 Forward Selection

There are many possible approaches for identifying important input dimensions of a model. For example, one can measure the model coefficients w in ridge regression (large coefficients mean that small changes in a feature value have a large effect on predictions, see, e.g., [26]) or the length scales λ in Gaussian process regression (small length scales mean that small changes in a feature value have a large effect on predictions, see, e.g., [27]). In random forests, to measure the importance of input dimension i, Breiman suggested perturbing the values in the ith column of the out-of-bag (or validation) data and measuring the resulting loss in predictive accuracy [28].

All of these methods run into trouble when input dimensions are highly correlated. While this does not occur with randomly sampled parameter configurations, it does occur with instance features, which cannot be freely sampled. Our goal is to build models that yield good predictions but yet depend on as few input dimensions as possible; to achieve this goal, it is not sufficient to merely find important parameters, but we need to find a set of important parameters that are as uncorrelated as possible.

Forward selection is a generic, model-independent tool that can be used to solve this problem [17,29].[1] Specifically, this method identifies sets of model inputs that are jointly *sufficient* to achieve good predictive accuracy; our variant of it is defined in Algorithm 1. After initializing the complete input set I and the subset of important inputs S in lines 1–2, the outer **for**-loop incrementally adds one input at a time to S. The **forall**-loop over inputs i not yet contained in S (and not violating the constraint of adding at most k_{max} parameters and m_{max} features) uses validation data to compute err(i), the root mean squared error (RMSE) for a model containing i and the inputs already in S. It then adds the input resulting in lowest RMSE to S. Because inputs are added one at a time, highly correlated inputs will only be added if they provide large *marginal* value to the model.

Note that we simply call procedure *learn* with a subset of input dimensions, regardless of whether they are numerical or categorical (for models that require a so-called “1-in-K encoding” to handle categorical parameters, this means we introduce/drop all K binary columns representing a K-ary categorical input at once). Also note that, while here, we use prediction RMSE on the validation set

[1] A further advantage of forward selection is that it can be used in combination with arbitrary modeling techniques. Although here, we focus on using our best-performing model, random forests, we also provide summary results for other model types.

Algorithm 1: **Algorithm 1: Forward Selection**
In line 10, *learn* refers to an arbitrary regression method that fits a function f to given training data. Note that input dimensions $1, \ldots, k$ are parameters, $k+1, \ldots, k+m$ are features.

```
Input  : Training data $\mathcal{D}_{train} = \langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\rangle$; validation data
         $\mathcal{D}_{valid} = \langle(\mathbf{x}_{n+1}, y_{n+1}), \ldots, (\mathbf{x}_{n+n'}, y_{n+n'})\rangle$; number of parameters, $k$;
         number of features, $m$; desired number $K \leq d = k + m$ of key inputs;
         bound on number of key parameters, $k_{max} \geq 0$; bound on number of
         key features, $m_{max} \geq 0$, such that $k_{max} + m_{max} \geq K$
Output: Subset of $K$ feature indices $S \subseteq \{1, \ldots, d\}$
1  $I \leftarrow \{1, \ldots, d\}$ ;
2  $S \leftarrow \emptyset$ ;
3  for $j = 1, \ldots, K$ do
4      $I_{\text{allowed}} \leftarrow I \setminus S$;
5      if $|S \cap \{1, \ldots, k\}| \geq k_{max}$ then $I_{\text{allowed}} \leftarrow I_{\text{allowed}} \setminus \{1, \ldots, k\}$;
6      if $|S \cap \{k+1, \ldots, k+m\}| \geq m_{max}$ then
       $I_{\text{allowed}} \leftarrow I_{\text{allowed}} \setminus \{k+1, \ldots, k+m\}$;
7      forall $i \in I_{allowed}$ do
8          $S \leftarrow S \cup \{i\}$;
9          forall $(\mathbf{x}_j, y_j) \in \mathcal{D}_{train}$ do $\mathbf{x}_j^S \leftarrow \mathbf{x}_j$ restricted to input dimensions in $S$;
10         $f \leftarrow$ learn$(\langle(\mathbf{x}_1^S, y_1), \ldots, (\mathbf{x}_n^S, y_n)\rangle)$;
11         $\text{err}(i) \leftarrow \sqrt{\sum_{(\mathbf{x}_j, y_j) \in \mathcal{D}_{valid}} (f(\mathbf{x}_j) - y_j)^2}$;
12         $S \leftarrow S \setminus \{i\}$;
13     $\hat{i} \leftarrow$ random element of $\arg\max_i \text{err}(i)$;
14     $S \leftarrow S \cup \{\hat{i}\}$;
15 return $S$;
```

to assess the value of adding input i, forward selection can also be used with any other objective function.[2]

Having selected a set S of inputs via forward selection, we quantify their relative importance following the same process used by Leyton-Brown et al. to determine the importance of instance features [17], which is originally due to [31]: we simply drop one input from S at a time and measure the increase in predictive RMSE. After computing this increase for each feature, we normalize by dividing by the maximal RMSE increase and multiplying by 100.

We note that forward selection can be computationally costly due to its need for repeated model learning: for example, to select 5 out of 200 inputs via forward selection requires the construction and validation of 200 + 199 + 198 + 197 + 196 = 990 models. In our experiments, this process required up to a day of CPU time.

[2] In fact, it also applies to classification algorithms and has, e.g., been used to derive classifiers for predicting the solubility of SAT instances based on 1–2 features [30].

2.3 Selecting Values for Important Parameters

Given a model f that takes k parameters and m instance features as input and predicts a performance value, we identify the best values for the k parameters by optimizing predictive performance according to the model. Specifically, we predict the performance of the partial parameter configuration $\mathbf{x}$ (instantiating k parameter values) on a problem instance with m selected instance features $\mathbf{z}$ as $f([\mathbf{x}^\intercal, \mathbf{z}^\intercal]^\intercal)$. Likewise, we predict its average performance across n instances with selected instance features $\mathbf{z}_1, \dots, \mathbf{z}_n$ as $\sum_{j=1}^{n} \frac{1}{n} \cdot f([\mathbf{x}^\intercal, \mathbf{z}_j^\intercal]^\intercal)$.

3 Algorithm Performance Data

In this section, we discuss the algorithm performance data we used in order to evaluate our approach. We employ data from three different combinatorial problems: propositional satisfiability (SAT), mixed integer programming (MIP), and the traveling salesman problem (TSP). All our code and data is available online: instances and their features (and feature computation code & binaries), parameter specification files and wrappers for the algorithms, as well as the actual runtime data upon which our analysis is based.

3.1 Algorithms and Their Configuration Spaces

We employ peformance data from three algorithms: `CPLEX` for MIP, `SPEAR` for SAT, and `LK-H` for TSP. The parameter configuration spaces of these algorithms are summarized in Table 1.

IBM ILOG `CPLEX` [32] is the most-widely used commercial optimization tool for solving MIPs; it is used by over 1 300 corporations (including a third of the Global 500) and researchers at more than 1 000 universities. We used the same configuration space with 76 parameters as in previous work [1], excluding all `CPLEX` settings that change the problem formulation (e.g., the optimality gap below which a solution is considered optimal). Overall, we consider 12 preprocessing parameters (mostly categorical); 17 MIP strategy parameters (mostly

Table 1. Algorithms and their parameter configuration spaces studied in our experiments.

Algorithm	Parameter type	# parameters of this type	# values considered	Total # configurations
	Boolean	6	2	
`CPLEX`	Categorical	45	3–7	1.90×10^{47}
	Integer	18	5–7	
	Continuous	7	5–8	
	Categorical	10	2–20	
`SPEAR`	Integer	4	5–8	8.34×10^{17}
	Continuous	12	3–6	
	Boolean	5	2	
`LK-H`	Categorical	8	3–10	6.91×10^{14}
	Integer	10	3–9	

categorical); 11 categorical parameters deciding how aggressively to use which types of cuts; 9 real-valued MIP "limit" parameters; 10 simplex parameters (half of them categorical); 6 barrier optimization parameters (mostly categorical); and 11 further parameters. In total, and based on our discretization of continuous parameters, these parameters gave rise to 1.90×10^{47} unique configurations.

SPEAR [33] is a state-of-the-art SAT solver for industrial instances. With appropriate parameter settings, it was shown to be the best available solver for certain types of SAT-encoded hardware and software verification instances [8] (the same IBM and SWV instances we use here). It also won the quantifier-free bit-vector arithmetic category of the 2007 Satisfiability Modulo Theories Competition. We used exactly the same 26-dimensional parameter configuration space as in previous work [8]. SPEAR's categorical parameters mainly control heuristics for variable and value selection, clause sorting, resolution ordering, and also enable or disable optimizations, such as the pure literal rule. Its numerical parameters mainly deal with activity, decay, and elimination of variables and clauses, as well as with the randomized restart interval and percentage of random choices. In total, and based on our discretization of continuous parameters, SPEAR has 8.34×10^{17} different configurations.

LK-H [34] is a state-of-the-art local search solver for TSP based on an efficient implementation of the Lin-Kernighan heuristic. We used the LK-H code from Styles et al. [35], who first reported algorithm configuration experiments with LK-H; in their work, they extended the official LK-H version 2.02 to allow several parameters to scale with instance size and to make use of a simple dynamic restart mechanism to prevent stagnation. The modified version has a total of 23 parameters governing all aspects of the search process, with an emphasis on parameterizing moves. In total, and based on our discretization of continuous parameters, LK-H has 6.91×10^{14} different configurations.

3.2 Benchmark Instances and Their Features

We used the same benchmark distributions and features as in previous work [23] and only describe them on a high level here. For MIP, we used two instance distributions from computational sustainability (RCW and CORLAT), one from winner determination in combinatorial auctions (REG), two unions of these (CR := CORLAT ∪ RCW and CRR := CORLAT ∪ REG ∪ RCW), and a large and diverse set of publicly available MIP instances (BIGMIX). We used 121 features to characterize MIP instances, including features describing problem size, the variable-constraint graph, the constraint matrix, the objective function values, an LP programming relaxation, various probing features extracted from short CPLEX runs and timing features measuring the computational expense required for various groups of features.

For SAT, we used three sets of SAT-encoded formal verification benchmarks: SWV and IBM are sets of software and hardware verification instances, and SWV-IBM is their union. We used 138 features to characterize SAT instances, including features describing problem size, three graph representations, syntactic features, probing features based on systematic solvers (capturing unit propagation and clause learning) and local search solvers, an LP relaxation, survey propagation, and timing features.

For TSP, we used `TSPLIB`, a diverse set of prominent TSP instances, and computed 64 features, including features based on problem size, cost matrix, minimum spanning trees, branch & cut probing, local search probing, ruggedness, and node distribution, as well as timing features.

3.3 Data Acquisition

We gathered a large amount of runtime data for these solvers by executing them with various configurations and instances. Specifically, for each combination of solver and instance distribution (`CPLEX` run on MIP, `SPEAR` on SAT, and `LK-H` on TSP instances), we measured the runtime of each of $M = 1\,000$ randomly-sampled parameter configurations on each of the P problem instances available for the distribution, with P ranging from 63 to 2 000. The resulting runtime observations can be thought of as a $M \times P$ matrix. Since gathering this runtime matrix meant performing $M \cdot P$ (i.e., between 63 000 and 2 000 000) runs per dataset, we limited each single algorithm run to a cutoff time of 300 CPU seconds on one node of the Westgrid cluster Glacier (each of whose nodes is equipped with two 3.06 GHz Intel Xeon 32-bit processors and 2–4 GB RAM). While collecting this data required substantial computational resources (between 1.3 CPU years and 18 CPU years per dataset), we note that this much data was only required for the thorough empirical analysis of our methods; in practice, our methods are often surprisingly accurate based on small amounts of training data. For all our experiments, we partitioned both instances and parameter configurations into training, validation, and test sets; the training sets (and likewise, the validation and test sets) were formed as subsamples of training instances and parameter configurations. We used 10 000 training subsamples throughout our experiments but demonstrate in Sect. 4.3 that qualitatively similar results can also be achieved based on subsamples of 1 000 data points.

We note that sampling parameter configurations uniformly at random is not the only possible way of collecting training data. Uniform sampling has the advantage of producing unbiased training data, which in turn gives rise to models that can be expected to perform well on average across the entire configuration space. However, because algorithm designers typically care more about regions of the configuration space that yield good performance, in future work, we also aim to study models based on data generated through a biased sequential sampling approach (as is implemented, e.g., in model-based algorithm configuration methods, such as SMAC [6]).

4 Experiments

We carried out various computational experiments to identify the quality of models based on small subsets of features and parameters identified using forward selection, to quantify which inputs are most important, and to determine good values for the selected parameters. All our experiments made use of the algorithm performance data described in Sect. 3, and consequently, our claims hold on average across the entire configuration space. Whether they also apply to biased samples from the configuration space (in particular, regions of very strong algorithm performance) is a question for future work.

4.1 Predictive Performance for Small Subsets of Inputs

First, we demonstrate that forward selection identifies sets of inputs yielding low predictive root mean squared error (RMSE), for predictions in the feature space, the parameter space, and their joint space. Figure 1 shows the root mean squared error of models fit with parameter/feature subsets of increasing size. Note in particular the horizontal line, giving the RMSE of a model based on *all* inputs, and that the RMSE of subset models already converges to this performance with few inputs. In the feature space, this has been observed before [17,29] and is intuitive, since the features are typically very correlated, allowing a subset of them to represent the rest. However, the same cannot be said for the parameter

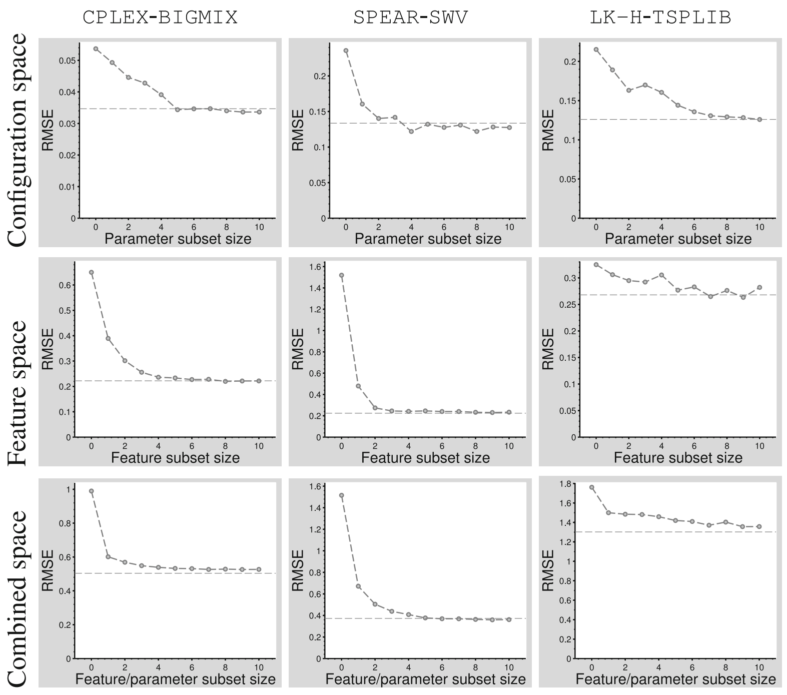


Fig. 1. Predictive quality of random forest models as a function of the number of allowed parameters/features selected by forward selection for 3 example datasets. The inputless prediction (subset size zero) is the mean of all data points. The dashed horizontal line in each plot indicates the final performance of the model using the full set of parameters/features.

space: in our experimental design, parameter values have been sampled uniformly at random and are thus independent (i.e., uncorrelated) by design. Thus, this finding indicates that some parameters influence performance much more than others, to the point where knowledge of a few parameter values suffices to predict performance just as well as knowledge of all parameters.

Figure 2 focuses on what we consider to be the most interesting case, namely performance prediction in the joint space of instance features and parameter configurations. The figure qualitatively indicates the performance that can be achieved based on subsets of inputs of various sizes. We note that in some cases, in particular in the SPEAR scenarios, predictions of models using all inputs closely resemble the true performance, and that the predictions of models based on a few inputs tend to capture the salient characteristics of the full models. Since the instances we study vary widely in hardness, instance features tend to be more predictive than algorithm parameters, and are thus favoured by forward selection. This sometimes leads to models that *only* rely on instance features, yielding predictions that are constant across parameter configurations; for example, see the predictions with up to 10 inputs for dataset CPLEX-CORLAT (the second row in Fig. 2). While these models yield low RMSE, they are uninformative about parameter settings; this observation caused us to modify forward selection as discussed in Sect. 2.2 to limit the number of features/parameters selected.

4.2 Relative Importance of Parameters and Features

As already apparent from Fig. 1, knowing the values of a few parameters is sufficient to predict marginal performance across instances similarly well as when knowing all parameter values. Figure 3 shows *which* parameters were found to be important in different runs of our procedure. Note that the set of selected key parameters was remarkably robust across runs.

The most extreme case is SPEAR-SWV, for which SPEAR's variable selection heuristic (sp-var-dec-heur) was found to be the most important parameter every single time by a wide margin, followed by its phase selection heuristic (sp-phase-dec-heur). The importance of the variable selection heuristic for SAT solvers is well known, but it is surprising that the importance of this choice dominates so clearly. Phase selection is also widely known to be important for the performance of modern CDCL SAT solvers like SPEAR. As can be seen from Fig. 1 (top middle), predictive models for SPEAR-SWV based on 2 parameters essentially performed as well as those based on all parameters, as is also reflected in the very low importance ratings for all but these two parameters.

In the case of both CPLEX-BIGMIX and LK-H-TSPLIB, up to 5 parameters show up as important, which is not surprising, considering that predictive performance of subset models with 5 inputs converged to that of models with all inputs (see Fig. 1, top left and right). In the case of CPLEX, the key parameters included two controlling CPLEX's cutting strategy (mip_limits_cutsfactor and mip_limits_cutpasses, limiting the number of cuts to add, and the number of cutting plane passes,

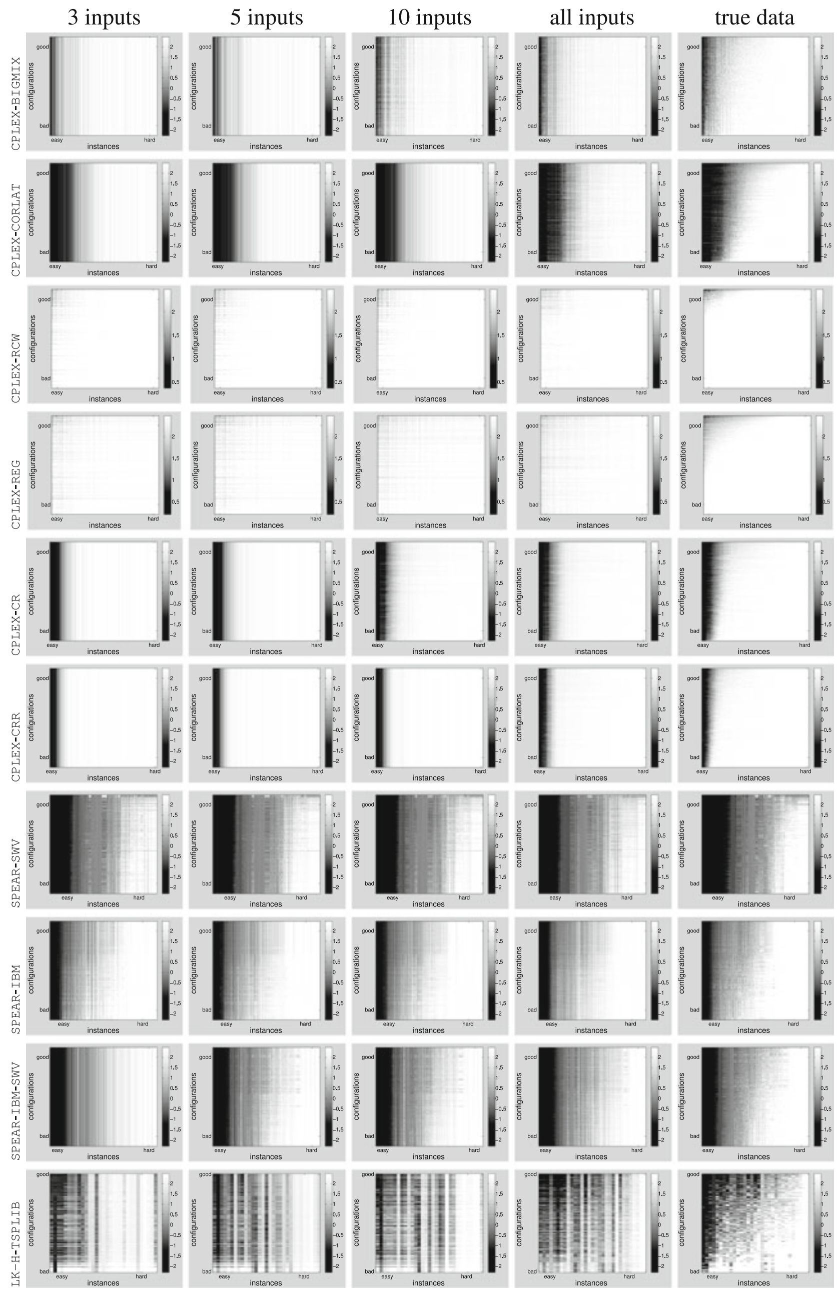
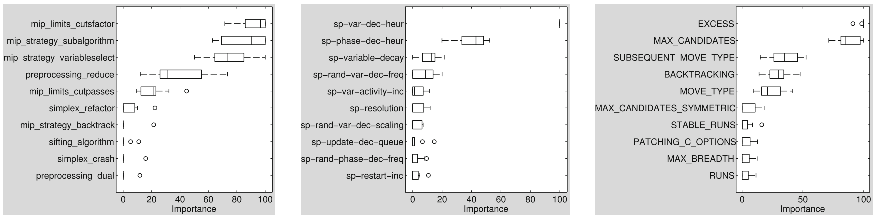


Fig. 2. Performance predictions by random forest models based on subsets of features and parameters. To generate these heatmaps, we ordered configurations by their average performance across instances, and instances by their average hardness across configurations; the same ordering (based on the true heatmap) was used for all heatmaps. All data shown is test data.



(a) `CPLEX-BIGMIX` (b) `SPEAR-SWV` (c) `LK-H-TSPLIB`

Fig. 3. Parameter importance for 3 example datasets. We show boxplots over 10 repeated runs with different random training/validation/test splits.

respectively), two MIP strategy parameters (mip_strategy_subalgorithm and mip_strategy_variableselect, determining the continuous optimizer used to solve subproblems in a MIP, and variable selection, respectively), and one parameter determining which kind of reductions to perform during preprocessing (preprocessing_reduce). In the case of `LK-H`, all top five parameters are related to moves, parameterizing candidate edges (EXCESS and MAX_CANDIDATES, limiting the maximum alpha-value allowed for any candidate edge, and the maximum number of candidate edges, respectively), and move types (MOVE_TYPE, BACKTRACKING, SUBSEQUENT_MOVE_TYPE, specifying whether to use sequential k-opt moves, whether to use backtracking moves, and which type to use for moves following the first one in a sequence of moves).

To demonstrate the model independence of our approach, we repeated the same analysis based on other empirical performance models (linear regression, neural networks, Gaussian processes, and regression trees). Although overall, these models yielded weaker predictions, the results were qualitatively similar: for `SPEAR`, all models reliably identified the same two parameters as most important, and for the other datasets, there was an overlap of at least three of the top five ranked parameters. Since random forests yielded the best predictive performance, we focus on them in the remainder of this paper.

As an aside, we note that the fact that a few parameters dominate importance is in line with similar findings in the machine learning literature on the importance of hyperparameters, which has informed the analysis of a simple hyperparameter optimization algorithm [36] and the design of a Bayesian optimization variant for optimizing functions with high extrinsic but low intrinsic imensionality [37]. In future work, we plan to exploit this insight to design better automated algorithm configuration procedures.

Next, we demonstrate how we can study the joint importance of instance features and algorithm parameters. Since foward selection by itself chose mostly instance features, for this analysis we constrained it to select 3 features and 2 parameters. Table 2 lists the features and parameters identified for our 3 example datasets, in the order forward selection picked them. Since most instance features are strongly correlated with each other, it is important to measure and understand our importance metric in the context of the specific subset of inputs it is computed for. For example, consider the set of important features for dataset

Table 2. Key inputs, in the order in which they were selected, along with their omission cost from this set.

Dataset	CPLEX-BIGMIX	SPEAR-SWV	LK-H-TSPLIB
1st selected	cplex_prob_time (10.1)	Pre_featuretime (35.9)	tour_const_heu_avg (0.0)
2nd selected	obj_coef_per_constr2_std (7.7)	nclausesOrig (100.0)	cluster_distance_std (0.8)
3rd selected	vcg_constr_weight0_avg (30.2)	sp-var-dec-heur (32.6)	EXCESS (10.0)
4th selected	mip_limits_cutsfactor (8.3)	VCG_CLAUSE_entropy (34.5)	bc_no1s_q25 (100.0)
5th selected	mip_strategy_subalgorithm (100.0)	sp-phase-dec-heur (27.6)	BACKTRACKING (0.0)

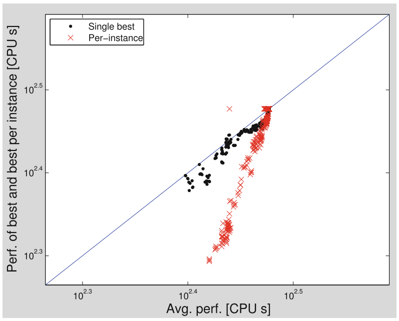
Table 3. Key parameters and their best fixed values as judged by an empirical performance model based on 3 features and 2 parameters.

Dataset	1st selected param	2nd selected param
CPLEX-BIGMIX	mip_limits_cutsfactor = 8	mip_strategy_subalgorithm = 2
CPLEX-CORLAT	mip_strategy_subalgorithm = 2	preprocessing_reduce = 3
CPLEX-REG	mip_strategy_subalgorithm = 2	mip_strategy_variableselect = 4
CPLEX-RCW	preprocessing_reduce = 3	mip_strategy_lbheur = no
CPLEX-CR	mip_strategy_subalgorithm = 0	preprocessing_reduce = 1
CPLEX-CRR	preprocessing_coeffreduce = 2	mip_strategy_subalgorithm = 2
SPEAR-IBM	sp-var-dec-heur = 2	sp-resolution = 0
SPEAR-SWV	sp-var-dec-heur = 2	sp-phase-dec-heur = 0
SPEAR-SWV-IBM	sp-var-dec-heur = 2	sp-use-pure-literal-rule = 0
LK-H-TSPLIB	EXCESS = −1	BACKTRACKING = NO

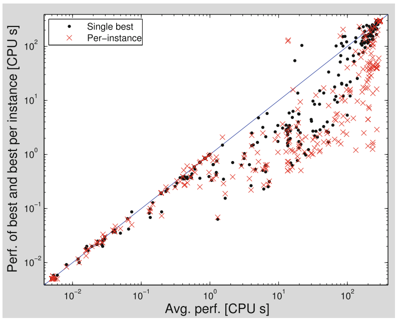
CPLEX-BIGMIX (Table 2, left). While the single most important feature in this case was cplex_prob_time (a timing feature measuring how long CPLEX probing takes), in the context of the other four features, its importance was relatively small; on the other hand, the input selected 5th, mip_strategy_subalgorithm (CPLEX's MIP strategy parameter from above) was the most important input in the context of the other 4. We also note that all algorithm parameters that were selected as important in this context of instance features (mip_limits_cutsfactor and mip_strategy_subalgorithm for CPLEX; sp-var-dec-heur and sp-phase-dec-heur for SPEAR; and EXCESS and BACKTRACKING for LK-H) were already selected and labeled important when considering only parameters. This finding increases our confidence in the robustness of this analysis.

4.3 Selecting Values for Key Parameters

Next, we used our subset models to identify which values the key parameters identified by forward selection should be set to. For each dataset, we used the same subset models of 3 features and 2 parameters as above; Table 3 lists the best predicted values for these 2 parameters. The main purpose of this experiment was to demonstrate that this analysis can be done automatically, and we thus



(a) CPLEX-RCW



(b) SPEAR-SWV

Fig. 4. Performance of random configurations *vs* configurations setting almost all parameters at random, but setting 2 key parameters based on an empirical performance model with 3 features and 2 parameters.

only summarize the results at a high level; we see them as a starting point that can inform domain experts about empirical properties of their algorithm in a particular application context and trigger further in-depth studies. At a high level, we note that CPLEX's parameter mip_strategy_subalgorithm (determining the continuous optimizer used to solve subproblems in a MIP) was important for most instance sets, the most prominent values being 2 (use CPLEX's dual simplex optimizer) and 0 (use CPLEX's auto-choice, which also defaults to the dual simplex optimizer). Another important choice was to set preprocessing_reduce to 3 (use both primal and dual reductions) or 1 (use only primal reductions), depending on the instance set. For SPEAR, the parameter determining the variable selection heuristic (sp-var-dec-heur) was the most important one in all 3 cases, with an optimal value of 2 (select variables based on their activity level, breaking ties by selecting the more frequent variable). For good average performance of LK-H on TSPLIB, the most important choices were to set EXCESS to -1 (use an instance-dependent setting of the reciprocal problem dimension), and to not use backtracking moves.

We also measured the performance of parameter configurations that actually set these parameters to the values predicted to be best by the model, both on average across instances and in an instance-specific way. This serves as a further way of evaluating model quality and also facilitates deeper understanding of the parameter space. Specifically, we consider parameter configurations that instantiate the selected parameters according to the model and assign all other parameter to randomly sampled values; we compare the performance of these configurations to that of configurations that instantiate *all* values at random. Figure 4 visualizes the result of this comparison for two datasets, showing that the model indeed selected values that lead to high performance: by just controlling two parameters, improvements of orders of magnitude could be achieved for some instances. Of course, this only compares to random configurations; in contrast to our work on algorithm configuration, here, our goal was to gain a better understanding of an

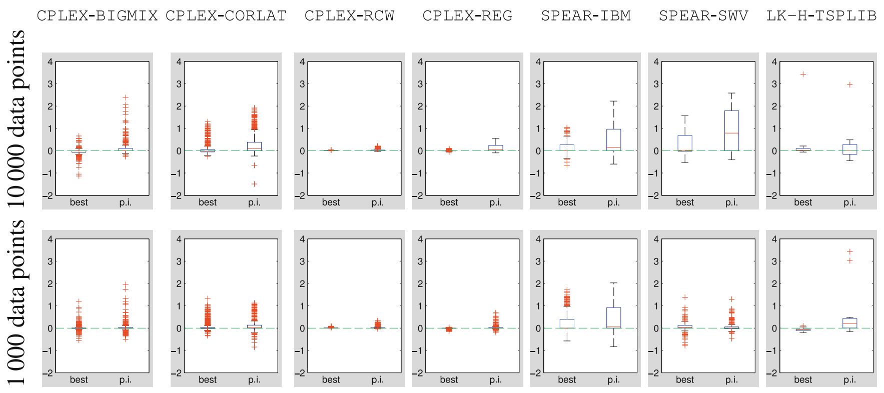


Fig. 5. Log_{10} speedups over random configurations by setting almost all parameters at random, except 2 key parameters, values for which (fixed best, and best per instance) are selected by an empirical performance model with 3 features and 2 parameters. The boxplots show the distribution of $\log_{10}$ speedups across all problem instances; note that, e.g., a $\log_{10}$ speedup of 0, -1, and 1 mean identical performance, a 10-fold slowdown, and a 10-fold speedup, respectively. The dashed green lines indicate where two configurations performed the same, points above the line indicate speedups. Top: based on models trained on 10 000 data points; bottom: based on models trained on 1 000 data points.

algorithms' parameter space rather than to improve over its manually engineered default parameter settings.[3] However, we nevertheless believe that the speedups achieved by setting only the identified parameters to good values demonstrate the importance of these parameters. While Fig. 4 only covers 2 datasets, Fig. 5 (top) summarizes results for a wide range of datasets. Figure 5 (bottom) demonstrates that predictive performance does not degrade much when using sparser training data (here: 1 000 instead of 10 000 training data points); this is important for facilitating the use of our approach in practice.

5 Conclusions

In this work, we have demonstrated how forward selection can be used to analyze algorithm performance data gathered using randomly sampled parameter configurations on a large set of problem instances. This analysis identified small sets of key algorithm parameters and instance features, based on which the performance of these algorithms could be predicted with surprisingly high accuracy. Using this fully automated analysis technique, we found that for high-performance solvers for some of the most widely studied NP-hard combinatorial problems, namely SAT, MIP and TSP, only very few key parameters (often just two of dozens) largely determine algorithm performance. Automatically constructed performance models, in our case based on random forests, were of sufficient

[3] In fact, in many cases, the best setting of the key parameters were their default values.

quality to reliably identify good values for these key parameters, both on average across instances and dependent on key instance features. We believe that our rather simple importance analysis approach can be of great value to algorithm designers seeking to identify key algorithm parameters, instance features, and their interaction.

We also note that the finding that the performance of these highly parametric algorithms mostly depends on a few key parameters has broad implications on the design of algorithms for NP-hard problems, such as the ones considered here, and of future algorithm configuration procedures.

In future work, we aim to reduce the computational cost of identifying key parameters; to automatically identify the relative performance obtained with their possible values; and to study which parameters are important in high-performing regions of an algorithm's configuration space.

References

1. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Proceedings of CPAIOR-10, pp. 186–202 (2010)
2. Nannen, V., Eiben, A.E.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Proceedings of IJCAI-07, pp. 975–980 (2007)
3. Ansotegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of solvers. In: Proceedings of CP-09, pp. 142–157 (2009)
4. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated F-race: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Empirical Methods for the Analysis of Optimization Algorithms. Springer, Heidelberg (2010)
5. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR **36**, 267–306 (2009)
6. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 5. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011)
7. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, vol. 7219, pp. 55–70. Springer, Heidelberg (2012)
8. Hutter, F., Babić, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: Proceedings of FMCAD-07, pp. 27–34 (2007)
9. Chiarandini, M., Fawcett, C., Hoos, H.: A modular multiphase heuristic solver for post enrolment course timetabling. In: Proceedings of PATAT-08 (2008)
10. Vallati, M., Fawcett, C., Gerevini, A.E., Hoos, H.H., Saetti, A.: Generating fast domain-optimized planners by automatically configuring a generic parameterised planner. In: Proceedings of ICAPS-PAL11 (2011)
11. Ridge, E., Kudenko, D.: Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In: Proceedings of PPSN-06, pp. 27–34 (2006)
12. Chiarandini, M., Goegebeur, Y.: Mixed models for the analysis of optimization algorithms. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 225–264. Springer, Berlin (2010)

13. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation: The New Experimentalism. Natural Computing Series. Springer, Berlin (2006)
14. Finkler, U., Mehlhorn, K.: Runtime prediction of real programs on real machines. In: Proceedings of SODA-97, pp. 380–389 (1997)
15. Fink, E.: How to solve it automatically: selection among problem-solving methods. In: Proceedings of AIPS-98, pp. 128–136. AAAI Press (1998)
16. Howe, A.E., Dahlman, E., Hansen, C., Scheetz, M., Mayrhauser, A.: Exploiting competitive planner performance. In: Biundo, S., Fox, M. (eds.) ECP 1999. LNCS, vol. 1809, pp. 62–72. Springer, Heidelberg (2000)
17. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the Empirical Hardness of Optimization Problems. In: Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 556–572. Springer, Heidelberg (2002)
18. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
19. Smith-Miles, K.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. 41(1), 6:1–6:25 (2009)
20. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Comput. Oper. Res. **39**(5), 875–889 (2012)
21. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: portfolio-based algorithm selection for SAT. JAIR **32**, 565–606 (2008)
22. Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K.: Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 213–228. Springer, Heidelberg (2006)
23. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: the state of the art. CoRR, abs/1211.0906 (2012)
24. Smith-Miles, K., van Hemert, J.: Discovering the suitability of optimisation algorithms by learning from evolved instances. AMAI **61**, 87–104 (2011)
25. Bartz-Beielstein, T., Markon, S.: Tuning search algorithms for real-world applications: a regression tree based approach. In: Proceedings of CEC-04, pp. 1111–1118 (2004)
26. Bishop, C.M.: Pattern recognition and machine learning. Springer, New York (2006)
27. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, Cambridge (2006)
28. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
29. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Empirical hardness models: methodology and a case study on combinatorial auctions. J. ACM **56**(4), 1–52 (2009)
30. Xu, L., Hoos, H.H., Leyton-Brown, K.: Predicting satisfiability at the phase transition. In: Proceedings of AAAI-12 (2012)
31. Friedman, J.: Multivariate adaptive regression splines. Ann. Stat. **19**(1), 1–141 (1991)
32. IBM Corp.: IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ (2012). Accessed 27 Oct 2012
33. Babić, D., Hutter, F.: Spear theorem prover. Solver description SAT competition (2007)
34. Helsgaun, K.: General *k*-opt submoves for the Lin-Kernighan TSP heuristic. Math. Program. Comput. **1**(2–3), 119–163 (2009)
35. Styles, J., Hoos, H.H., Müller, M.: Automatically configuring algorithms for scaling performance. In: Hamadi, Y., Schoenauer, M. (eds.) LION 6. LNCS, vol. 7219, pp. 205–219. Springer, Heidelberg (2012)

36. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. JMLR **13**, 281–305 (2012)
37. Wang, Z., Zoghi, M., Hutter, F., Matheson, D., de Freitas, N.: Bayesian optimization in a billion dimensions via random embeddings. ArXiv e-prints, January (2013). arXiv:1301.1942