

Performance-Influence Models of Multigrid Methods: A Case Study on Triangular Grids

Alexander Grebhahn^{1*}, Carmen Rodrigo², Norbert Siegmund³,
Francisco J. Gaspar², Sven Apel¹

¹*Department of Computer Science and Mathematics, University of Passau, Germany*

²*IUMA, Department of Applied Mathematics, University of Zaragoza, Spain*

³*Department of Media, Bauhaus-University Weimar, Germany*

SUMMARY

Multigrid methods are among the most efficient algorithms for solving discretized partial differential equations. Typically, a multigrid system offers various configuration options to tune performance for different applications and hardware platforms. However, knowing the best-performing configuration in advance is difficult, because measuring all multigrid-system variants is costly. Instead of direct measurements, we use machine learning to predict the performance of the variants. Selecting a representative set of configurations for learning is non-trivial, though, but key to prediction accuracy. We investigate different sampling strategies to determine the tradeoff between accuracy and measurement effort. In a nutshell, we learn a performance-influence model that captures the influences of configuration options and their interactions on the time to perform a multigrid iteration and relate this to existing domain knowledge. In an experiment on a multigrid system working on triangular grids, we found that combining pair-wise sampling with the D-Optimal experimental design for selecting a learning set yields the most accurate predictions. After measuring less than 1% of all variants, we were able to predict the performance of all variants with an accuracy of 95.9%. Furthermore, we were able to verify almost all knowledge on the performance behavior of multigrid methods provided by two experts.

Copyright © 2010 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: Performance Prediction, Triangular Grids, Multigrid Method, Configurable Systems, SPL Conqueror, Sampling

1. INTRODUCTION

Many important problems, ranging from scientific phenomena and industrial applications to problems in economics and medicine, are mathematically modeled by partial differential equations (PDEs). Discretizations based on finite differences, volumes, or elements adequately approximate the underlying continuous problems, giving rise to large sparse systems of algebraic equations, which shall be efficiently solved. Since their development in the 1970s, multigrid methods [1, 2, 3, 4] have been proved to be among the most efficient iterative algorithms for solving this type of equations. Success factors of multigrid methods are their low computational complexity and their convergence being independent of the discretization parameters. They are ‘optimal’ in a complexity sense, since the number of arithmetic operations needed to solve a discrete problem is proportional to the number of unknowns in the problem. Moreover, a characteristic feature of the iterative

*Correspondence to: Alexander Grebhahn, Department of Computer Science and Mathematics, University of Passau, Innstr. 33, 94032 Passau, Germany.

multigrid approach is that its convergence speed is independent of the discretization grid size. Multigrid methods exploit hierarchies of computational levels of various resolutions to substantially accelerate the convergence of a basic iterative solver. They base on two main principles: the *smoothing property* of classic iterative methods and the *coarse-grid correction* principle. The two are combined by means of the multigrid algorithm, but care is needed to choose the algorithmic variants and parameters used in the multigrid system to achieve optimal performance for a given problem. We call the choices of algorithms and parameters of the multigrid system *configuration options*, and a valid selection of configuration options a *configuration*, which gives rise to a running *variant* of the multigrid system.

It is well-known that the performance of multigrid methods strongly depends on making the right algorithmic and parameter choices, namely the smoother, the construction of coarser levels and the discrete problem considered on them, the inter-grid transfer operators (restriction and prolongation), the cycle type, and the number of pre- and post-smoothing steps [3]. On the one hand, there are no general rules to guide the task of successfully choosing the best variant. On the other hand, the direct measurement of all system variants to find the best one is costly and in many cases infeasible.

Since multigrid methods are iterative algorithms, the time to solution can be separated into the number of iterations and the time per iteration. To compute the number of iterations needed to achieve a specific convergence, local Fourier analysis (LFA) can be used. LFA was introduced by Achi Brandt in 1977 [1], and since its development, it has been successfully applied in many contexts, becoming the main quantitative analysis to study the convergence of geometric multigrid algorithms [3, 5, 6]. LFA not only provides accurate predictions of convergence rates, but also gives advice for an adequate composition of the method. Although the number of iterations of a multigrid system configuration can be computed using LFA, it does not give any indications about the time to solution of the given variant. Thus, LFA has to be combined with another approach to identify the performance-optimal variant.

Machine learning could be a feasible approach to predict the time needed for an iteration. The goal is to learn the influence of individual configuration options and their interactions on performance [7, 8] such that we can compute the performance-optimal variant in terms of the time needed for an iteration. To this end, we use SPL Conqueror's approach based on multivariate regression and forward feature selection to obtain a *performance-influence model*, which captures the relevant influences in a human-readable way. As input for learning a performance-influence model, we use a structured sample of variants that are distributed over the whole configuration space according to certain criteria that are specified by a sampling strategy. We distinguish between sampling of binary and numeric options, due to the different value ranges and their substantial differences on the influences on performance. For each of the two groups, we have a set of different sampling strategies that make different assumptions about the influence of the configuration options on the performance of a system variant. Thus, selecting a suitable sampling strategy is essential to learn an accurate performance-influence model. One goal of this article is to compare the different sampling strategies regarding accuracy of the derived performance-influence models and the number of configurations selected by the strategies.

In prior work, we demonstrated that SPL Conqueror's approach is able to learn accurate performance-influence models after measuring only a small number of system variants [7, 9]. There, we demonstrated that this even works for complex systems, such as the Java garbage collector. Here, we aim at predicting the time needed for one iteration of a multigrid system variant after measuring only a small number of variants. Additionally, we perform a deeper analysis of the learned performance-influence model, to validate whether the identified influences match existing domain knowledge (and are not just an artifact of learning in the presence of noise). In the multigrid system we study, we identified the influence of different smoothers, cycle types, a relaxation parameter, pre- and post-smoothing steps, and also the geometry within the grid on the time needed to perform one iteration of the multigrid system. A deeper analysis of the performance-influence model is imperative as SPL Conqueror's approach is meant to support domain experts in exploring the performance behavior of their applications. As a first step, we have to test whether it is possible to discover existing domain knowledge using SPL Conqueror's approach (e.g., which

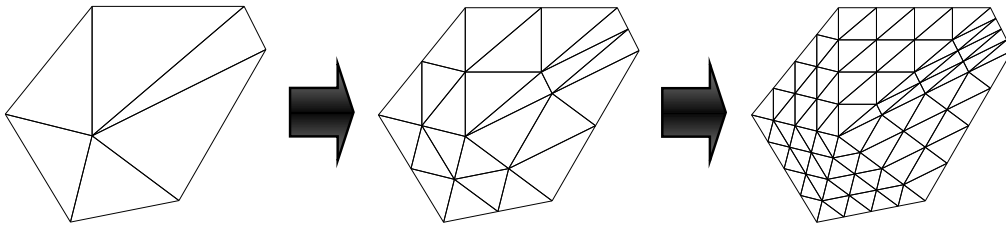


Figure 1. Hierarchy of semi-structured grids.

kind of smoothers need more time for an iteration of a multigrid algorithm or what is the underlying performance-contribution function of the pre-smoothing and post-smoothing steps) and possibilities to discover also new knowledge about the influences of the options.

Overall, we make the following contributions:

- We show that performance-influence models can be derived automatically to predict the time needed to perform a single iteration of a multigrid system that solves the Poisson equation, using triangular grids as underlying data structure.
- We compare different sampling strategies regarding measurement effort and accuracy of the models learned from the set of variants selected by the sampling strategies.
- We validate whether discovered influences of the configuration options and interactions among them match existing knowledge about theoretical influences of the options.

In our experiments, considering a system that solves the Poisson equation on a two-dimensional grid, we can predict the runtime of a multigrid iteration with an accuracy of about 87 %, independently of the sampling strategies used for selecting a set of system variants used as learning set for SPL Conqueror. Using the best combination of the sampling strategies regarding prediction accuracy, we achieve an accuracy of almost 96 %. For all of the different sampling strategies, we have to measure less than 1 % of all configurations during learning. Furthermore, we were able to discover almost all existing knowledge about the influence of configuration options provided by two domain experts. Overall, our predictions are in line with the provided knowledge, except for one hypothesis and four relevant option combinations, as we discuss in Section 5. This finding, however, opens an interesting avenue of further work.

2. CASE STUDY: A MULTIGRID SOLVER ON TRIANGULAR GRIDS

In this section, we describe the multigrid system that we use as case study in our experiments. We provide details on the different algorithms and parameters we use in our later experiments to provide information about the influences of the options on the performance of the system and to show the differences between the different configuration options. To this end, we first describe the system with its algorithms and parameters in Section 2.1 and then use a domain modeling technique to present the variability we consider in our experiments in a human readable way in Section 2.2.

2.1. Domain Analysis

Grid structure: Mathematical models describing problems in physics and engineering often must be solved on complex computational domains. Triangular grids are usually preferred for solving these problems, due to the geometric flexibility, which enables stakeholders to fit the geometry of arbitrarily shaped domains. A classic approach is to consider an unstructured triangular grid, which leads to a stiffness matrix in sparse matrix format that, however, cannot be treated by using

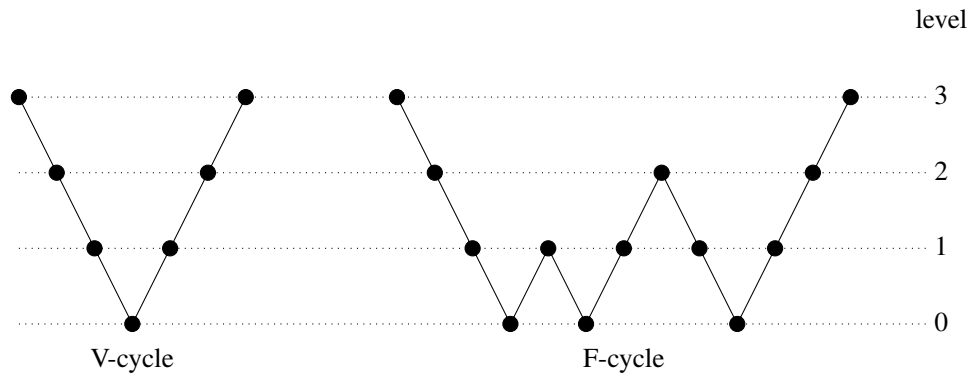


Figure 2. Different multigrid cycle types.

geometric data structures. Semi-structured grids provide a viable framework to achieve a good agreement between the flexibility of the unstructured grids and the efficiency of the structured data structures [10, 11, 12]. Although a multigrid algorithm can work with semi-structured grids efficiently, there is a large number of algorithmic and parameters choices to tune its performance further.

Smoother: First, the relaxation process, which is used to eliminate the high-frequency components of the error on each grid of the hierarchy, is one of the most important parts of the algorithm [3]. It is commonly called a *smoother*, and has to be carefully chosen in many applications (such as in anisotropic problems, fluid-flow models, and porous-media flows). There is an extensive list of iterative methods that can be applied as smoothers inside a multigrid algorithm, ranging from basic pointwise relaxation methods, such as Jacobi or Gauss-Seidel, to blockwise smoothers or even other more sophisticated options, such as ILU-type or distributive smoothers. Pointwise relaxation is the cheapest alternative, but in some applications it does not perform satisfactorily [4]. Furthermore, pointwise methods can be generalized to blockwise iterative schemes by considering the update of a whole set of unknowns at each time. This is the case of linewise smoothing, which, for example, simultaneously updates all the unknowns located at the same line of the grid [4]. These schemes are computationally more expensive, but they become very attractive and even mandatory when anisotropies appear [4, 12]. Another example of blockwise relaxation are Vanka smoothers, which are widely used when solving a saddle point problem [13, 14]. In our case study, we focus on the application of Jacobi and three-color pointwise smoothers as well as on lexicographic and zebra-type linewise relaxations on triangular grids [12]. Moreover, we need to specify the number of smoothing iterations that we apply on each grid before processing to a coarser grid (pre-smoothing) and after processing to a finer grid (post-smoothing). Here, advanced blocking mechanisms, such as temporal blocking, can be used to reduce the time needed to perform pre-smoothing and post-smoothing [15].

Relaxation parameter: Furthermore, we tune smoothing properties of the relaxation process by adding a relaxation parameter to our system, which has an influence on the convergence rate and therefore on the number of iterations needed for convergence.

Discretization operator: Another relevant configuration option of multigrid systems is the choice of the discretization operator on the grid hierarchy. We will simply use direct discretization on each grid of the hierarchy in our system because, for the Poisson equation, which is the problem considered in our algorithm, we do not need to apply more complex discretization strategies, such as a Galerkin coarse-grid operator [3].

Cycle types: In general, the idea of using a sequence of grids permits us to visit them in different ways. These possibilities are determined by the cycle index, indicating the number of multigrid steps to perform on coarser grids. A cycle index of 1 leads to the V-cycle [3], which is the easiest

recursive definition of a multigrid cycle. In a V-cycle, the defined number of pre-smoothing and post-smoothing sweeps are applied only once on each grid. By visiting coarser grids multiple times in one iteration of the multigrid algorithm, other types of cycles can be defined. For example, a cycle index of 2 gives rise to the W-cycle [3]. The also commonly used F-cycle [3] goes down to the coarsest grid and recursively interpolates to the next finer grid. In each of these interpolations, we apply a V-cycle until we get to the finest grid. In Figure 2, we present an example for the structure of the V-cycle and the F-cycle. The latter usually gives a convergence rate close to that of the W-cycle, but with lower computational cost. As a consequence, we focus on the V-cycle and the F-cycle.

Restriction and prolongation: As a further possibility for tuning, we need to decide how we transfer information between the coarse and fine grids, that is, the choice of restriction and prolongation operators. The choice of the inter-grid transfer operators is, of course, closely related to the coarsening strategy, so that the corresponding configuration options have a combined influence on the performance of the overall system.

Grid geometry: To summarize, the choice of the algorithms and parameters used in the multigrid algorithm has a strong influence on its performance and, as a consequence, also on the performance of the overall application. However, the different parameters and algorithms may interfere with each other. For example, the geometry of the grid can induce some anisotropies that may imply a deterioration in the performance of some multigrid algorithms. In particular, each regularly structured patch of the grid will be characterized by two angles of the corresponding input triangle, α and β . Therefore, for each of these patches, it is very important to select good algorithms with respect to these angles. Consequently, we take them into account when studying the performance of the multigrid solver.

2.2. Domain Modeling

To specify which variants of the multigrid system are valid, we use a *feature model* [16]. Feature models have a tree-like structure defining relations between options (also called features). Additionally, one can define cross-tree constraints between options in the form of logic formulas. In Figure 3, we present the feature model of the configurable multigrid system working on triangular grids that is used in our experiments (Section 4 and 5). The system provides numerical and binary configuration options. While binary options can only be disabled or enabled, which maps to 0 and 1, respectively, numeric options can have a larger value domain. In Figure 3, numeric configuration options are denoted with dashed boxes and binary options with solid boxes. For numeric options, we provide the considered value domain, described by the minimal value, the maximal value, and the default value of the option. For example, the *pre-smoothing* option takes an integer value between 0 and 8, with a default value of 4. In general, binary configuration options can be mandatory or optional and also be grouped, for instance, into alternatives, to describe that exactly one option must be selected in an *alternative group* (e.g., one cycle type must be selected in every system variant). The model of Figure 3 contains also two constraints on numeric options of the system. They state that, at least, one *pre-smoothing* or one *post-smoothing* step has to be performed, and that the sum of the two interior angles α and β has to be between 90 and 180.

Overall, the multigrid system that we use for our experiments can use one of four different smoothers, it can use different cycle types, and also have a relaxation parameter. Additionally, the number of pre-smoothing and post-smoothing iterations on each grid level can be configured. Last, we also see the size of the interior angles of an element of the grid as another configuration option.

3. PERFORMANCE-INFLUENCE MODELS

Knowing the performance of all variants of a configurable system is essential to identify the optimal system variant, to identify specific performance characteristics, such as performance bugs or unknown interactions between configuration options, and, as result, to understand the overall system. To this end, we aim at learning a *performance-influence model* of the system that captures the influence of configuration options and their interactions on performance in a human readable

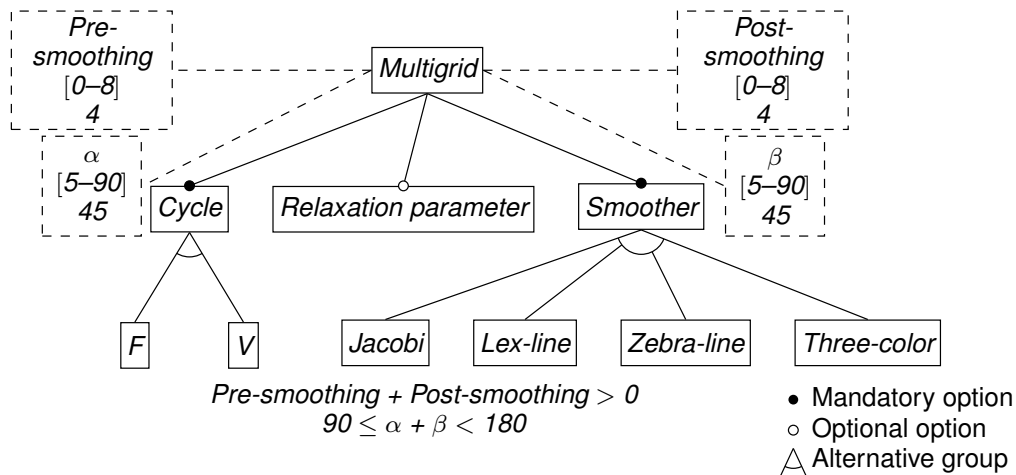


Figure 3. Feature model of the configurable multigrid system

form [7]. To learn a performance-influence model, we have to select a representative set of system variants and learn performance characteristics based on the observed performance of this set. Because we do not need to perform any modification on the code of the system, this machine learning approach is a black box approach. However, identifying a representative set of variants is not trivial, because influences of the options are often unknown and options might interact with each other. Thus, selecting the optimal sampling strategy is a key success factor. In what follows, we present different sampling strategies (Section 3.1) and outline SPL Conqueror's machine-learning approach (Section 3.2).

3.1. Sampling

To sample a proper set of configurations for learning a performance-influence model, we need to consider two types of configuration options: binary options and numeric options. These two types have to be handled differently, because of their different value domains. Hence, we split the set of configuration options into two sets, one containing only binary options and one containing only numeric options. For the set of binary options, we use sampling heuristics developed for detecting interactions between features [8]. For the numeric options, we use experimental designs [17]. All of these strategies aim at identifying the individual influences and interactions up to a certain degree (i.e., the Pair-Wise heuristics aims at identifying all interactions among each pair of binary options). The assumption that interactions up to a certain degree are most relevant is reasonable, because it has been observed that interactions of a high degree (interactions between a high number of options) usually have a small influence compared to interactions of a lower degree or individual influences and are unlikely to occur [18]. After sampling the two configuration spaces, we obtain two sets of partial configurations (i.e., a set containing combinations of selected binary options and a set containing numeric options with their chosen values). Then, we compute the Cartesian product of the two sets of partial configurations to create the configurations used as input for learning.

Binary Sampling Heuristics.

We use heuristics that aim at selecting those system variants that help in identifying interactions of a certain degree including the influence of individual options [8, 19]. In particular, we use three different sampling heuristics: the Option-Wise heuristics, the Pair-Wise heuristics, and the Negative Option-Wise heuristics.

The *Option-Wise heuristics (OW)* selects variants such that individual influences of configuration options can be pinpointed. To this end, we generate one variant for each binary option, such that the option is enabled and all other options are disabled, if possible. After creating this set of variants, we additionally select a variant with all options disabled. Based on this set, we can determine the

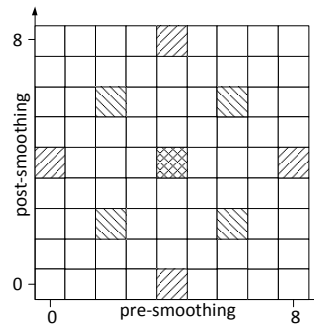





Figure 4. The Central Composite Inscribed Design for two numeric options with value ranges of 0 to 8. The center point is marked with , the corner points with , and the axis points with .

individual influences of all options ruling out interactions. This leads to a number of variants that is linear in the number of binary options.

The *Pair-Wise heuristics (PW)* selects a set of system variants such that interactions between each pair of binary configuration options can be identified. To this end, one distinct variant is selected for each pair of options, where only the options of interest are enabled and all other options are disabled, if possible. Thus, we select one variant for each pair of options. Using this heuristics, it is possible to identify all interactions between each pair of options, but no interactions between three or more binary options (only by chance, if at all). The number of variants selected this way is quadratic in the number of binary options.

The *Negative Option-Wise heuristics (NegOW)* selects variants with the maximal number of options enabled (where as the Option-Wise heuristics selects variants with a minimal number of options enabled). As a result, this heuristics aims at identifying the influence of configuration options in the presence of all possible interactions. For each option, one variant is selected, such that the option itself is disabled and all other options are enabled, if possible. Much like the OW heuristics, the NegOW heuristics selects a number of variants that is linear in the number of binary options.

Experimental Designs.

Sampling in the presence of numeric options is a widely researched area known under the umbrella of *experimental designs* [17]. These designs are, for example, used as response-surface methods to identify the influence of independent variables on a dependent variable, such as in chemistry or biology [17, 20]. In the last century, a large number of different experimental designs have been proposed. Here, we focus on a small set of widely used and successful experimental designs. In general, experimental designs can be separated into two groups: standard designs and optimal designs [17]. Standard designs use a predefined pattern to select configurations from the overall configuration space. By contrast, optimal designs select the configurations based on mathematical criteria, such as maximizing the distance between configurations in the configuration space. To identify an optimal set according to a given criterion, it is necessary to validate all different subsets based on the criterion. This is time consuming and does not scale for larger configuration spaces.

As representatives of standard experimental designs, we use the *Central Composite Design* and the *Plackett-Burman Design*. While the Central Composite Design is one of the widely used designs used to identify a response surface [17], the Plackett-Burman Design yielded the most promising results in a previous study [7]. For the optimal designs, we selected the *D-Optimal Design* [21], because it selects configurations according to an optimization criteria that fits best for our purpose (configurations with the largest distance to each other are selected). Last, we also use a random selection of variants, since this is a standard method in machine learning and often leads to good results [22, 23].

Central Composite Design: Box and Wilson introduced the *Central Composite Design (CCD)* in 1951 [24]. It selects configurations for learning a second-order model [17]. A second-order model

		options							
configurations		0	1	2	2	0	2	1	1
		1	0	1	2	2	0	2	1
		1	1	0	1	2	2	0	2
		2	1	1	0	1	2	2	0
		0	2	1	1	0	1	2	2
		2	0	2	1	1	0	1	2
		2	2	0	2	1	1	0	1
		1	2	2	0	2	1	1	0
		0	0	0	0	0	0	0	0

Figure 5. Plackett-Burman Design for eight options using the seed (9,3). In each column, the values of one option are given and, in each row, the values of one configuration. In our experiments, we map the values 0, 1, and 2 to the minimal, the center, and the maximal value of the options.

considers the linear influence of all options, the quadratic influence of all options, and all interactions between two options [25]. To this end, the configurations selected by the design consist of three portions:

- The corner points of the configuration space. These are the points with the minimal or maximal value of the numeric configuration options.
- A set of axial and star points with a defined distance to the center point of the value range.
- The center point of the configuration space.

These three sets of points give rise to $2^k + 2k + 1$ different configurations, where k is the number of numeric options. Depending on the distance of the axis points to the center point, different variants of the design are created. Specifically, we use the *Central Composite Inscribed Design*, in which axes and corner points have the same distance to the center point. In Figure 4, we give an example for the configurations selected by the Central Composite Inscribed Design when considering *pre-smoothing* and *post-smoothing*. Each cell represents a valid value combination of the two options, and the hatched cells are the selected value combinations.

Plackett-Burman Design: We use the *Plackett-Burman Design (PBD)* as the second standard design. It was developed for configuration spaces, in which the strength of interaction effects between options are negligible compared to the individual influences of configuration options [26]. Using this design, the number of configurations selected does not grow exponentially with the number of configuration options, as for the CCD. Instead, the design defines seeds specifying the number of configurations and the number of different values considered for the numeric option. In Figure 5, we give an example for a PBD with a seed specifying that 9 value combinations have to be selected for a configuration space of 8 options. Using this seed, 3 different values of options are considered, which are denoted by 0, 1, and 2. We map the values 0, 1, and 2 to the minimal, the center, and the maximal value of the value range of the options in question. Thus, each row of the table defines the values used in one configuration. The initial seed in the first row is shifted to the right, to compute the resulting set of configurations. One additional configuration, in which all numeric options are set to their minimal value, is added to this set of configurations. Henceforth, we use PBD(9, 3) to refer to a Plackett-Burman Design using a seed that defines that 9 configurations are selected and 3 different values of one option are considered. In our experiments, we also use seeds defining that 5 or 7 distinct values of a single option are considered. One of these seeds defines that 5 distinct values and 125 configurations are selected, leading to the PBD(125, 5). The other seed we use defines that 7 distinct values and 49 configurations are selected, which leads to the PBD(49, 7). For both seeds, we make an equidistant mapping from the values defined in the seeds to the actual values of the options in question.

D-Optimal Design: As a representative of optimal designs, we selected the *D-Optimal Design (DOD)* proposed by Smith [27]. The general idea behind optimal designs is to select a subset of a defined size of all configurations that is optimal with respect to a specific mathematical property. To this end, the configurations selected are stored in a separate model matrix X , which is optimized regarding the optimization criterion. The DOD aims at selecting configurations having the maximal possible distance to each other. Mathematically, this is mapped to a minimal determinant of the dispersion matrix $(X^T X)^{-1}$. However, identifying the optimal set of configurations is complex, because of the exponential size of the configuration space and the exponential number of valid subsets. To address this problem, different iterative algorithms leading to almost optimal designs have been proposed [21, 28]. The idea of these algorithms is to select a set of configurations first and to improve the selection iteratively by replacing configurations of the set with other configurations from the configuration space. Due to the exponential number of possible subsets, identifying the optimal set can consume considerable time. In some cases, the algorithms select only a locally optimal set of variants, resulting in a nearly optimal design. In our implementation, we use the k-exchange algorithm [28].

Random Design: Last, we use also a *Random Design (RD)* selecting a random set of configurations of the configuration space. This is rather straightforward, because we know the value range of the configuration options and constraints among the options. To guarantee reproducibility of the selection, we apply pseudo random selection with a fixed seed. We use this selection to validate whether systematic sampling is more beneficial than a random sampling.

3.2. Performance-Influence Models

To describe the performance influence of individual configuration options and interactions between them, we learn a *performance-influence model* Π . Mathematically, a performance-influence model is a projection from a configuration $c \in C$ to the performance of the corresponding system variant in \mathbb{R} , that is, $\Pi : C \rightarrow \mathbb{R}$. In the following, we give a simplified excerpt of a performance-influence model, which we learned to capture the time needed to perform a single multigrid iteration:

$$\overbrace{3.01}^{\pi_1} + \overbrace{1.54 \cdot \text{pre-smoothing}}^{\pi_2} + \overbrace{1.96 \cdot \text{post-smoothing}}^{\pi_3} - \overbrace{0.57 \cdot \text{post-smoothing} \cdot \text{V-cycle}}^{\pi_4} + \dots$$

In this model, we see that the configuration options *pre-smoothing*, *post-smoothing*, and *V-cycle* have an influence on performance. Also, we see a constant minimum that cannot be attributed to a specific option or interaction. This execution time is described in term π_1 with the constant value 3.01. According to the model, *pre-smoothing* and *post-smoothing* have an individual influence on performance (π_2 and π_3), and *post-smoothing* also interacts with the *V-cycle*, which is manifest as a product of two options (π_4). To evaluate the model for a given configuration, we replace the configuration options with their selected values. Binary options are replaced with 1 if the option is enabled and with 0 otherwise.

To learn a performance-influence model, we use a domain independent combination of *multivariate regression* and *forward feature selection* [29]: We use multivariate regression to determine the coefficients of the different terms of a given model (e.g., 3.01 for term π_1) and forward feature selection to select the terms that are iteratively added to the model [7]. The general idea of the learning procedure is to start with a simple model and to iteratively expand it to a more complex but also more accurate model. In each iteration of the learning procedure, we create a set of refined models and select the most accurate one as the basis for the next iteration. Each of the refined models contains all terms from the original model and one additional term representing the individual influence of one configuration option or one interaction between configuration options that are already found to be influential in the original model. Subsequently, the coefficients of the terms are computed based on the variants selected in the sampling process, using multivariate regression. Last, we compute the prediction error of the models created during this refinement process. After identifying the model with the smallest prediction error, we use it as basis for the next refinement step. This refinement process is performed until extending the model does not lead to a reduced prediction error. For more details of the learning procedure, we refer to previous work [7].

4. PERFORMANCE PREDICTION & ACCURACY

In our evaluation, we focus on identifying the performance contribution of individual configuration options on the performance of the configurable system. To this end, we are interested in the prediction accuracy of SPL Conqueror and in the effect of the different sampling strategies on accuracy. In this section, we describe the experimental setup to learn a performance-influence model with the goal to predict the time needed to perform a single iteration of our multigrid system. The goal of our experiments is to compare the sampling strategies used to learn the model regarding the accuracy of the derived model and the number of variants selected to learn the model. To this end, we conducted a series of experiments using the different sampling strategies, presented in Section 3.1. More to the point, the research question we address is:

RQ1: *What is the prediction accuracy and measurement effort of the different sampling strategies?*

4.1. Experiment Setup

Our configurable multigrid system of Section 2 solves a linear finite-element discretization of the Poisson equation on a two-dimensional grid. We measured a large number of variants of the multigrid system in a brute-force manner, to have a large test set (i.e. the ground truth), which we used to assess the prediction accuracy of the different sampling strategies and the overall learning procedure. Since the two interior angles α and β can have any value between 0 and 90 degree, we had to reduce the configuration space. To this end, we considered only variants, in which both interior angles are divisible by five. This reduced configuration space still contains 239 360 variants in total. We measured all of these variants on a MacBook Pro with a Core i5 2.7GHz and 8GB RAM, running OS X 10.10 (Yosemite).

In our experiments, we combined each of the sampling heuristics for binary options with each of the experimental designs (Section 3.1), and we learned a performance-influence model for each of the combinations. Then, we evaluated the prediction accuracy by comparing the performance prediction by the model against measured performance for all system variants (the ground truth). In particular, we are interested in the mean prediction error (\bar{e}) and the number of measurements selected by the sampling strategies ($|\mathcal{X}|$). We computed the mean prediction error for the models as follows:

$$\bar{e} = \frac{1}{n} \sum_{0 \leq i < n} \frac{|t_i^{\text{measured}} - t_i^{\text{predicted}}|}{t_i^{\text{measured}}}.$$

To control for the influence of randomness in RD, we performed a series of experiments with different sample seeds. Since the different sets of variants lead to different models in the learning process, we present only the mean prediction error of all models. Regarding the number of samples used by the RD and DOD, we use 49 configurations to be comparable to the PBD with the seed (49, 7), denoted as RD(49) and DOD(49), and 125 configuration to be comparable with the PBD with the seed (125,5), denoted as RD(125) and DOD(125), to be able to compare these designs.

4.2. Performance Prediction

Results. In Table I, we present the results of our experiments of predicting the runtime of one iteration of the multigrid cycle. In every cell, we show results for a combination of a binary and a numeric sampling strategy. To compute one of these models, we need less than a minute. We can predict the time needed to perform a single multigrid iteration irrespective of the sampling strategy with a prediction error of less than 13 %, after measuring a maximum of 1750 configurations (less than 1 % of all configurations). Regarding the binary sampling strategies, we observe that the PW heuristics leads to the most accurate performance-influence models. The mean prediction error of the models learned using the PW heuristics is between 4.1 % and 5.4 %, on average, while models learned with the OW or the NegOW heuristics have an error of, at least, 5.4 % and 6.6 %, on average. However, the superior prediction accuracy of the PW heuristics requires more than twice the number of multigrid system variants to be measured, as compared to the OW and NegOW

Table I. Results for different sampling strategies. For each combination of a binary sampling heuristics and an experimental design, we provide the mean prediction error (\bar{e}) of the models and the number of variants ($|X|$) used to learn the models.

	OW		PW		NegOW	
	\bar{e}	$ X $	\bar{e}	$ X $	\bar{e}	$ X $
RD(49)	7.2 %	300	5.4 %	700	7.7 %	300
RD(125)	5.6 %	750	4.8 %	1 750	7.2 %	750
PBD(9,3)	5.9 %	54	5.4 %	126	6.9 %	54
PBD(49,7)	5.4 %	294	4.4 %	686	6.6 %	294
PBD(125,5)	5.7 %	750	4.4 %	1 750	7.1 %	750
CCD	8.5 %	150	4.9 %	350	12.9 %	150
DOD(49)	5.7 %	300	4.4 %	700	6.9 %	300
DOD(125)	5.6 %	750	4.1 %	1 750	6.9 %	750
Brute Force	–	239 360	–	239 360	–	239 360

heuristics. For the experimental designs, we can see that using the D-Optimal Design with the larger number of configurations selected (125) leads to the most accurate performance-influence models in combination with the OW and the PW heuristics. Only for the NegOW heuristics, we see that another design (the Plackett-Burman Design with the seed (49,7)) leads to the most accurate model.

Discussion. In general, we observe a high prediction accuracy of our learned performance-influence models (see Table I). Some of the models are more accurate than others, though, and, for some of them, it is necessary to measure a considerably higher number of system variants. To identify the best tradeoff between prediction accuracy and measurement effort, we present the Pareto front of the sampling combinations in Figure 6 (marked with a black line[†]). We can see that almost all Pareto-optimal sampling strategies rely on the PW heuristics. Only the Pareto-optimal combination with the highest error rate (\bar{e}) uses another binary sampling strategy (OW heuristics).

The reduction of the error rate when using the PW heuristics indicates the presence of relevant interactions between binary configuration options. However, we have to keep in mind that although PW sampling leads to more accurate predictions, the differences in prediction accuracy are less than four percent between the OW and PW heuristics (using the same experimental design). This indicates that the corresponding interactions have only a small influence on the overall performance or they affect only a small number of configurations. In Table II, we see such a relationship of strong interactions between binary and numeric options and weak interactions between binary options. For example, the interactions described in the term π_8 (*F-cycle · Zebra-line*) can only be identified if the PW heuristics is used. Although the coefficient of the term indicates that the interaction has a strong influence on performance, it only affects a small number of configurations and, even for these configurations, its influence on performance is smaller than the influence of the interaction described in term π_{11} (*pre-smoothing · Zebra-line*), for example. The reason is that, when using the model to predict the performance of a configuration of the system, the coefficient of term π_{11} is multiplied with the value of the *pre-smoothing* option, which is in the range of 0 to 8, while the coefficient of term π_8 is multiplied only with 1 (if both options are selected).

In summary, for RQ1, we can conclude that we can learn performance-influence models with a mean error rate of less than 13 %. That is, we can predict performance with an accuracy of about 87 % after measuring less than 1 % of all system variants. To achieve the best prediction accuracy, a combination of the D-Optimal Design using as many configurations as the Plackett-Burman Design with the seed (125,5) and the PW heuristics should be used. If not solely prediction accuracy, but also the measurement effort is of interest, the Plackett-Burman Design using the seed (9,3) is a

[†]A solution x is Pareto optimal if and only if there is no other solution which dominates x in all dimensions [30].

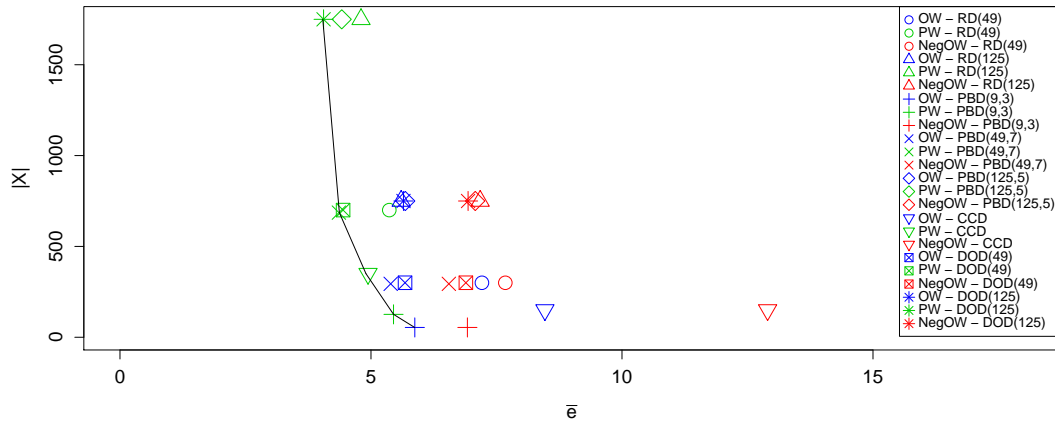


Figure 6. The tradeoff between the average prediction error rate (\bar{e}) and the number of configurations ($|X|$) for the different combinations of binary and numeric sampling strategies. The Pareto-optimal sampling combinations are highlighted by the Pareto front with a black line.

proper alternative. The selection of another design leads to an increase of prediction error of only 1.3 %, while requiring less than 10 % of measurements (PBD(125,5) vs. PBD(9,3)), leading to a substantial reduction of the measurement effort.

5. DISCOVERY OF DOMAIN KNOWLEDGE

One goal of using SPL Conqueror is to learn models that help domain experts in gathering new knowledge and validating existing knowledge about the influences of configuration options on the performance of their systems. In this section, we discuss the influences of the configuration options of our case study system that are predicated by theory and domain knowledge (see Section 2). We use these domain knowledge on the theoretical influence of configuration options to formulate four hypotheses on the time needed to perform one iteration. Then, we use our performance-influence model of Section 4.1 to examine whether the domain knowledge on the influences is represented in the model. This way, we test whether we can discover domain knowledge with our approach. We formulate the following research question:

RQ2: *Do our models capture existing domain knowledge accurately in terms of expected interactions and influences of configuration options?*

5.1. Discovering the Influence of Configuration Options

Due to different computational costs of the smoothers and the cycle types (see Section 2), it is clear that the choice of these algorithms will influence the CPU time necessary for an iteration of the multigrid system. The same holds for different numbers of pre-smoothing and post-smoothing steps, since the more smoothing the more expensive it is to perform an iteration of the algorithm, and the more time will be needed. The remaining configuration options affect the convergence of the multigrid algorithm, but not the time per iteration. For example, the geometry of the grid (characterized by the considered angles α and β) has a strong influence on the behavior of the multigrid algorithm, because an anisotropic grid implies that a multigrid based on a pointwise smoother will not converge satisfactorily. However, these parameters do not affect the time per iteration, as the number of unknowns to solve is the same for triangles with different angles.

H1: *All smoother types, all cycle types, and the number of pre-smoothing and post-smoothing steps have a measurable influence on performance.*

Table II. List of the terms of the performance-influence model learned using the PW heuristics in combination with DOD(125). We ordered the terms by the number of contributing configuration options and by the absolute strength of the influence (all terms with an absolute coefficient > 0.01).

Term	Coefficient	Options involved
π_1	2.51	
π_2	2.10	<i>post-smoothing</i>
π_3	1.93	<i>pre-smoothing</i>
π_4	0.92	<i>Jacobi</i>
π_5	0.72	<i>Three-color Gauss-Seidel</i>
π_6	0.08	<i>F-cycle</i>
π_7	2.23	<i>F-cycle · Lex-line</i>
π_8	2.12	<i>F-cycle · Zebra-line</i>
π_9	0.79	<i>pre-smoothing · Lex-line</i>
π_{10}	-0.78	<i>post-smoothing · Three-color Gauss-Seidel</i>
π_{11}	0.73	<i>pre-smoothing · Zebra-line</i>
π_{12}	-0.68	<i>post-smoothing · Jacobi</i>
π_{13}	0.58	<i>post-smoothing · F-cycle</i>
π_{14}	-0.56	<i>pre-smoothing · V-cycle</i>

In one iteration of the multigrid algorithm, a defined number of pre-smoothing and post-smoothing steps are performed at each level of the grid. Both during pre-smoothing and post-smoothing, all points of the domain are updated via a local stencil operation, which consumes most of the time of the iteration. This is because the points considered by an application smoother have to be loaded into the main memory. Although optimization techniques can be used to tune performance, we did not use them in our case study. Consequently, if several pre-smoothing or post-smoothing steps have to be performed, we complete the first iteration over the domain, also called sweep, before starting with the second one. Thus, the time needed to perform a single sweep is independent of whether it is performed after another sweep. If the time of a single sweep would be affected of being performed after another sweep, the performance influence of sweeps had to be modeled with a complex function within the performance-influence model to achieve a high accuracy. However, because we do not expect such an interaction, we formulate the following hypothesis:

H2: *The time needed for a single iteration correlates with the number of pre-smoothing and post-smoothing steps, where the number of pre-smoothing and post-smoothing steps affect performance linearly.*

In our case study, we consider two different cycle types: the V-cycle and the F-cycle. While the V-cycle goes down to the coarsest level only once, the F-cycle goes down to the coarsest grid multiple times and recursively to the next finer grid, as explained in Section 2. As a consequence, the pre-smoothing and post-smoothing sweeps are performed multiple times on a specific grid, leading to an increased runtime. Thus, we formulate for following hypothesis:

H3: *One iteration of a V-cycle needs less time than an iteration of the F-cycle for an otherwise identical configuration.*

As said in Section 2, we have two different types of smoothers in our multigrid system, and we can select exactly one in each configuration. Specifically, we have two pointwise smoothers, the *Jacobi* and the *Three-color Gauss-Seidel* smoother, and two linewise smoothers, the *Lex-line* and the *Zebra-line* smoother. While the pointwise smoothers update one unknown after the other, the linewise smoothers perform an update of a set of unknowns at the same time. Although a linewise smoother has to perform a smaller number of updates in a sweep of the computation domain, each individual update consumes more time than an update using a pointwise smoother. This additional

execution time in performing one update is so large that the linewise smoothers need more time than pointwise smoothers to perform one iteration of the multigrid solver. Thus, we formulate the following hypothesis:

H4: *An iteration needs less time when using a pointwise smoother than using a linewise smoother for an otherwise constant configuration.*

5.2. Discovering Performance Characteristics based on a Performance-Influence Model

To validate the hypotheses H1 to H4, we have a closer look at the terms of a performance-influence model that we learned in Section 4. In Table II, we present the coefficients and terms of the model we learned using the PW heuristics in combination with the D-Optimal Design, representing the most accurate model we learned (see Table I).

Hypothesis H1

If H1 is correct, pre-smoothing and post-smoothing as well as the different smoothers and cycle types should appear in the performance-influence model we learned. For the smoothers and cycle types, all of the corresponding options except for one should part of the model. Furthermore, none of the other options *alpha*, *beta*, and *relaxation parameter* should appear in the model.

As we can see in Table II, the options *alpha*, *beta*, and *relaxation parameter* are not in the model. Additionally, we can also see that all of the smoothers and all of the cycle types appear in, at least, one term of the model. Additionally, the configuration options *pre-smoothing* and *post-smoothing* are also contained in the model. Hence, we accept H1.

Hypothesis H2

Since a larger number of pre-smoothing and post-smoothing steps increases the time needed to perform a single iteration, our model should represent it with positive coefficients for terms, in which, at least, one of the two options (pre-smoothing or post-smoothing) appears. In Table II, we see that the options have individual influences, but also interact with other options. In the terms describing interactions between either pre-smoothing or post-smoothing and other options, we see some terms with a negative coefficient (term π_{10} , π_{12} , and π_{14}). However, the terms π_2 and π_3 have larger influences on performance than their interaction terms, which is in line with H2.

To further verify this hypothesis, we have to consider the underlying function describing the influence of the *pre-smoothing* and the *post-smoothing* option within the individual terms. This is because, based on domain knowledge, the influences of the numeric options should be linear. In performance-influence models, non-linear influences are represented with multiple occurrences of an option in a term (e.g., *pre-smoothing* · *pre-smoothing*). Since *pre-smoothing* and *post-smoothing* do not occur multiple times in a single term, they have only a linear influence on the performance. So, we accept hypothesis H2.

Hypothesis H3

To verify H3, it is necessary to consider all terms in which the *F-cycle* and the *V-cycle* occur. Based on term π_6 , we can state that the F-cycle needs more time than a V-cycle. This runtime overhead of the F-cycle is further increased by all other terms considering either *F-cycle* or *V-cycle* (see terms π_7 to π_8 and π_{13} to π_{14}). That is, all terms including the *F-cycle* option induce a performance decrease, and all terms considering the *V-cycle* induce a performance increase. So, we accept hypothesis H3.

Hypothesis H4

To verify H4, we have to consider all terms describing the influence of the different smoothers on performance, which appear in the terms π_4 , π_5 , and π_7 to π_{12} . Based on the terms π_4 and π_5 , we see that selecting either the *Jacobi* smoother or the *Three-color Gauss-Seidel* smoother results in an increased execution time, which contradicts our hypothesis. However, considering the terms π_{10} and π_{12} , we see that the overhead introduced by the two pointwise smoothers is decreased dramatically with an increasing number of post-smoothing steps. Additionally, with an increasing number of pre-smoothing steps, captured by the terms π_9 and π_{11} , the time needed for one iteration using a linewise smoother will increase. We can also witness a performance decrease when a linewise smoother is used in combination with the F-cycle, based on the terms π_7 and π_8 .

For almost all of the value combinations of the pre-smoothing and post-smoothing steps, it is easy to see that the time needed for an iteration is higher when a linewise smoother is used, compared to using a pointwise smoother for an otherwise constant configuration. However, if the *V-cycle* is used in combination with either one pre-smoothing and zero post-smoothing step or zero pre-smoothing and one post-smoothing steps, the predicted time needed for an iteration using the *Jacobi* smoother or the *Three-color Gauss-Seidel* smoother is greater than using one of the linewise smoothers. After having a deeper look at the configurations, which were predicted incorrectly, we saw only a small performance difference between them. So, in the end, we can only partially accept hypothesis H4, because the predicted time for four of the relevant value combinations are not in line with our hypothesis.

Summary

In summary, concerning RQ2, we were able to accept three out of four hypothesis based on the influences described in the performance-influence model that we learned using the PW heuristics in combination with the DOD(125). For H4, we can only partially accept it, because the predicted time for an iteration of four relevant value combinations is not in line with the hypothesis. We are able to confirm the validity of existing domain knowledge using SPL Conqueror.

6. THREATS TO VALIDITY

6.1. Internal Validity

In our experiments, we selected only a subset of the existing sampling strategies for binary and numeric options. Thus, we might have missed the optimal sampling strategy regarding measurement effort and prediction accuracy. However, our selection covers different properties and has been shown to be effective in prior work [7]. For the binary option heuristics, we use strategies focusing on identifying the individual influences of all configuration options and of pair-wise interactions between configuration options. For the numeric sampling, we use well known and successfully applied experimental designs.

Since we rely on the measured performance of configurations, the learning procedure may be subject to measurement bias. To reduce this threat, we measured the time needed to perform a single iteration multiple times and used the average in our learning procedure. To additionally quantify the influence of measurement bias on a performance-influence model, we performed an evaluation in a parallel line of research, where we know the influence of the configuration options. In these experiments, we added noise to the measurements and learned the models subsequently. Then, we used the models to predict the configurations. In this evaluation, we saw that our approach is robust against measurement bias, provided the randomness not greater than 12 % of the measured value.

6.2. External Validity

We have formulated four hypotheses on how configuration options influence the performance of our multigrid system and whether influences we learn match existing domain knowledge. It would be insightful to perform such an analysis also on other systems. For complex systems with a high number of different configuration options, though, validating domain knowledge based on

the identified influence might become challenging, because the performance-influence model might become very complex.

There is an inherent trade-off between internal and external validity in designing an experiment [31]. Here, we focused on internal validity (using only a single system and looking only at the time per iteration), because we want to make robust and reliable statements about whether a machine learning approach can be used in such a scenario, and we want to analyze in a controlled environment whether the performance characteristics match the expectations of domain experts. Only once this is verified, one can make a reliable leap forward and increase external validity [31].

In previous work, we predicted the time to solution of one multigrid system and the time needed for an iteration in three different multigrid systems [32]. Our prediction results are in line with the results presented in the previous work. Based on the predicted time for an iteration in a combination with LFA, the time to solution can be computed. Since the number of iterations can be computed analytically using LFA, we did not focus on predicting this number.

We demonstrated that we were able to predict the time needed for an iteration of all variants of the multigrid system with a high accuracy after measuring only a small number of configurations. However, we can not make the assumption that it is possible for all multigrid systems. That is, if the options of a given system have complex influences on performance and interact with each other in intricate ways, predicting the time needed for an iteration after measuring only a small number of system variants becomes challenging.

Additionally, we were able to discover almost all terms in depth that were identified during the learning process with domain knowledge, which is expected to hold for the entire domain of multigrid solvers. We view this experiment as a further piece of the puzzle in our endeavor of handling the complex performance-optimization process of multigrid systems.

7. RELATED WORK

There are many machine-learning techniques, such as support-vector machines [33], Bayesian networks [34], and evolutionary algorithms [35], that can be used to identify the optimal variant of a configurable system. However, these techniques fail in quantifying the influence of individual options and interactions between them. This way, it is impossible to determine critical configuration options and interactions, which is a key requirement in the high-performance computing domain.

In general, one has to differentiate between analytical-performance models and empirical-performance models. Analytical models are often created manually by experts, based on their knowledge on the application, while empirical models rely on measurements. Manually creating an analytical model is complicated and error-prone. The reason is that many aspects influence the design, and experts cannot control all of them. Furthermore, if there are unknown interactions an expert is not aware of, an analytical model will fail producing accurate predictions, because the designer of the model has to understand the whole application to produce an accurate analytical model [36]. For example, Gahvari et al. presents a model for an algebraic multigrid solver, in which they considered application and machine parameters [37]. Another example of an analytical model is presented by Kerbyson et al., who learned a model for a multidimensional hydrodynamics code [38].

Jain et al. use Random Forests as learning technique for learning empirical-performance models [39]. They consider properties of the application, such as the average number of bytes per link, as input for the learning algorithm. Guo et al. learn a global prediction model using Classification and Regression Trees [40, 41]. One drawback of these approaches is that neither the influences of configuration options nor their interactions are modeled explicitly. Moreover, they are limited to binary options. So, the influence of one configuration option on the performance of a system variant cannot be deduced.

Tallent et al. [42] use an approach to learn hierarchical models describing the performance of an application. To learn these models, the code of the system has to be annotated. With the annotations, the approach can be tailored to learn models for specific parts of the application. The hierarchy in the model is specified by the call hierarchy of the methods being annotated and the program annotations, in general. So, their technique learns hierarchical models, while we learn flat models. Another

important difference is that, we do not require the source code of the application, since we treat the multigrid system as a black box, which increases applicability and facilitates automation [8, 19].

Another approach to learn performance models is used by Mantis [43, 44]. They use it to predict the performance of Android apps depending on the program's input and environment. To learn a performance model, they execute an instrumented version of the program with different parameters. Then, they use linear regression to identify the influence of the parameters on performance. However, as their approach relies on an instrumentation of the program, they stick to a programming language and require the program as a white box.

To identify performance bottlenecks of high-performance applications, Calotoiu et al. [45] learn performance models after performing an automatic instrumentation of the source code using instrumentation tools such as Scalasca[‡]. However, currently they support learning models only with a single parameter.

8. CONCLUSION

Multigrid methods are among the most efficient iterative algorithms for solving sparse systems derived from the discretization of partial differential equations. However, the performance of multigrid systems strongly depends on the choice of the used algorithms and parameters. To identify the optimal combination of configuration options and to identify unknown performance characteristics and bottlenecks, we propose to learn performance-influence models using a combination of multivariate regression and forward feature selection. These models can be used to predict performance of all configurations of the system, help in understanding which options are most critical, and state in a human readable way, which options interact with each other. For this purpose, the influence of configuration options and their interactions are described explicitly in the model. In this paper, we evaluated the applicability of this approach to predict the time needed for a single iteration of a configurable multigrid system working on triangular grids. In a set of experiments, we determined the influence of different structured sampling strategies on the prediction accuracy of the produced models. We found that the time per iteration needed for all multigrid system variants can be predicted with an accuracy of about 87 %, on average, after measuring less than 1 % of all system variants. The best combination of sampling strategies (Pair-wise heuristics in combination with the D-Optimal Design) used in our experiments leads to a prediction accuracy of almost 96 %, after measuring only 0.1 % of all variants. Additionally, we evaluated whether we can discover interactions between options and individual influences known by domain experts using SPL Conqueror's approach. We were able to identify almost all influences and interactions stated by two experts and, in this way, confirm existing domain knowledge. This opens up the possibility of providing performance-influence models to domain experts to increase their understanding of a given system.

ACKNOWLEDGEMENTS

This work is supported by the German Research Foundation (DFG), as part of the Priority Programme 1648 "Software for Exascale Computing", under contract AP 206/7. Sven Apel's work is also supported by the DFG under the contracts AP 206/4 and AP 206/6. The work of Francisco J. Gaspar and Carmen Rodrigo is supported in part by the Spanish project FEDER /MCYT MTM2013-40842-P and the Diputación General de Aragón (Grupo consolidado PDIE). This has been cooperative work initiated at the Dagstuhl Seminar *Advanced Stencil-Code Engineering* in April 2015.

REFERENCES

1. Brandt A. Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation* 1977; **31**(138):333–390.

[‡]<http://www.scalasca.org/>

2. Hackbusch W. *Multi-Grid Methods and Applications*. Springer Series in Computational Mathematics, Springer, 1985.
3. Trottenberg U, Oosterlee CW, Schüller A. *Multigrid*. Academic Press, 2001.
4. Wesseling P. *An Introduction to Multigrid Methods*. John Wiley, 1992.
5. Stüben K, Trottenberg U. Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications. *Multigrid Methods, Lecture Notes in Mathematics*, vol. 960. Springer, 1982; 1–176.
6. Wienands R, Joppich W. *Practical Fourier Analysis for Multigrid Methods*. Chapman and Hall/CRC Press, 2005.
7. Siegmund N, Grebhahn A, Apel S, Kästner C. Performance-influence Models for Highly Configurable Systems. *Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ESEC/FSE, ACM, 2015; 284–294.
8. Siegmund N, Kolesnikov SS, Kästner C, Apel S, Batory D, Rosenmüller M, Saake G. Predicting Performance via Automated Feature-interaction Detection. *Proceedings of the International Conference on Software Engineering*, ICSE, IEEE, 2012; 167–177.
9. Grebhahn A, Kuckuk S, Schmitt C, Köstler H, Siegmund N, Apel S, Hannig F, Teich J. Experiments on Optimizing the Performance of Stencil Codes with SPL Conqueror. *Parallel Processing Letters* 2014; **24**(3).
10. Bergen BK, Gradl T, Hülsemann F, Rude U. A Massively Parallel Multigrid Method for Finite Elements. *Computing in Science Engineering* 2006; **8**(6):56–62.
11. Bergen BK, Hülsemann F. Hierarchical Hybrid Grids: Data Structures and Core Algorithms for Multigrid. *Numerical Linear Algebra with Applications* 2004; **11**(2-3):279–291.
12. Rodrigo C, Gaspar FJ, Lisboa FJ. *Geometric multigrid Methods on Triangular Grids: Application to Semi-Structured Meshes*. Lambert Academic Publishing, 2012.
13. Vanka SP. Block-implicit Multigrid Solution of Navier-Stokes Equations in Primitive Variables. *Journal of Computational Physics* 1986; **65**(1):138–158.
14. Rodrigo C, Gaspar FJ, Lisboa FJ. On a Local Fourier Analysis for Overlapping Block Smoothers on Triangular Grids. *Applied Numerical Mathematics* 2016; **105**:96–111.
15. Nguyen A, Satish N, Chhugani J, Kim C, Dubey P. 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC, IEEE, 2010; 1–13.
16. Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Technical Report CMU/SEI-90-TR-2*, CMU, SEI 1990.
17. Montgomery DC. *Design and Analysis of Experiments*. John Wiley & Sons, 2006.
18. Myers RH, Montgomery DC. *Response Surface Methodology: Process and Product in Optimization Using Designed Experiments*. John Wiley & Sons, Inc., 1995.
19. Siegmund N. Measuring and Predicting Non-Functional Properties of Customizable Programs. Dissertation, University of Magdeburg, Germany 2012.
20. Ferreira S, Bruns R, Ferreira H, Matos G, David J, Brando G, da Silva E, Portugal L, dos Reis P, Souza A, et al.. Box-Behnken Design: An Alternative for the Optimization of Analytical Methods. *Analytica Chimica Acta* 2007; **597**(2):179–186.
21. Fedorov V. *Theory of Optimal Experiments*. Probability and Mathematical Statistics, Academic Press, 1972.
22. Bergstra J, Pinto N, Cox D. Machine Learning for Predictive Auto-tuning with Boosted Regression Trees. *Innovative Parallel Computing (InPar)*, IEEE, 2012; 1–9.
23. Ganapathi A, Datta K, Fox A, Patterson D. A Case for Machine Learning to Optimize Multicore Performance. *Proceedings of the First USENIX Conference on Hot Topics in Parallelism*, HotPar, USENIX Association, 2009; 1–6.
24. Box GEP, Wilson KB. On the Experimental Attainment of Optimum Conditions. *Journal of the Royal Statistical Society. Series B (Methodological)* 1951; **13**(1):pp. 1–45.
25. Khuri AI, Mukhopadhyay S. Response surface methodology. *Wiley Interdisciplinary Reviews: Computational Statistics* 2010; **2**(2):128–149.
26. Plackett RL, Burman JP. The Design of Optimum Multifactorial Experiments. *Biometrika* 1946; **33**(4):pp. 305–325.
27. Smith K. On the Standard Deviations of Adjusted and Interpolated Values of an Observed Polynomial Function and its Constants and the Guidance they give Towards a Proper Choice of the Distribution of Observations. *Biometrika* 1918; **12**(1/2):pp. 1–85.
28. Johnson ME, Nachtshiem CJ. Some Guidelines for Constructing Exact D-Optimal Designs on Convex Design Spaces. *Technometrics* 1983; **25**(3):271–277.
29. Molina LC, Belanche L, Nebot A. Feature Selection Algorithms: A Survey and Experimental Evaluation. *Proceedings of the International Conference on Data Mining*, ICDM, IEEE, 2002; 306–313.
30. Veldhuizen DAV, Lamont GB. Evolutionary Computation and Convergence to a Pareto Front. *Stanford University, California*, Morgan Kaufmann, 1998; 221–228.
31. Siegmund J, Siegmund N, Apel S. Views on Internal and External Validity in Empirical Software Engineering. *Proceedings of the International Conference on Software Engineering (ICSE)*, IEEE, 2015; 9–19.
32. Grebhahn A, Kuckuk S, Schmitt C, Köstler H, Siegmund N, Apel S, Hannig F, Teich J. Experiments on Optimizing the Performance of Stencil Codes with SPL Conqueror. *Parallel Processing Letters* 2014; **24**(3):Article 1441 001, 19 pages.
33. Steinwart I, Christmann A. *Support Vector Machines*. Springer, 2008.
34. Jensen F, Nielsen T. *Bayesian Networks and Decision Graphs*. 2nd edn., Springer, 2007.
35. Simon D. *Evolutionary Optimization Algorithms*. John Wiley & Sons, 2013.
36. Ipek E, de Supinski BR, Schulz M, McKee SA. An Approach to Performance Prediction for Parallel Applications. *Euro-Par Parallel Processing: International Euro-Par Conference*, Springer, 2005; 196–205.
37. Gahvari H, Baker AH, Schulz M, Yang UM, Jordan KE, Gropp W. Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms. *Proceedings of the International Conference on Supercomputing*, ICS, ACM, 2011; 172–181.

38. Kerbyson DJ, Alme HJ, Hoisie A, Petrini F, Wasserman HJ, Gittings M. Predictive Performance and Scalability Modeling of a Large-scale Application. *Proceedings of the Conference on Supercomputing*, SC, ACM, 2001; 37.
39. Jain N, Bhatele A, Robson MP, Gamblin T, Kale LV. Predicting Application Performance Using Supervised Learning on Communication Features. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC, ACM, 2013; 95:1–12.
40. Guo J, Czarniecki K, Apel S, Siegmund N, Wasowski A. Variability-Aware Performance Prediction: A Statistical Learning Approach. *Proceedings of the International Conference on Automated Software Engineering*, ASE, ACM, 2013; 301–311.
41. Sarkar A, Guo J, Siegmund N, Apel S, Czarniecki K. Cost-Efficient Sampling for Performance Prediction of Configurable Systems. *Proceedings of the International Conference on Automated Software Engineering*, ASE, 2015; 342–352.
42. Tallent NR, Hoisie A. Palm: Easing the Burden of Analytical Performance Modeling. *Proceedings of the International Conference on Supercomputing*, ICS, ACM, 2014; 221–230.
43. Chun BG, Huang L, Lee S, Maniatis P, Naik M. Mantis: Predicting System Performance through Program Analysis and Modeling. *Computing Research Repository* 2010; .
44. Kwon Y, Lee S, Yi H, Kwon D, Yang S, Chun BG, Huang L, Maniatis P, Naik M, Paek Y. Mantis: Automatic Performance Prediction for Smartphone Applications. *Proceedings of the USENIX Conference on Annual Technical Conference*, USENIX ATC, USENIX Association, 2013; 297–308.
45. Calotoiu A, Hoefler T, Poke M, Wolf F. Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC, ACM, 2013; 45:1–45:12.