Cost-Efficient Sampling for Performance Prediction of Configurable Systems

Atri Sarkar , Jianmei Guo
University of Waterloo, Canada
Email: a9sarkar,gjm@gsd.uwaterloo.ca

Norbert Siegmund , Sven Apel
University of Passau, Germany
Email: norbert.siegmund,apel@uni-passau.de

Krzysztof Czarnecki
University of Waterloo, Canada
Email: kczarnec@gsd.uwaterloo.ca

***Abstract*—A key challenge of the development and maintenance of configurable systems is to predict the performance of individual system variants based on the features selected. It is usually infeasible to measure the performance of all possible variants, due to feature combinatorics. Previous approaches predict performance based on small samples of measured variants, but it is still open how to dynamically determine an ideal sample that balances prediction accuracy and measurement effort. In this paper, we adapt two widely-used sampling strategies for performance prediction to the domain of configurable systems and evaluate them in terms of sampling cost, which considers prediction accuracy and measurement effort simultaneously. To generate an initial sample, we introduce a new heuristic based on feature frequencies and compare it to a traditional method based on t-way feature coverage. We conduct experiments on six real-world systems and provide guidelines for stakeholders to predict performance by sampling.**

I. Introduction

Selecting configuration options or *features* allows stakeholders to customize a configurable system in various ways, giving rise to a multitude of system *variants* or *configurations*. Each feature can have an effect on the functional and non-functional properties (e.g., performance and cost) of the system.

Analyzing performance is a critical step in the evaluation of software quality. It helps developers in judging how far the software matches the performance requirements. However, especially in the case of configurable systems, this task is not trivial. Due to feature combinatorics, the number of variants of a configurable system often increases exponentially with the number of features the system provides. Take SQLite, one of the most widely used database engine, for example: only 39 features give rise to over 3 million variants [17]. Due to an often complex benchmarking process, measuring even a single system variant may be costly. In the light of these problems, recent approaches predict performance based on a small sample of measured variants. Siegmund et al. [17] proposed a measurement-based prediction approach that detects performance-relevant feature interactions using specific sampling heuristics that meet different feature-coverage criteria. Guo et al. [5] used a statistical learning technique to infer performance prediction rules based on random samples. These approaches depend on certain sampling strategies (i.e., selecting a specific set of configurations to be measured) that provide fixed termination criteria for the sampling process to achieve an acceptable prediction accuracy (e.g., $90\,\%$). However, these sampling strategies cannot dynamically adjust the sampling process, including termination, in terms of the specific characteristics of a given system, so they may measure more variants than necessary. For example, as reported by Guo et al. [5], for some systems an acceptable prediction accuracy can be achieved using very small samples.

In this work, we aim at a smart sampling strategy that dynamically determines a "good" sample for a given system. A sample is good if it is small enough to decrease the measurement effort and large enough to increase the prediction accuracy at the same time. To quantify the goodness of a sample, we introduce a composite model of sampling cost [18], which considers the measurement effort and prediction accuracy simultaneously. We investigate two sampling strategies widely used in data mining: a classical technique, called *progressive* sampling [15], and a state-of-the-art technique, called *projective* sampling [10]. We conduct experiments on six real-world configurable systems and compare the two sampling strategies in terms of sampling cost. Furthermore, we enhance projective sampling by incorporating two heuristics for initial sample generation: a heuristic based on t-way (e.g., 2-way and 3-way) feature coverage, as commonly used in combinatorial testing [8], and a novel heuristic based on feature frequencies. We empirically compare the performance of four projective functions used in projective sampling on our six real-world configurable systems.

In summary, we make the following contributions:

- We adapt progressive and projective sampling strategies to performance prediction of configurable systems, and we compare them in terms of the sampling cost, balancing prediction accuracy and measurement effort.
- We propose a heuristic based on feature frequencies to guide the initial sample generation of projective sampling. We compare it to a common heuristic based on t-way feature coverage.
- Empirical results on six configurable systems demonstrate that projective sampling using the feature-frequency heuristic is cost-efficient. That is, it hits a sweet spot between prediction accuracy and measurement effort. Moreover, we empirically identify the best projective function for projective sampling in our experimental setup.

The implementation and all experimental data are available at https://github.com/atrisarkar/ces

II. A Bird's Eye View of Performance Prediction by Sampling

Figure 1 illustrates the general process of performance prediction by sampling. It starts with an initial sample of

measured configurations, which are used to build the prediction model. A good initial sample significantly reduces the iterations of the entire prediction process. State-of-the-art approaches fix the size of the initial sample to the number of features or potential feature interactions of a system [5], [17]. However, such a strategy might not be the optimal one, as the number of features (and their interactions) can be high and, at the same time, an acceptable prediction accuracy might be achieved using a substantially smaller set of measured configurations. In our approach, we use a combination of random sampling and feature-coverage heuristics to dynamically build the initial sample. In particular, we propose a feature-frequency heuristic for the initial sample generation, and we compare it to a traditional technique based on t-way feature coverage, commonly used in combinatorial testing [8]. Then, we build prediction models using a statistical learning technique, called Classification and Regression Tree (CART), which has been demonstrated to be fast and accurate for performance prediction of configurable systems [5].

In previous work, prediction accuracy was the main evaluation metric used to estimate the utility of the prediction models [5], [17], [19]. In this paper, we put forward the idea that, since there is a cost involved in measuring the sample of configurations for building the prediction model, a metric must consider both measurement effort and prediction accuracy to comprehensively evaluate the prediction model. To this end, we propose sampling cost as the evaluation metric that quantifies the utility of a sampling strategy by taking not only prediction accuracy into account, but also measurement effort. We will present the cost model in Section IV.

Most prediction models, including the ones used in our study are built in an iterative manner. The performance engineer measures a few configurations of a system (i.e., the sampling set), which are divided into a *training* set and a *testing* set. The training set is used to build a prediction model. This model is then evaluated using an evaluation metric on the testing set. If the value of the metric for this model falls within an acceptable range, the process stops, otherwise more measurements are added to the sample for refining the prediction model. This iterative process can be illustrated in the form of a learning curve [15], as shown in Figure 2.

The learning curve of Figure 2 relates accuracy to the size of the training set. The horizontal axis represents the size of the training set used to build the prediction model; the vertical axis shows the accuracy of the corresponding model calculated using the testing set. An ideal learning curve has three distinct regions. The first region has a steep incline, indicating rapid increase in accuracy when adding sample points. The second (optional) region has a gradual increase in prediction accuracy. Finally, the third region saturates in a plateau, where adding further sample points will not result in significant accuracy improvements anymore. In traditional progressive sampling, the smallest sample size for which the prediction model returns acceptable values in terms of the evaluation metric is called the *optimal* sample size.

Our goal is to design a smart sampling strategy that reaches the optimal sample size as fast as possible in terms of sampling cost. To this end, we define a stopping criterion based on the sampling cost. Moreover, we investigate two sampling strategies widely used in data mining: progressive sampling

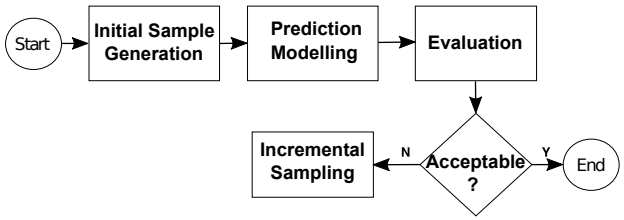


Figure 1: General process of performance prediction by sampling

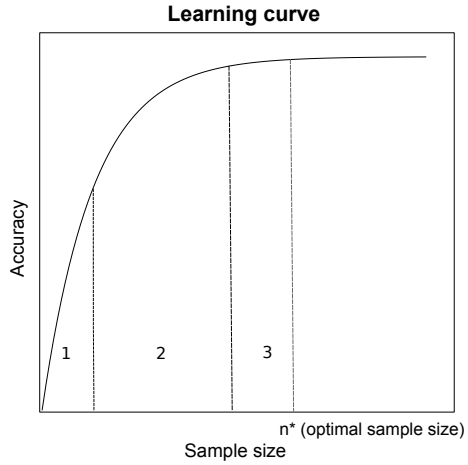


Figure 2: Regions of the learning curve in dependence of the sample size: (1) Steep incline; (2) Gradual incline; (3) Plateau; n^*: marks the optimal sample size

[15] and projective sampling [10], which will be explained in Section V.

III. Definitions and Running Example

We represent all features of a configurable software system as a set X of binary decision variables. If a feature is selected in a configuration, then the corresponding variable x is equal to 1, and 0 otherwise. We denote the number of all features of a system as N, that is, $X = \{x_1, x_2, ..., x_N\}$. We represent each configuration of a system as an N-tuple, assigning value 1 or 0 to each variable in X. We denote all valid configurations of a configurable system as set $\mathbf{X}$.

The learning curve represents a mapping between the training-set size and the corresponding accuracy. The pair $\lambda_n = (n, \epsilon_n)$ represents a point in the learning curve, where n is the size of the training set and ϵ_n denotes the prediction error of a model built with a training set of size n. Assuming $S_n \subseteq \mathbf{X}$ as the training set (of size n), since we reuse samples from previous iterations, the sample set S_n has only one additional new configuration as compared to set S_{n-1}.

For example, one of our subject systems of Section VII, Apache, has a total of 9 features, and the total number of all valid configurations is 192. We follow the strategy used by Guo et. al. [5] to generate the valid configurations. Table I shows the learning-curve points for Apache measured at an interval of 10 configurations. At each step, 10 additional configurations are measured and added to the training set. The accuracy of

Table I: Learning curve points (λ_n) for Apache (n : sample set size, ϵ_n : relative error %)

n	ϵ_n	n	ϵ_n
10	19.26	60	7.52
20	11.12	70	7.44
30	8.62	80	7.25
40	8.23	90	7.17
50	7.76		

the prediction model (CART in our case) is calculated at each step based on a testing set of a size equal to the training set, randomly sampled from the set of configurations not measured so far.

IV. Cost Model

Typically, performance prediction models are evaluated on the basis of their prediction accuracy. It is also common knowledge, and apparent from the learning curve (Figure 2), that usually a larger training set results in higher prediction accuracy. However, a large training set is often infeasible in terms of measurement effort. Thus, any performance prediction model built for this purpose should be evaluated not only in terms of prediction accuracy, but also in terms of measurement cost involved in building the training and testing sets. Weiss and Tian [18] introduced the concept of *utility-based* sampling, in which they combined the above two factors in the form of a composite cost model. We have modified the original cost model of Weiss and Tian [18] to include the cost incurred in measuring the testing set along with the training set:

$$\begin{aligned} TotalCost &= \text{Cost}_{Measurement(Training)} \\ &+ \text{Cost}_{Measurement(Testing)} \\ &+ \text{Cost}_{ModelBuilding} \\ &+ \text{Cost}_{PredictionError} \end{aligned} \quad (1)$$

We can simplify the above cost model by ignoring the cost incurred in building a performance prediction model, as for CART, which is used in our approach, this cost is computationally insignificant, compared to the other cost factors. Moreover, we use a 50:50 split between the training and testing sets in our sampling strategy, (i.e., the size of training set is the same as the size of testing set). Therefore, given a training and testing set of size n each, we have the following cost function of n:

$$TotalCost(n) = 2n + \epsilon_n \cdot |S| \cdot R \quad (2)$$

where $2n$ is the number of sample configurations in the training and testing sets, ϵ_n is the prediction error of this performance prediction model built with the n configurations, |S| is the *score set* (i.e., the number of configurations whose performance value will be predicted by the model), and R is a tuning parameter that controls the ratio of the cost incurred due to the prediction error to the cost of acquiring training samples. For example, R = 0.5 means that the cost to measure a configuration for the training sample is twice the cost arising from an incorrect prediction of the performance of a configuration. The actual value of R is problem specific and shall be set by domain experts. One way to do this is by basing the measurement effort and prediction cost in the same unit. For instance, companies often use *man-hours* as the unit of choice to quantify investment efforts [2]. In such scenarios, the value of R can be derived by calculating the ratio of investment required in *man-hours* for the two factors.

An interesting characteristic of the cost function is that, for a well-behaved, monotonically non-decreasing learning curve, the cost function is convex (see Figure 3b) [10]. This characteristic enables us to easily find the global minimum of the cost with respect to the sample size (see Section V).

V. Progressive and Projective Sampling

In this section, we discuss two sampling strategies for building the training set independent of the prediction model. We assume the learning curve of the prediction model to be well behaved, that is, monotonically non-decreasing. We argue that this is a reasonable assumption based on evidence from previous work [13] [4] and on our experience studying the data from our six subject systems. There are local variations, though, where additional sample points sometimes result in a reduced prediction accuracy, and we address them by using a moving-average smoothing technique on the data points [6]. Still, for a wide range of sample sizes, we see the learning curve to be monotonically non-decreasing in our experiments.

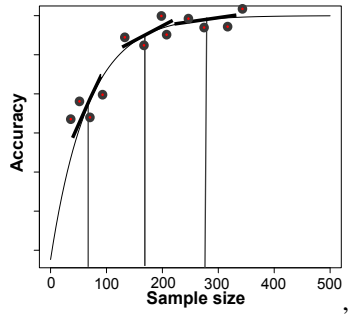
A. Progressive Sampling

Progressive sampling is a popular sampling strategy that has been used for a variety of learning models [15] [11]. The central idea is to use a sampling schedule $n_0, n_1, n_2, n_3, ..., n_k$, where each n_i is an integer that specifies the size of the sample set that is used to build a performance prediction model at iteration i. Based on how the size of the sample set in each iteration is calculated, progressive-sampling strategies can be divided into two kinds [7]. The first one is *arithmetic* progressive sampling, where in each iteration, we add a constant number of additional sample points to the training set according to the equation $n_i = n_0 + i * a$. The second kind is *geometric* progressive sampling, where the sample-set sizes are built in geometric progression, according to the equation $n_i = n_0 * a^i$. The parameter a is a constant that defines how fast we increase the size of the sample set.

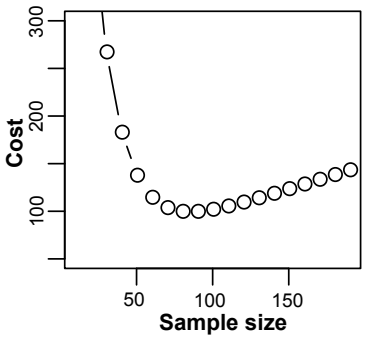
The primary difference between arithmetic and geometric progressive sampling is the number of sample points we add in each iteration, and using geometric progressive sampling, we can hit the plateau region in the learning curve in fewer iterations [15]. This is an advantage in cases where building the model is an expensive process. For example, as listed in Table I, if we set an acceptable prediction accuracy of the system to 92%, we can observe that the optimal sample size for that accuracy is 50. If we start with an initial training set size of 10 and add 10 more configurations in each iteration reusing samples from preceding iterations, the arithmetic sampling scheme needs

$$\begin{aligned} 10_{(iteraton_0)} + 10_{(iteraton_1)} + 10_{(iteraton_2)} \\ +10_{(iteraton_3)} + 10_{(iteraton_4)} \end{aligned}$$

that is, 50 measurements with the model being built 5 times, once in each iteration. For geometric progressive sampling, if we start with the same 10 measurements and a minimum common ratio of 2, the number of measurements needed is



(a) Gradient-based



(b) Cost minimization

Figure 3: Two stopping criteria for progressive sampling

$$10_{(iteraton_0)} + 10_{(iteraton_1)} + 20_{(iteraton_2)} + 40_{(iteraton_3)}$$

80, which is 30 more than that of arithmetic progressive sampling, although the model is built only 4 times. For performance prediction of configurable systems, the cost of acquiring training samples by measuring system configurations usually overrides the cost of building the performance prediction model (CART in our case), thus we consider only arithmetic progressive sampling in what follows.

Stopping Criteria: For both arithmetic and geometric progressive sampling, we need to decide when to stop sampling more configurations for measurement. This is a critical step that needs to be performed in every iteration and to check whether the built prediction model has converged to an acceptable prediction accuracy. Next, we discuss two common stopping criteria.

1) Gradient-Based: Linear Regression with Local Sampling (LRLS) uses the gradient of the learning curve to detect convergence [12]. Using this method, we build additional models in the local neighborhood of n_i and determine their accuracy. We use these additional sample points to fit a linear regression line and calculate the gradient, as illustrated in Figure 3a. If the gradient is less than a certain threshold, we stop sampling and designate the sample size used in that iteration to be the final sample size. However, this naive approach does not take the factor cost into account. Furthermore, it has the drawback of possibly getting stuck in a local plateau of the learning curve.

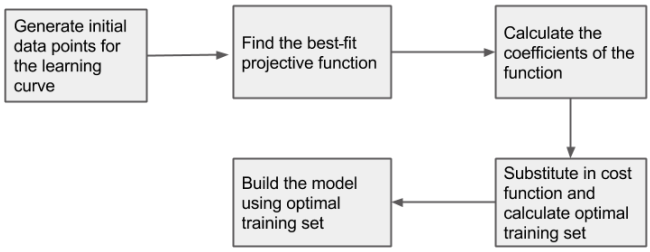


Figure 4: Overview of projective sampling

2) Cost Minimization: As the measurement cost is of primary importance in our case, LRLS-based convergence detection is not well suited. A more pragmatic approach is to use the cost function of Equation 2 to detect convergence of the learning curve [18]. Using this method, for each iteration, we calculate the total cost of using the model with n_i sample measurements. This problem now translates into an *optimization* problem, where the objective is to find n_i that minimizes the cost function. Since the cost function is a convex function for well-behaved learning curves (cf. Figure 3b), we can calculate a sample size that minimizes the total cost when we observe the first increase in total cost. Thus, if the first increase in total cost is observed in iteration i, for a sample size of n_i, then the optimal sample size guaranteeing minimum cost is n_{i-1}. As our evaluation of prediction models is based on cost, we use the cost-minimization stopping criterion for progressive sampling, when comparing it to projective sampling, which we describe next.

B. Projective Sampling

One of the weaknesses of progressive sampling is that the prediction model can converge only after several iterations with a large sample size, and there is no way to determine this unless we have actually built the model. This defeats the entire purpose of the prediction procedure, as there is a risk that, even after spending resources in running benchmark tests for several configurations, the cost and accuracy of the model at the point of convergence might not be acceptable for the user. *Projective* sampling addresses this problem by approximating the learning curve using a minimal set of initial sample points [10], thus providing stakeholders with an estimate of the cost projection of the entire prediction process, thus helping them to decide whether to adopt the prediction model for their system. We define the cost incurred in generating this projected learning curve as the *decision cost*.

In Figure 4, we provide an overview of the steps involved in projective sampling. Projective sampling starts with an empty training set, adding a constant number n_δ of configurations to the training set in each iteration. There is no minimum constraint to the value of the constant n_δ apart from being greater than 0, and we have seen from our experiments that even a value of 1 can give good results. In each iteration, we build the model and calculate the accuracy, this way generating a sample point for the learning curve. The size of these initial sample points or the number of iterations the algorithm should run, varies with the number of features of the system, and we

Table II: Projective functions of learning curves

Name	Equation	Optimal Sample Size
Logarithmic	$err(n) = a + b.log(n)$	$n^* = -(R \cdot \|S\| \cdot b)/2$
Weiss and Tian	$err(n) = a + bn/(n+1)$	$n^* = \sqrt{(-R \cdot \|S\| \cdot b)/2}$
Power Law	$err(n) = an^b$	$n^* = (\frac{-2}{R \cdot \|S\| \cdot a \cdot b})^{\frac{1}{b-1}}$
Exponential	$err(n) = ab^n$	$n^* = \log_b\left(\frac{-2}{R \cdot \|S\| \cdot a \cdot \ln b}\right)$

use a novel feature-frequency heuristic for sample generation (Section VI) to build this set.

This initial set of sample points represents a partial learning curve. In the next step, we search for a best-fit function that can extrapolate the remaining learning curve. Learning curves for black-box performance prediction methods, such as CART, exhibit usually good correlation with one of four different types of projective functions [4] [10], as shown in Table II. Parameters a and b are the co-efficients of the projective functions, and $|S|$ and R are the score set and the cost ratio as defined in Equation 2. Using the information from the initial sample points, we follow the approach proposed by Last [10] in selecting the projective learning function that exhibits highest correlation with the sample points. Once we have determined the best-fit function that can approximate the learning curve accurately, we can calculate the coefficients of the projected function using the least-squares method [14].

The optimal sample size can be defined as the size of the training sample that minimizes the cost function of Equation 2, ensuring the most optimal tradeoff between measurement cost and prediction accuracy. In projective sampling, we have knowledge about the projected learning curve, which gives us an estimate of the prediction error as a function of the sample size. If we substitute the value of err in the cost equation, we can calculate the total cost of the prediction process as a function of the sample size n. Figure 3b shows an example of cost versus sample size for Apache. We can see from the figure that the cost function is convex, which holds for any monotonically non-decreasing learning curve [10]. Since all the candidate projective learning curves follow this property, the first derivative $d(Cost(n))/dn = 0$ of the cost function is a global minimum. The solution of this equation gives us the sample size n^* that guarantees minimum cost. Table II shows the values of n^* (cf. Fig. 2) for the four different learning-curve equations in terms of the cost-equation variables of Equation 2. In the following sections, we use n^* to represent the optimal sample size that minimizes the cost function of Equation 2.

VI. Initial Sample Generation

An important step in projective sampling is the generation of the initial sample points that are used to project the learning curve. Given a sample point λ_i of a learning curve, our objective is to sample a set $\Lambda_\delta = \{\lambda_1, \lambda_2, ..., \lambda_\delta\}$, such that Λ_δ can be used to generate the projected learning curve accurately.

There are two key aspects that need to be considered when designing sampling strategies to build this set of sample points. First, one of the advantages of projective sampling is that it gives stakeholders an estimate of the optimal sample size n^* with minimal investment in terms of measurement cost. However, to generate this projected curve, they need to measure δ configurations and build the initial sample set. As defined in the last section, the cost of measuring these configurations is the decision cost. Thus, the size of the set or the value of δ is critical. To make the sampling strategy cost efficient, the value of δ should be less than n^*, otherwise we would end up measuring more than the optimal number of configurations. Second, the accuracy of the projected learning curve matters. It is important that the initial sample set should be able to produce a projected learning curve that approximates the real learning curve accurately. Since the size of this initial set Λ_δ needs to be kept small, there is a high probability that a suboptimal strategy in generating these initial sample points will result in a projective function that is not an accurate reflection of the learning behavior of the model. This can have a cascading effect throughout the entire sampling process and result in a value of sample size (n^*) that is not optimal in terms of cost.

To generate the initial sample set for projective sampling, we propose a heuristic based on feature frequencies: For a configurable system, users can select or deselect features, and there is typically a relation between features and the overall performance of the system in terms of throughput or execution time [5]. As a consequence, the performance of the system may vary substantially depending on whether a certain feature is selected or not. For example, in a Web application, the system might slow down if logging is enabled, because it takes extra time to perform I/O operations involved in directing messages to a log file. Thus, the first requirement of a good representative sample is that the sample configurations in our initial sample set Λ_δ should have each feature selected, at least, once. Also, since feature deselection can have an influence on the performance values too, it is important that the sample configurations in Λ_δ should have each feature deselected, at least, once. These constraints on feature selection and deselection apply only to optional features; mandatory features are active for all the configurations in the sample set.

In terms of our problem definition (Section III), the set S_δ represents the set of δ distinct configurations in the initial sample set Λ_δ. For a given feature i, the frequency of this feature is defined by the following equation:

$$1 \leq \sum_{j=1}^{\delta} x_i(j) < \delta \tag{3}$$

where $x_i \in \{1, 0\}$ is a Boolean variable representing a feature i being selected or deselected for a configuration j. Table III provides an example set S_δ of sample configurations and their corresponding feature selections.

The initial sample set Λ_δ should exhibit a good correlation with the projected learning curve. This is because the projected learning curve generated using the sample set is indicative of the learning progress of the prediction model with respect to the sample size. If the correlation is low, the projected curves might not model the learning behavior of the system accurately. Prior work has shown that random sampling can be used for accurate performance prediction of configurable systems [5]. In our approach, we use a combination of incremental random

Table III: Coverage of selected features ($x_i = 1$) and deselected feature ($x_i = 0$)

Conf.	Features							
	x_1	x_2	x_3	..	x_i	..	..	x_N
1	0	1	1	0	0	0	1	1
2	0	0	1	1	1	0	0	0
3	1	1	0	1	0	1	1	1
4	0	1	0	1	0	1	0	0
5	1	1	0	0	0	1	0	1
6	0	0	0	1	1	1	0	0
7	1	1	0	1	0	0	0	1
8	1	0	0	0	1	0	0	1

Table IV: Feature frequency table ***T***. Row 1 : frequency of selected features. Row 2: frequency of deselected features

	Features							
	x_1	x_2	x_3	..	x_i	..	..	x_N
selected	4	5	2	..	3	..	..	5
deselected	4	3	6	..	5	..	..	3

sampling and a feature-frequency heuristic to build the initial sample. To keep track of feature frequencies, our algorithm uses a $2 \times N$ feature-frequency table ***T*** (Table IV), where the columns of the table represent N optional features of the system. Each cell in the first row contains the number of configurations in the training set, for which the corresponding feature is selected; the second row contains the number of configurations in the training set, for which the feature is deselected. Table IV shows the feature frequencies for the configurations in Table III. For example, feature x_2 is selected in 5 and deselected in 3 configurations.

Algorithm 1 Generate configurations for projective sampling

1: **while** ($curr\_freq < thresh\_freq$ **AND** $mean\_err > err\_thresh$) **OR** ($curr \leq 3$) **do**
2: $c \leftarrow$ RAND() ▷ Randomly generate a configuration c
3: $S_{curr} \leftarrow S_{curr} \cup c$
4: UPDATE(T) ▷ Update the feature-freq table ***T***
5: $\epsilon_{curr} \leftarrow$ CART(S_{curr}) ; $\varepsilon \leftarrow \varepsilon \cup \epsilon_{curr}$
6: $mean\_err$ = MEAN(ε)
7: $\lambda_{curr} = (curr, \epsilon_{curr})$; $\Lambda_\delta \leftarrow \Lambda_\delta \cup \lambda_{curr}$ ▷ Add the current learning curve sample point (λ_{curr}) to Λ_δ
8: $curr\_freq \leftarrow \min(t[1,j], t[2,j])$; $curr \leftarrow curr + 1$
9: **end while**

Algorithm 1 defines the steps involved in the generation of Λ_δ. The most important parameter in the algorithm is $thresh\_freq$. This parameter sets a lower bound on the values of the feature-frequency table, which means that the sample configurations used to generate the sample points for the learning curve should have all the features selected and deselected for, at least, $thresh\_freq$ times. This threshold makes sample generation robust and diminishes the effect of any outliers in the sample configurations. The second parameter is the error threshold, err_thresh, which allows the algorithm to stop sampling if the models built so far have an acceptable accuracy. For some systems, the performance may be constant, irrespective of the variability among configurations, which means that the features do not have a significant influence on performance. In these systems, we can generate a prediction model with high accuracy using a very small set of configurations. Keeping a threshold value on the mean error avoids unnecessary measurements, since the current performance prediction model already yields a sufficient accuracy. The third parameter is the number of sample points that we need to project for a non-linear learning curve, which is 3, at least.

In the first step, the algorithm randomly samples a valid configuration and adds it to the current sample set S_{curr} (Lines 2 and 3). The feature-frequency table ***T*** is then updated by calculating the number of features that are selected and deselected in S_{curr} (Line 4). In the next step (Line 5), the CART prediction model is built using the current configuration set S_{curr}, and the prediction error ϵ_{curr} is calculated. The mean prediction error, which helps in evaluating the accuracy of the CART models built till this point, is calculated in Line 6. Using the set S_{curr} and ϵ_{curr}, we obtain a sample point λ_{curr} for the learning curve. The algorithm then adds the sample point λ_{curr} generated in the current iteration to the set Λ_δ (Line 7). Variable $curr\_freq$ holds the current minimum value of feature selection and deselection frequencies in ***T***. In the end, the set Λ_δ forms the final set of sample points we need to project the learning curve.

The performance profile of a system may not depend solely on individual features, but also interactions among features. For example, in a Web application, the performance may take a hit when caching is turned off and the application performs blocking reads [20]. Effects of interacting features have been studied extensively for fault localization in the field of combinatorial testing, where it has been seen that covering a 2-way and 3-way feature interactions can detect 93% and 98% of defects in a software [8]. Effects of feature interactions on performance prediction have also been studied too [16] [17]. Whereas a strategy based on *(t-way)* feature coverage might be an effective method to generate training samples for a prediction model, our objective is fundamentally different. Our primary objective is to generate sample points that estimate the learning behavior of the prediction model.

In Section VIII, we compare our method to a strategy based on 2-way and 3-way feature coverage, and show that feature-frequency-based sample generation is more effective than one based on *(t-way)* feature coverage.

VII. SUBJECT SYSTEMS

To compare progressive and projective sampling, we evaluate the two sampling strategies in terms of the sampling cost on six real-world configurable software systems. The six subject systems used in our evaluation include:

- *Apache HTTP Server* is the most popular Web server on the Internet. In our experiments, we consider 9 features resulting in 192 valid configurations.
- *Berkeley DB (C)* is an embedded key-value-based database library that provides scalable high-performance database management services to applications. Some of the

Table V: Overview of the six subject systems. Lang: language; LOC: lines of code; $|\mathbf{X}|$: number of all valid configurations; N: number of all features

	System	Domain	Lang.	LOC	$\lvert\mathbf{X}\rvert$	N
1	Apache	Web Server	C	230,277	192	9
2	LLVM	Compiler	C++	47,549	1,024	11
3	x264	Encoder	C	45,743	1,152	16
4	Berkeley DB	Database	C	219,811	2,560	18
5	Berkeley DB	Database	JAVA	42,596	400	26
6	SQLite	Database	C	312,625	3,932,160	39

applications using Berkeley DB are Subversion, Bitcoin, and Sendmail. In our experiments, we consider 18 features resulting in 2560 valid configurations.

- *Berkeley DB (Java)* is a re-implementation of Berkeley DB in Java with 26 features and 400 valid configurations.
- *LLVM* is a popular compiler and virtual-machine framework used for a variety of languages. We consider 11 features and 1024 valid configurations.
- *SQLite* is the most popular lightweight relational database management systems today. It is used by several browsers and operating systems as an embedded database. We consider 39 features that give rise to more than 3 million valid configurations.
- *x264* is a video-encoding library that encodes video streams to H.264/MPEG-4 AVC format. It is used by several video converters and media players, such as VLC. x264 has 16 features and 1152 valid configurations.

More information on these systems can be found as a part of *SPL Conqueror* project [17]; Table V provides an overview of the subject systems.

VIII. EVALUATION

By conducting experiments on six real-world configurable systems, we aim at answering the following research questions:

- **RQ1**: Between progressive and projective sampling, which is cost efficient? (Section VIII-A)
- **RQ2**: Which is the best projective function that fits the learning curve of a configurable system? (Section VIII-B)
- **RQ3**: Is a heuristic based on t-way feature coverage or feature frequencies better for the initial sample generation of projective sampling? (Section VIII-C)

A. RQ1: Progressive vs. Projective

Since cost efficiency is the primary determinant for judging the effectiveness of a sampling strategy, we compared the total cost of sampling and prediction according to Equation 2, for all six subject systems, using progressive and projective sampling, as shown in Table VI. For both progressive and projective sampling, we calculated the cost and accuracy of building prediction models with the optimal sample set size (n^*). The value of n^* is determined through the respective sampling techniques. In the case of projective sampling, the accuracy shown is the real accuracy calculated after the prediction model is built with n^* samples and not the accuracy derived through the projected learning curve at n^*. To calculate the optimal sample size, we have set the size of the score set (S) to be proportional to the total number of configurations of the system ($\mathbf{X}$). Specifically, the size of S is set to $|\mathbf{X}|/3$. The tuning parameter R, which controls the cost ratio between measurement effort and prediction accuracy, is set to 1. This means that we equally weigh the cost incurred in measuring samples and the cost due to prediction error. We can see from Table VI that, for all the six subject systems, projective sampling is more cost efficient than progressive sampling.

Table VI: Cost and accuracy of progressive and projective sampling

	Cost		**Accuracy (%)**	
	Progressive	Projective	Progressive	Projective
Apache	616	**602**	92	**92**
Berkeley DB (C)	86679	**11206**	1	**88**
Berkeley DB (Java)	355	**328**	96	**96**
LLVM	1263	**946**	96	**97**
SQLite	2517	**1828**	98	**99**
x264	2807	**2121**	92	**95**

Although we prioritize cost efficiency over the absolute accuracy, we also compared the two strategies based on accuracy, to validate whether the accuracy of the model is acceptable. The prediction accuracy was calculated based on a testing set that was of the same size as the training set. Also, we ran the experiment 30 times and the prediction accuracy reported was the average of the runs. We see in Table VI that projective sampling outperforms progressive sampling in terms of cost and also in terms of accuracy. For Berkeley DB (C), progressive sampling gets stuck in a local optimum and produces a very low accuracy of 1%. Progressive and projective sampling are comparable in terms of accuracy and cost for Apache and Berkeley DB (Java), where the learning curve behaves ideally with no jumps or temporal variations. However, for all other learning curves, projective sampling is considerably more cost efficient than progressive sampling.

Decision Cost. One of the key benefits of projective sampling is its ability to give a prognosis of the learning behavior of the prediction model. In other words, stakeholders can use the projected learning curve to obtain an estimate of the prediction accuracy. They can check whether this value falls under an acceptable range and decide whether to use the prediction model. The cost incurred in taking this decision is the same as the cost incurred in building the initial configuration set, which is in our case the size of the set Λ_δ. Table VII shows this cost for each of the six systems. We can see that Λ_δ is significantly smaller than the total cost incurred when the final prediction model is built with an optimal training set size.

B. RQ2: Comparison of Learning Curves

Figure 5 shows the learning curves for each of the six subject systems generated by progressive and projective sampling. The black squares denote the initial learning curve sample points (Λ_δ) generated using the feature-frequency heuristic. Since the parameter err_thresh controls the number of iterations of the sampling algorithm, its value affects the size of Λ_δ. We have set the value of err_thresh to 5%. The only exception is SQLite, where the value is set to 1%. The reason is that, for SQLite, we observed a high prediction accuracy

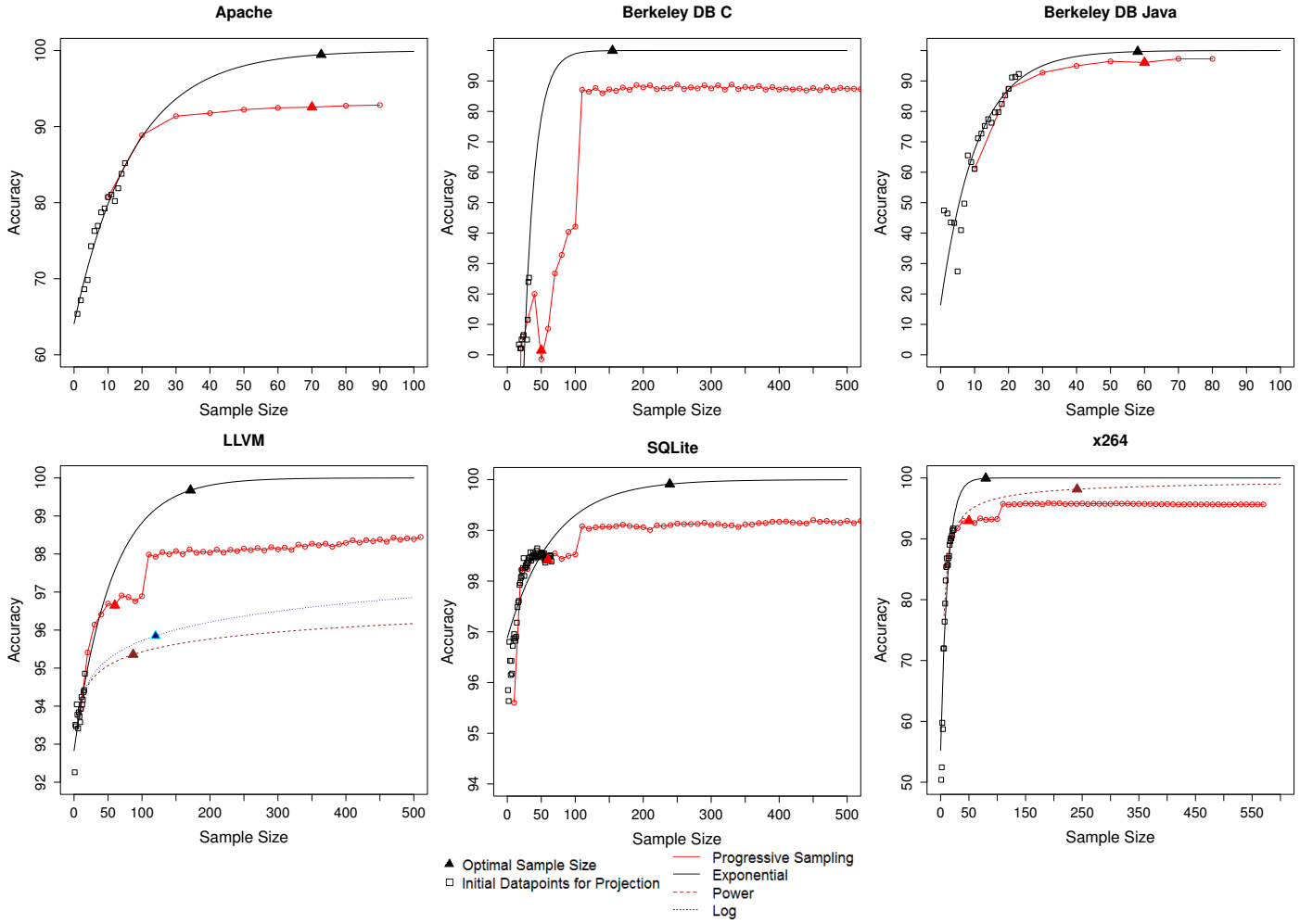


Figure 5: Learning curves for progressive and projective sampling

from the CART model with a very small sample size, and thus we set a lower value for effective comparison of feature-frequency sampling. We have empirically set the value of the parameter *thresh_freq* at 5 for all subject systems. For projective sampling, we choose the projective function that has the highest correlation with the initial set Λ_δ and for which the optimal number of samples (n^*) generated by that curve is less than the total number of configurations of the system (**X**). The optimal sample size for a class of projective functions can be calculated from the equations in Table II. The upper bound on the optimal sample size helps us to eliminate unrealistic sample sizes and thus to narrow down our set of candidate projective functions. For comparison, apart from the chosen projective function for each system, Figure 5 also shows all the functions whose correlation value was greater than 0.9. The triangular wedges show the optimal sample sizes (n^*) for each of the curves. For both progressive and projective sampling, these values are generated using the cost-minimization techniques described in Section V.

We can see from Figure 5 that exponential functions are the most robust among all the projective functions when used to fit the learning curve. They exhibit correlations greater than 0.91 for all the six systems, and they have been selected as projective functions for 4 out of the 6 systems (Apache, Berkeley C, Berkeley Java, SQLite). For the remaining two systems, a logarithmic function (LLVM) and a power-law function (x264) have been selected. The effectiveness of exponential functions is corroborated by the fact that they had overestimated the accuracy of the learning curve at the optimal sample size by only 5%, with a standard deviation of 4%. For the generation of a cost-effective optimal sample size n^*, progressive sampling is sensitive to local variations in the learning curve and gets stuck in a suboptimal region, for example, in Berkeley C. In contrast, projective sampling generates realistic values of the *optimal* sample set, which are high in accuracy as well as cost efficient.

Table VII: Comparison of decision cost (size of Λ_δ) and total cost

	Apache	Berkeley DB (C)	Berkeley DB (Java)	LLVM	SQLite	x264
Decision Cost	15	32	23	16	65	23
Total Cost	602	11206	328	946	1828	2121

Table IX: t-way vs. feature-frequencies; r: Pearson's correlation coefficient.

		r	p *value*	**Total Cost**
Apache	2-way	0.981	0.000017	764
	3-way	0.977	0	610
	feature frequency	**0.989**	0	**602**
Berkeley DB (C)	2-way	0.858	0.000001	69572
	3-way	0.903	0	11212
	feature frequency	**0.92**	0	**11206**
Berkeley DB (Java)	2-way	0.895	0.000468	818
	3-way	0.943	0	6102
	feature frequency	**0.971**	0	**328**
LLVM	2-way	–	–	–
	3-way	0.006	0.981767	2082
	feature frequency	**0.915**	0.000001	**946**
SQLite	2-way	0.888	0	2470
	3-way	0.941	0	**1697**
	feature frequency	**0.943**	0	1828
x264	2-way	0.942	0.000005	2798
	3-way	0.968	0	2803
	feature frequency	**0.97**	0	**2121**

Table VIII: Size of Λ_δ (t-way vs. feature-frequency)

	Apache	Berkeley DB (C)	Berkeley DB (Java)	LLVM	SQLite	x264
2-way	8	20	10	10	25	12
3-way	18	56	27	17	77	29
Feature frequency	15	32	23	16	65	23

C. RQ3: T-Way vs. Feature Frequency

In this section, we compare the t-way heuristic to select the initial sample set, which is the de-facto standard in combinatorial [9] and product-line testing [3], to our proposal of a feature-frequency heuristic. We mentioned earlier that the objective of the initial sample generation is to model the learning behavior of the system and achieve a high correlation between the initial dataset Λ_δ and the projective learning curve. Thus, the first criterion we use to evaluate the t-way and feature-frequency heuristics is the degree of correlation between the initial dataset and the chosen projective function for each system. The second criterion is based on the total cost of the prediction process. Ideally, the correlation should be close to 1 and the total cost (calculated at optimal sample size n^*) be minimal.

We use the tool JENNY to generate 2-way and 3-way feature-coverage configurations used as the initial sample points Λ_δ.[1] Table IX shows the comparison between 2-way, 3-way, and feature-frequency sampling in terms of correlation (Pearson correlation coefficient and p value) and the total cost.

We can see in Table VIII that 2-way feature coverage scores worse than both 3-way and feature frequency in terms of both correlation and cost. This is due to the number of sample configurations or the size of Λ_δ. The size of Λ_δ is generally smaller when a 2-way heuristic is used, compared to a 3-way or feature-frequency heuristic. Feature-frequency sampling induces comparable cost to 3-way sampling for two systems (Apache and Berkeley C). However, the correlation value for the feature-frequency heuristic in both systems is higher. For the rest of the systems, we can see that the feature-frequency heuristic is better in terms of both correlation and cost. These results, along with the fact that the size of Λ_δ is smaller using the feature-frequency heuristic as compared to the 3-way heuristic for all the six systems, shows us that the feature-frequency heuristic can generate more accurate learning curves with a lower numbers of measurements.

[1] Jenny : http://burtleburtle.net/bob/math/jenny.html

IX. Threats to Validity

To increase internal validity, we performed automated random sampling, this way, avoiding misleading effects of specifically selected samples for building prediction models. We randomly selected samples of specific sizes (e.g., Λ_δ) from the entire population of each subject system. We repeated each random sampling 30 times for training and testing the prediction models, and we reported only the mean of cumulative results for the evaluation metrics, such as sampling cost, prediction accuracy, and correlation coefficients. In addition, we used a widely-accepted tool for t-way feature-coverage generation, to make sure that the generated sets are correct.

In our experiments, the value of the tuning parameter R, which controls the cost ratio between measurement effort and prediction accuracy, is set to 1. However, the value of this parameter is domain-specific, and can be set by a domain expert. Nevertheless, in cases where the precise value of this ratio is unknown, giving equal weights to both the cost factors seems to be a fair assumption. In addition, we repeated our experiments with multiple values of the parameter R and observed our results to be robust for these small ranges. However, we leave a systematic sensitivity analysis to future work.

We have defined our cost model based on prior research, and we argue that the measurement cost of both training and

testing samples needs to be incorporated in the cost model. Still, there might be alternative definitions of cost models, which are different than ours and equally valid.

To increase external validity, we used a public dataset consisting of six real-world systems, covering different domains, with different sizes, different configuration mechanisms, and different programming languages. All the subject systems used in our case study are deployed and used in real-world scenarios. However, we are aware that the results of our experiments are not automatically transferable to all other configurable systems, but we are confident that we controlled this threat sufficiently.

X. Related Work

A. Performance Prediction

Recent approaches have used a combination of measurement and prediction techniques to evaluate the performance of software systems. Among the performance prediction models, it is important to distinguish between two categories of models found in literature. The first type of models, which can be referred to as *white-box* models, are built early in the life cycle, by studying the underlying design and architecture of the software system in question. The idea is to identify performance bottlenecks early, so that developers can take corrective actions. Queueing networks, Petri Nets, and Stochastic Process Algebras are commonly used for this task [1]. The second type of models, called *black-box* models, do not make any assumption on the design and architecture, effectively treating the system as a black box. In this paper, we use black-box predictive models.

Guo et al [5] used CART to predict the performance of configurable systems. On the same dataset as ours, they observed an average accuracy of 94%. We build on their prediction model, and we study how to determine the minimum sample size, rather than proposing a new learning technique. Yi et al. [21] proposed an algorithm based on Fourier Learning for the performance prediction of configurable systems. Their method provides theoretical guarantee of prediction accuracy and confidence level, but it follows typical random sampling that is terminated only in terms of prediction accuracy.

Westermann et al. [19] analyzed various statistical inference techniques to predict the performance of configurable systems. They also analyzed three different configuration generation methods, including Random Breakdown, Adaptive Equidistant Breakdown, and Adaptive Random Breakdown. In their work, they do not take the measurement cost into account. We use a composite cost function to guarantee an optimum between accuracy and measurement cost. Our approach has further the advantage of giving stakeholders an early prognosis of the prediction model through a minimal decision cost.

Siegmund et al. [17] used a measurement-based technique to predict performance by detecting feature interactions. In follow-up work [16], they consider also numeric configuration options by combining experimental designs with binary-option sampling. The number of samples needed to be measured using their approach is higher than other prediction models, such as CART, due to their focus on explaining the performance of a system (i.e., making the influences of all features and their interactions explicit), which is a different goal.

B. Sampling Strategies

Provost et al. [15] introduced the idea of progressive sampling and proved that geometric progressive sampling is more efficient than arithmetic progressive sampling when model-building cost is high. However, in the case of performance prediction of configurable systems, the cost of building prediction models, such as CART, is comparatively low, but the measurement cost is often high. In this case, arithmetic progressive sampling is more efficient.

Weiss and Tian [18] combined measurement cost and accuracy into a single composite cost function and used it to evaluate the prediction process. We use their cost functions in our approach. They evaluated only progressive sampling approaches and did not develop a sampling strategy to minimize the cost.

Last [10] used the cost function proposed by Weiss and Tian [18] and proposed projective sampling, which guarantees a minimum cost and provides a numeric value for the optimal sample size. We adapted their approach of projective sampling for performance prediction models. However, they did not provide guidelines on how to generate a good initial sample for projecting the learning curve. We solve this problem using a heuristic based on feature frequencies and compare it to a typical heuristic based on t-way feature coverage.

XI. Conclusion

We adapted two sampling strategies, progressive sampling and projective sampling, for the performance prediction of configurable systems. To evaluate and compare the two sampling strategies, we use the sampling cost, which considers the prediction accuracy and the measurement effort simultaneously. In addition, we used two heuristics based on feature frequencies and t-way feature coverage to generate the initial sample in projective sampling. We conducted empirical studies on six real-world configurable systems to determine an ideal sampling strategy for performance prediction of configurable systems.

Our key findings are as follows. First, projective sampling is better than the progressive sampling in terms of both sampling cost and prediction accuracy, but it suffers from a dependency on a proper initial sample and projective function. To obtain a good initial sample for projective sampling, our heuristic based on feature frequencies is more effective than the standard approaches based on 2-way and 3-way feature coverage. Among four common projective functions, the exponential function is the best to fit the learning curves of our subject systems accurately and robustly. Furthermore, we recommend arithmetic progressive sampling instead of geometric progressive sampling, because measuring the performance of configurations is often more costly than learning a prediction model based on the training set.

Our empirical findings are meant to help stakeholders in designing sampling strategies for performance prediction. In the future, we will perform a systematic sensitivity analysis of the cost ratio between measurement effort and prediction accuracy in our cost model.

ACKNOWLEDGMENT

This work has been funded by the Canadian Natural Sciences and Engineering Research Council, Ontario Research Fund, and German Research Foundation (AP 206/4, AP 206/5, and AP206/6).

REFERENCES

[1] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295–310, 2004.

[2] B. W. Boehm, *Software engineering economics*. Prentice-hall Englewood Cliffs (NJ), 1981, vol. 197.

[3] M. B. Cohen, M. B. Dwyer, and J. Shi, "Coverage and adequacy in software product line testing," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2006, pp. 53–63.

[4] L. J. Frey and D. Fisher, "Modeling decision tree performance with the power law," in *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, 1999, pp. 59–65.

[5] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski, "Variability-aware performance prediction: A statistical learning approach," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 301–311.

[6] W. Härdle, *Smoothing techniques: With implementation in S*. Springer Science & Business Media, 1991.

[7] G. H. John and P. Langley, "Static versus dynamic sampling for data mining." in *Proceedings of the Conference on Knowledge Discovery and Data mining (KDD)*. ACM, 1996, pp. 367–370.

[8] D. R. Kuhn, R. N. Kacker, and Y. Lei, "Practical combinatorial testing," *NIST Special Publication*, vol. 800, no. 142, p. 142, 2010.

[9] R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *IT Professional*, vol. 10, no. 3, pp. 19–23, 2008.

[10] M. Last, "Improving data mining utility with projective sampling," in *Proceedings of the Conference on Knowledge Discovery and Data mining (KDD)*. ACM, 2009, pp. 487–496.

[11] A. Lazarevic and Z. Obradovic, "Data reduction using multiple models integration," in *Principles of Data Mining and Knowledge Discovery*. Springer, 2001, pp. 301–313.

[12] R. Leite and P. Brazdil, "Improving progressive sampling via meta-learning on learning curves," in *Proceedings of the European Conference on Machine Learning (ECML)*. Springer, 2004, pp. 250–261.

[13] H. Liu and H. Motoda, "On issues of instance selection," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 115–130, 2002.

[14] D. C. Montgomery, G. C. Runger, and N. F. Hubele, *Engineering statistics*. John Wiley & Sons, 2009.

[15] F. Provost, D. Jensen, and T. Oates, "Efficient progressive sampling," in *Proceedings of the Conference on Knowledge Discovery and Data mining (KDD)*. ACM, 1999, pp. 23–32.

[16] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2015.

[17] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 167–177.

[18] G. Weiss and Y. Tian, "Maximizing classifier utility when there are data acquisition and modeling costs," *Data Mining and Knowledge Discovery*, vol. 17, no. 2, pp. 253–282, 2008.

[19] D. Westermann, J. Happe, R. Krebs, and R. Farahbod, "Automated inference of goal-oriented performance prediction functions," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 2012, pp. 190–199.

[20] C. Yilmaz, A. S. Krishna, A. Memon, A. Porter, D. C. Schmidt, A. Gokhale, and B. Natarajan, "Main effects screening: A distributed continuous quality assurance process for monitoring performance degradation in evolving software systems," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2005, pp. 293–302.

[21] Y. Zhang, J. Guo, E. Blais, and K. Czarnecki, "Performance prediction of configurable software systems by Fourier learning," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 2015.