Meta-learning Performance Prediction of Highly Configurable Systems: A Cost-oriented Approach

by

Atri Sarkar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2016

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 work, 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 develop two sampling algorithms. One based on a traditional method of *t-way* feature coverage, and another based on a new heuristic of *feature-frequencies*. Using empirical data from six real-world systems, we evaluate the two sampling algorithms and discuss trade-offs. Furthermore, we conduct extensive sensitivity analysis of the cost model metric we use for evaluation, and analyze stability of learning behavior of the subject systems.

Acknowledgements

I would like to thank Prof. Krzysztof Czarnecki for giving me this opportunity. Prof. Czarnecki, with his tenacity, knowledge and insight has truly been an inspiration to all of us.

I thank Dr. Jianmei Guo for being a mentor and guiding me throughout the work. Without his work on performance prediction, it would not have been possible for me to delve deeper into the subject. I also thank Prof. Sven Apel and Dr. Norbert Siegmund for their input and feedback on the work.

I thank everyone at GSD Lab (Dr. Michal Antkiewicz, Manish, Eldar, Jimmy, Rodrigo, Jordan, Peiyuan, Pavel, Ed and Hristiyan) for keeping me in good company. Especially Leonardo Passos, for all the conversations, and making this part of grad school life bearable.

Most importantly, I would like to thank my wife Piyusha for being a pillar of support, and allowing us to venture on this chaotic yet exciting journey. My parents Ashim and Purabi, for always believing in me and teaching me to question everything.

Dedication

To Piyusha, Purabi and Ashim.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A hallmark of modern software systems is *configurability*. In fact, most large scale software from web-servers to database systems to complex enterprise applications provide users with configurable options to tailor the behavior of the system. Selecting configuration options or *features* allow stakeholders to customize these highly-configurable systems in various ways, giving rise to a multitude of system *variants* or *configurations*. These configuration options may often be referred to as *features*, and act as a source for *product line* architecture [7][1]. Along with the desired functional role, the features often have secondary influences on non-functional properties like performance or cost.

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 [31]. 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. [31] 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. [13] 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

[1]However, we contextualize our study to any configurable system, without holding any assumption on its relation to product lines. We overload the term *feature*, and use it to refer to the configurable options.

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.

In this work, we study the learning behavior of performance prediction models with respect to size and quality of training samples. Consequently, we aim at a smart sampling strategy that dynamically determines a "good" sample for a given system. A sample is considered 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 [33], 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 [27], and a state-of-the-art technique, called *projective* sampling [20]. 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 [18], and a novel heuristic based on *feature frequencies*. We empirically compare the results of these sampling strategies on our six real-world configurable systems. The thesis focuses primarily on *variability-aware* performance prediction, which studies performance of a system with respect to configurable features. We do not take into account performance influence due to other factors like workload.

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.

- We conduct a thorough sensitivity analysis on our cost model parameters. Apart from giving us an insight on how different parameter values influence the prediction process, our analysis can help practitioners choose the appropriate strategy for their system.

- Learning behavior of a system is often a derivative of two major factors. The underlying prediction model and the specific system under study. We use our case studies to look deeper into this assumption, and study how sample quality play a role in determining the learning behavior of a prediction model.

1.1 Performance prediction in practice

Figure 1.1 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 [31, 13]. 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 technique based on t-way feature coverage, commonly used in combinatorial testing [18]. 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 [13].

In previous work, prediction accuracy was the main evaluation metric used to estimate the utility of the prediction models [31, 13, 34]. Recognizing the fact that there is a cost involved in measuring the sample of configurations for building the prediction model, in our work we 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. Section 3 discusses the cost model in detail.

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[2]. 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

[2]Although we do not explicitly consider *evaluation* set, our cost model of section 3 is generic enough to include these additional measurements

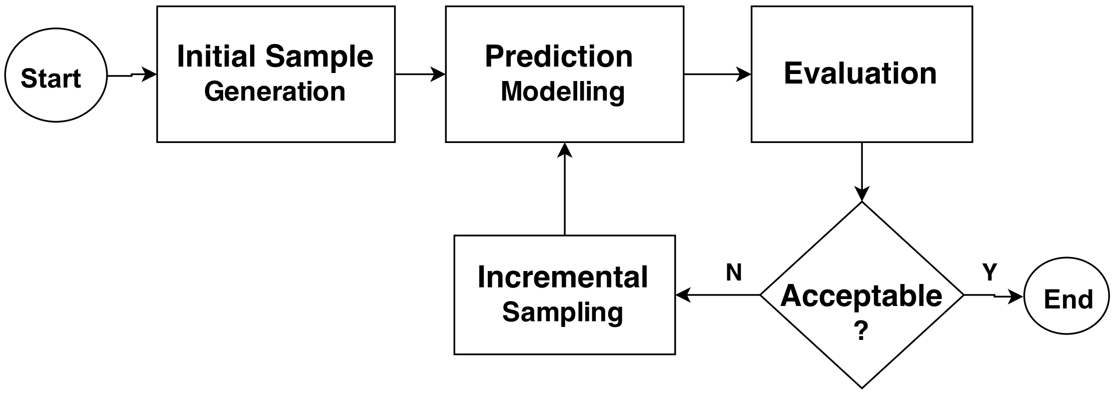


Figure 1.1: General process of performance prediction by sampling

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 [27], as shown in Figure 1.2.

The learning curve of Figure 1.2 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 [27] and projective sampling [20], which will be explained in section 4.

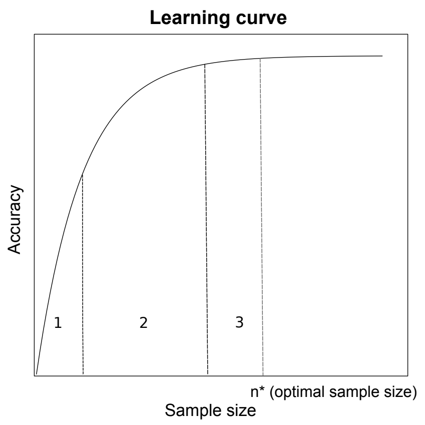


Figure 1.2: Regions of the learning curve in dependence of the sample size: (1) Steep incline; (2) Gradual incline; (3) Plateau; n^*: marks the optimal sample size

Table 1.1: Learning curve points (λ_n) for Apache (n : sample set size, ϵ_n : relative error %)

n	ϵ_n	n	ϵ_n
10	21.51	60	7.88
20	10.92	70	7.53
30	9.00	80	7.43
40	8.62	90	7.28
50	8.15		

Chapter 2

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 **X**.

A learning curve represents a mapping between a training-set size and the corresponding accuracy. A pair $\lambda_n = (n, \epsilon_n)$ represents a point in a 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 a 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 6, 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. [13] to generate the valid configurations. Table 1.1 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 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.

Chapter 3

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 1.2), 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 [33] 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 [33] to include the cost incurred in measuring the testing set along with the training set:

$$\begin{aligned} TotalCost &= \text{Cost}_{Measurement(Training)} \\ &+ \text{Cost}_{Measurement(Testing)} \\ &+ \text{Cost}_{ModelBuilding} \\ &+ \text{Cost}_{PredictionError} \end{aligned} \tag{3.1}$$

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. Therefore, given a training set of size n, we have the following cost function of n:

$$TotalCost(n) = \theta \cdot n + \epsilon_n \cdot |S| \cdot R \tag{3.2}$$

where θ (>1) is the *multiplier* accounting for the additional samples in the testing set. For example, a 50:50 split between the training and testing sets, will result in a θ value 2 (1+1) - one each for training and testing set. Similarly, a θ value 3 (2+1) would mean that the testing set is twice the size of the training set. ϵ_n is the prediction error of this performance prediction model built with the n configurations, $|\mathrm{S}|$ 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 [9]. 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 4.1b) [20]. This characteristic enables us to easily find the global minimum of the cost with respect to the sample size (see Section 4).

Chapter 4

Sampling Strategies

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 [23] [12] 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 [15]. Still, for a wide range of sample sizes, we see the learning curve to be monotonically non-decreasing in our experiments.

4.1 Progressive Sampling

Progressive sampling is a popular sampling strategy that has been used for a variety of learning models [27] [21]. 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 [16]. 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 [27]. This is an advantage in cases where building the model is an expensive process. For example, as listed in Table 1.1, if we set an acceptable prediction accuracy of the system to 92%, we can observe that the optimal sample size for that accuracy is 60. 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

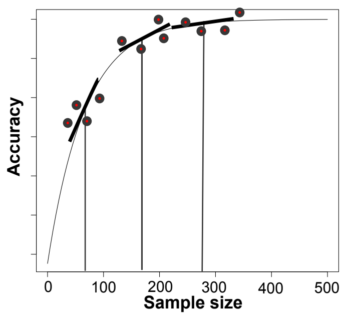
$$\begin{aligned}10_{(iteraton_0)} + 10_{(iteraton_1)} + 20_{(iteraton_2)}\\ +40_{(iteraton_3)}\end{aligned}$$

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.

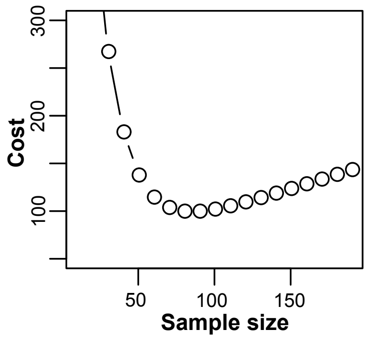
Stopping Criteria: For both arithmetic and geometric progressive sampling, we need to decide when to stop sampling more configurations for measurement. This is a critical step that needs to be performed in every iteration and to check whether the built prediction model has converged to an acceptable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

4.1.1 Gradient-Based

Linear Regression with Local Sampling (LRLS) uses the gradient of the learning curve to detect convergence [22]. 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 4.1a. 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.



(a) Gradient-based



(b) Cost minimization

Figure 4.1: Two stopping criteria for progressive sampling

4.1.2 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 3.2 to detect convergence of the learning curve [33]. 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 4.1b), 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.

4.2 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 [20], thus providing stakeholders with an estimate of the

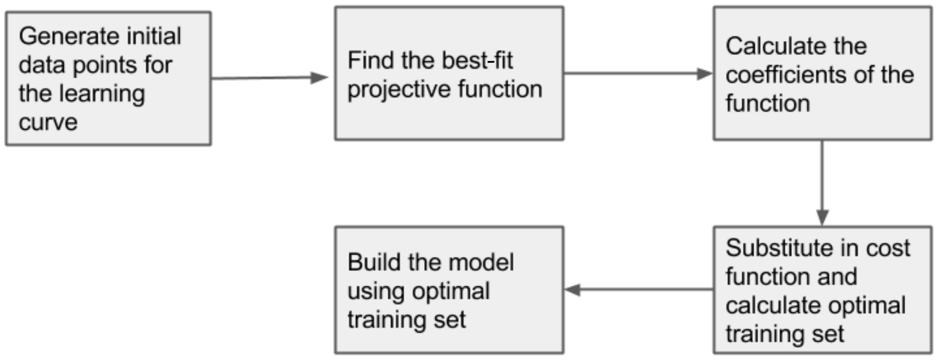

Generate initial data points for the learning curve
Find the best-fit projective function
Calculate the coefficients of the function
Substitute in cost function and calculate optimal training set
Build the model using optimal training set

Figure 4.2: Overview of projective sampling

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 4.2, 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 use a novel feature-frequency heuristic for sample generation (Section 5) 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 [12] [20], as shown in Table 4.1. 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 3.2. Using the information from the initial sample points, we follow the approach proposed by Last [20] 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 [24].

The optimal sample size can be defined as the size of the training sample that minimizes the cost function of Equation 3.2, ensuring the most optimal tradeoff between measurement cost and

Table 4.1: Projective functions of learning curves

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

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 4.1b 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 a well-behaved learning curve having the distinct regions as described in section 1.1 [20]. 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 4.1 shows the values of n^* (cf. Fig. 1.2) for the four different learning-curve equations in terms of the cost-equation variables of Equation 3.2. Detailed derivation of the formulae is presented in appendix A.1. In the following sections, we use n^* to represent the optimal sample size that minimizes the cost function of Equation 3.2.

Chapter 5

Initial Sample Generation

5.1 Feature-frequency based 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 [13]. 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 2), 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:

$$1 \leq \sum_{j=1}^{\delta} x_i(j) < \delta \tag{5.1}$$

where $x_i \in \{1, 0\}$ is a Boolean variable representing a feature i being selected or deselected for a configuration j. Table 5.1 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 [13]. In our approach, we use a combination of incremental random 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 $\boldsymbol{T}$ (Table 5.2), 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 5.2 shows the feature frequencies for the configurations in Table 5.1. For example, feature x_2 is selected in 5 and deselected in 3 configurations.

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

Table 5.1: 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 5.2: 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

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 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. 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

Algorithm 1 Feature-frequency based sample generation for projective sampling

```
1:  while ($curr\_freq < thresh\_freq$ 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 $\boldsymbol{T}$
5:      $\epsilon_{curr} \leftarrow$ CART($S_{curr}$)
6:      $\lambda_{curr} = (curr, \epsilon_{curr})$
7:      $\Lambda_\delta \leftarrow \Lambda_\delta \cup \lambda_{curr}$   ▷ Add the current learning curve sample point ($\lambda_{curr}$) to $\Lambda_\delta$
8:      $curr\_freq \leftarrow \min(t[1, j], t[2, j])$
9:      $curr \leftarrow curr + 1$
10: end while
```

sample points we need to project the learning curve.

5.2 *t-way* sample generation

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 [35]. 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 [18]. Effects of feature interactions on performance prediction have also been studied too [30] [31]. 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.

The first step in *t-way* sampling is to generate a covering array (CA) of strength 't'. A covering array for a system of X features has the following properties:

(1) Following the formulation from section 2, each feature $x_i : 1 \leq i \leq X$ can take 2 values. Either 1 when the feature is active, or 0 when inactive.

(2) If $S_{CA}(t)$ is a covering array of strength *t*, the rows of any $|S_{CA}(t)| \times t$ subarray must cover all possible combination of feature values (2^t), for the participating *t* features in the subarray, at least once[1].

where $|S_{CA}(t)|$ is the size (number of rows) of the covering array. Algorithm 2 shows the process of generating samples using the t-way covering array. Compared to the previous algorithm, t-way selects a random configuration from the t-way covering array without replacement, instead of the entire configuration space. Also, since the covering array has a fixed size, the sampling stops when all the configurations from the covering array are included in S_{curr}. Sampling from a pre-defined set ($S_{CA}(t)$), eliminates the need of feature-frequency based stopping criteria (line 4 and 8 of algorithm 1).

In Section 7, we compare the two sampling methods on the basis of total cost and accuracy.

[1] The restriction of *at least once* is a more generalized form of *at least 'k' times*, used popularly for combinatorial testing

Algorithm 2 t-way sample generation for projective sampling

```
1: while ($curr \leq |S_{CA}(t)|$) do
2:     $c \leftarrow$ SELECT()                                   ▷ Select a configuration from $S_{CA}(t)$
3:     $S_{curr} \leftarrow S_{curr} \cup c$
4:     $\epsilon_{curr} \leftarrow$ CART($S_{curr}$)
5:     $\lambda_{curr} = (curr, \epsilon_{curr})$
6:     $\Lambda_\delta \leftarrow \Lambda_\delta \cup \lambda_{curr}$          ▷ Add the current learning curve sample point ($\lambda_{curr}$) to $\Lambda_\delta$
7:     $curr \leftarrow curr + 1$
8: end while
```

Chapter 6

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 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.

Table 6.1: 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	$\|\mathbf{X}\|$	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

- *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 [31]; Table 6.1 provides an overview of the subject systems.

Chapter 7

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 7.1)
- **RQ2**: Which is the best projective function that fits the learning curve of a configurable system? (Section 7.2)
- **RQ3**: Is a heuristic based on t-way feature coverage or feature frequencies better for the initial sample generation of projective sampling? (Section 7.3)

7.1 RQ1: Progressive vs. Projective

Since cost efficiency is the primary determinant for judging the effectiveness of a sampling strategy, we compare the total cost of sampling and prediction according to Equation 3.2, for all six subject systems, using progressive and projective sampling. For both progressive and projective sampling, we calculate 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 figures shown is the real accuracy calculated after the prediction model is built with n^* samples and not the accuracy estimated through the projected learning curve at n^*. To calculate the optimal sample size, we 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. Evaluation of sensitivity of our model to the cost ratio (R) is presented later in section 8.

Figure 7.1 shows the change in total cost of sampling with respect to sample size for the six subject systems. The blue marker and the n^* value is the optimal sample size and cost estimated by progressive sampling; the black wedge indicates the global minimum and the black square indicates cost estimated through projective sampling. Since projective sampling works on a cost curve which is an approximation of the real cost, the black square of projective sampling cost might not necessarily lie on the real cost curve which is shown in the figure. We see that progressive sampling was able to reach the global minimum for only 3 out of the 6 subject systems. This was possible in cases where the cost curve exhibited certain conditions : presence of singular minima (Apache and Berkeley DB (JAVA)) *or* the first local minima (ordered by the sample size) is also the global minima (SQLite). The cost curve follows this condition when the learning is well behaved i.e monotonically non-decreasing with distinct regions (steep incline, gradual incline, plateau), as seen in the curves of Apache and Berkeley DB (JAVA) (Fig. 7.2). Projective sampling, on the other hand, is more robust. For learning curves that do not follow this established pattern, e.g temporal variations in Berkeley DB (C), jumps in x264, projective sampling is an accurate estimation of the global minimum cost. Even for well-behaved learning curves, projective sampling's estimate of cost is either same (Apache) or in close proximity of global minimum (Berkeley DB (JAVA) and SQLite).

Although we prioritize cost efficiency over the absolute accuracy, we also compare the two strategies based on accuracy, to validate whether the accuracy of the model is acceptable. The prediction accuracy is calculated based on a testing set that is of the same size as the training set. Also, we run the experiment 100 times and the prediction accuracy reported is the average of the runs. We see in Table 7.1 that in terms of accuracy, there is no significant difference between progressive and projective sampling. This observation vindicates the idea that prediction accuracy by itself is not an appropriate metric for judging the utility of the sampling process.

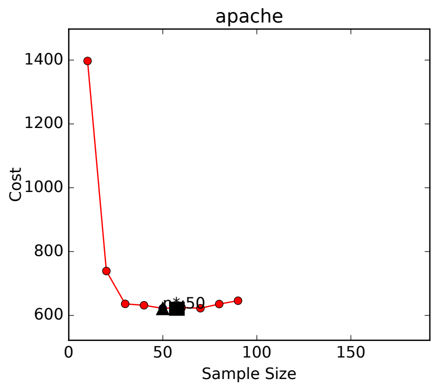
Decision Cost. One of the key benefits of projective sampling is its ability to give a prognosis of the learning behavior of the prediction model. In other words, stakeholders can use the projected learning curve to obtain an estimate of the prediction accuracy and cost. They can check whether this value 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 7.2 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.

Table 7.1: Prediction accuracy at n^* (optimal sample size) achieved by different sampling strategies

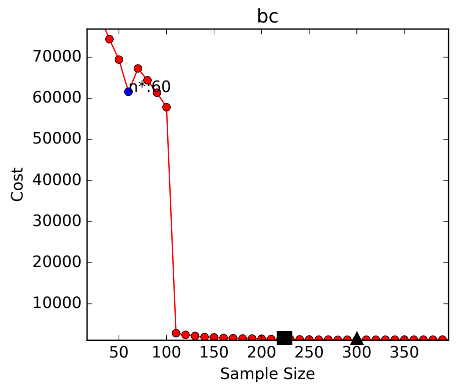
		Apache	Berkeley DB (C)	Berrkeley DB (JAVA)	LLVM	x264	SQLite
feature-frequency	ff = 3	91.78	94.70	94.23	96.33	95.58	96.71
	ff = 4	92.02	96.25	95.08	96.51	95.82	96.73
	ff = 5	92.10	96.10	95.00	96.58	97.37	96.74
	ff = 6	92.15	98.51	95.78	96.64	97.49	96.59
t-way	t = 2	92.23	92.43	95.46	95.86	94.04	98.81
	t = 3	92.38	99.43	95.25	96.54	98.64	99.30
progressive		92.71	99.58	97.04	98.50	99.81	96.86

Table 7.2: Comparison of decision cost (size of Λ_δ) and total cost

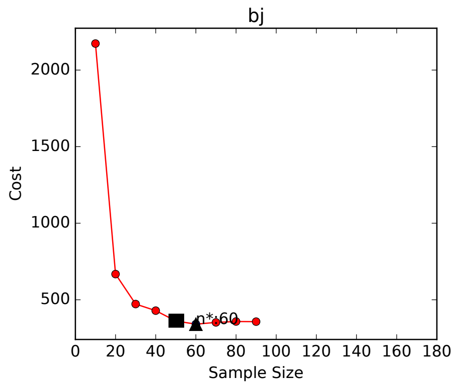
	Apache	Berkeley DB (C)	Berkeley DB (Java)	LLVM	x264	SQLite
Decision Cost	19	22	24	17	23	671
Total Cost	633	1390	366	1269	671	7604



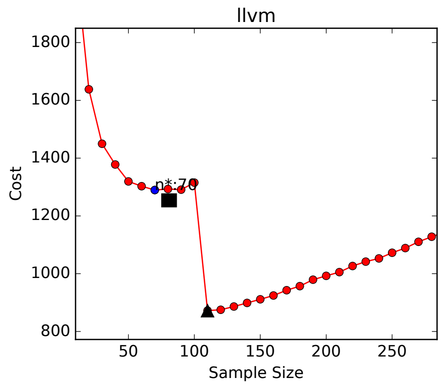
(a) Apache



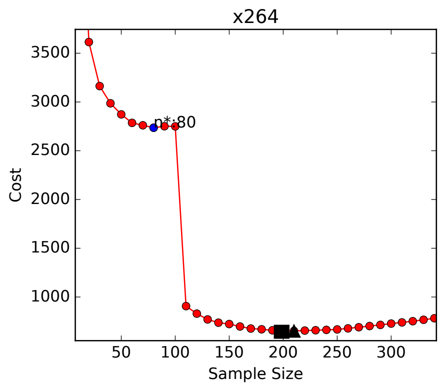
(b) Berkeley DB C



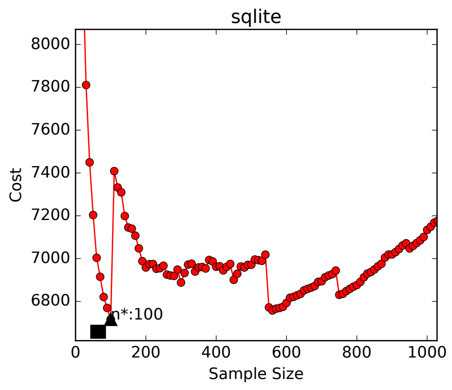
(c) Berkeley DB JAVA



(d) LLVM



(e) x264



(f) SQLite

Figure 7.1: Cost curve and minimum total cost estimated through various sampling strategies. ● - Progressive sampling; ■-Projective sampling; ▲-Global minimum; ●—● - Cost curve

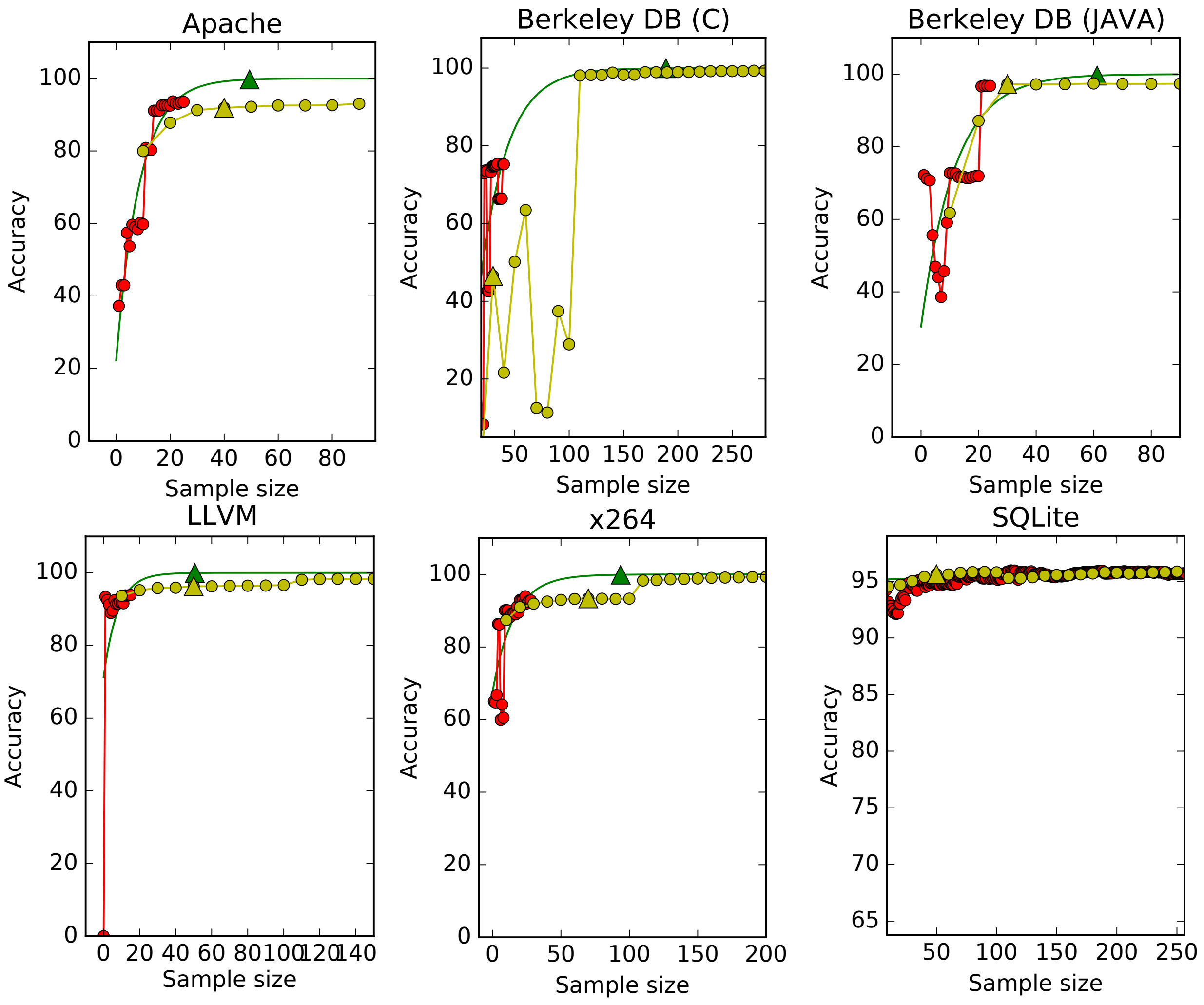


Figure 7.2: Learning curves for progressive and projective sampling. ●—● - Progressive sampling; ___ - Projective sampling (exponential curve); ▲,▲ - Optimal sample size (n^*); ● - Initial data-points for projection

7.2 RQ2: Comparison of Learning Curves

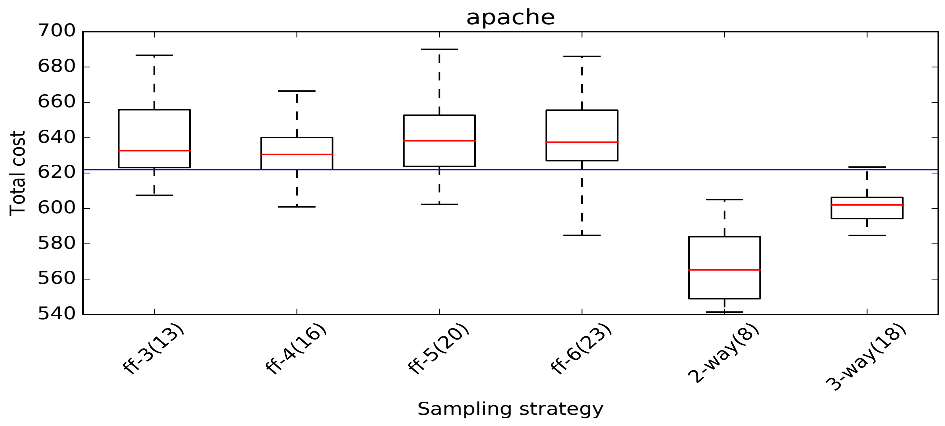
Figure 7.2 shows the learning curves for each of the six subject systems generated by progressive and projective sampling. Although, for evaluation we had run the sampling process 100 times, this figure shows the curve for a randomly selected run. The red circles denote the initial learning curve sample points (Λ_δ), generated using the feature-frequency heuristic. The yellow curve shows the actual learning curve of the system, and the green curve shows the projected one.

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 4.1. 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. 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 4.

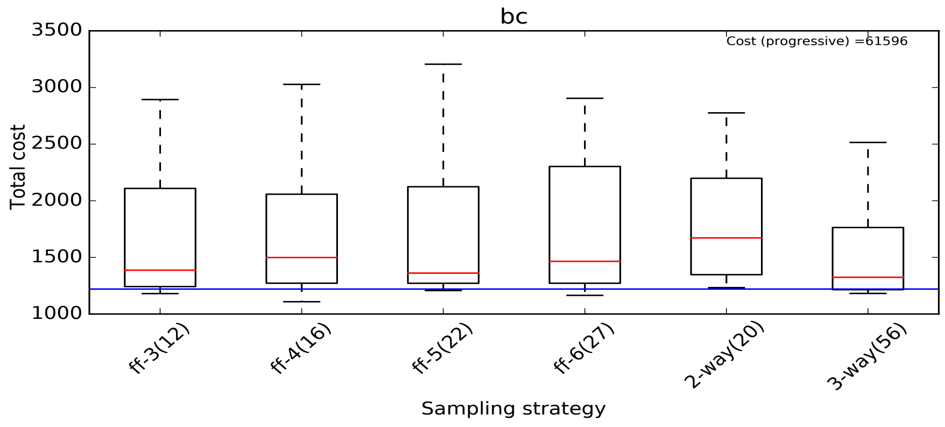
We can see from Figure 7.2 that exponential functions are the most robust among all the projective functions when used to fit the learning curve - they were the chosen class for all 6 systems. Although the figure shows the results of a single run, as we would see later in section 9, this is in tune with our overall findings. 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 DB (C). In contrast, projective sampling generates realistic values of the *optimal* sample set, which are high in accuracy as well as cost efficient.

Table 7.3: Size of Λ_δ (t-way vs. feature-frequency)

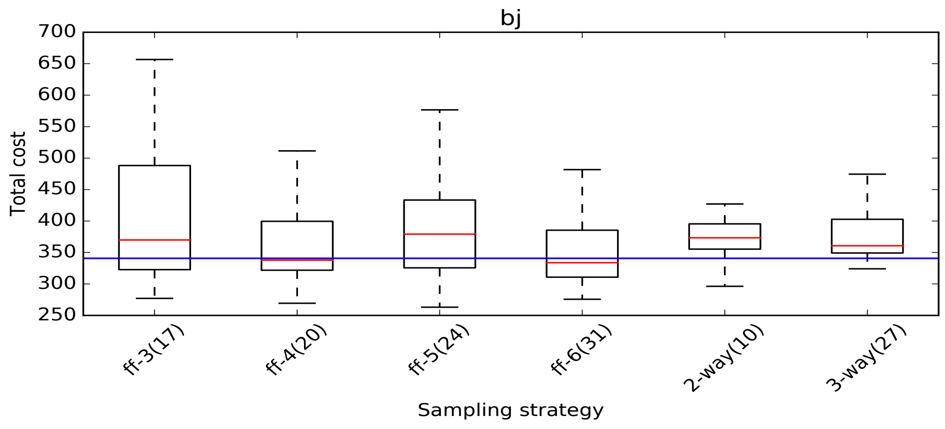
		Apache	Berkeley DB (C)	Berkeley DB (Java)	LLVM	SQLite	x264
t-way	2-way	8	20	10	10	21	12
	3-way	18	56	27	17	60	29
feature frequency	ff=3	13	12	17	12	450	14
	ff=4	16	16	20	14	522	19
	ff=5	20	22	24	18	671	21
	ff=6	23	27	31	20	803	27



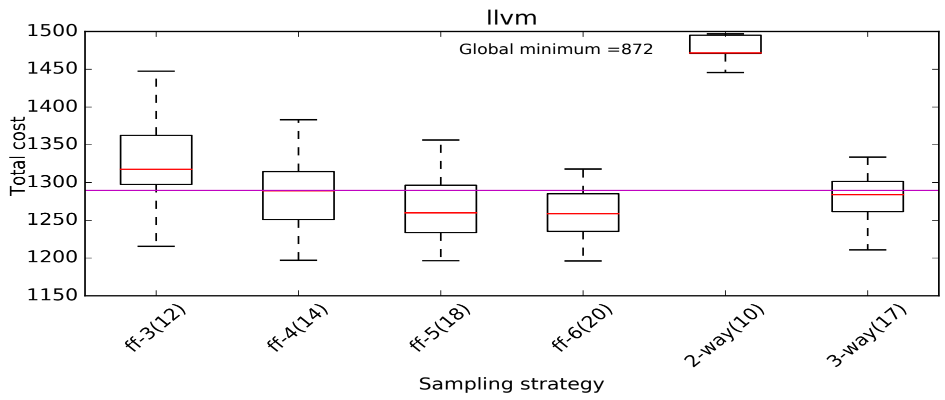
(a) Apache



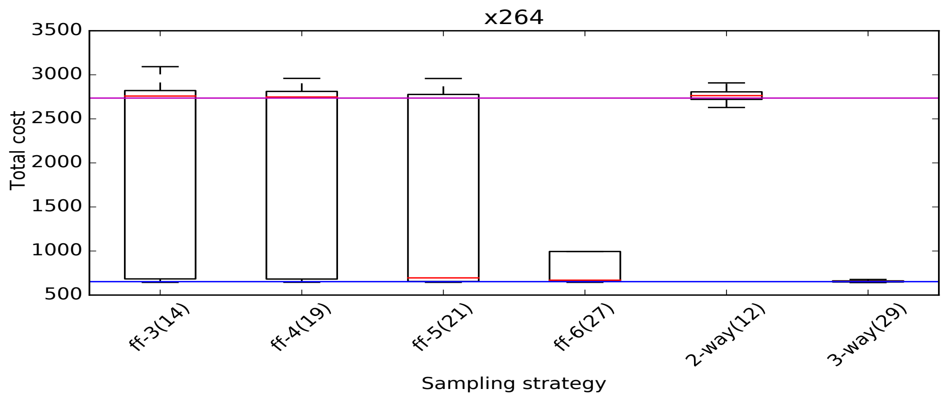
(b) Berkeley DB C



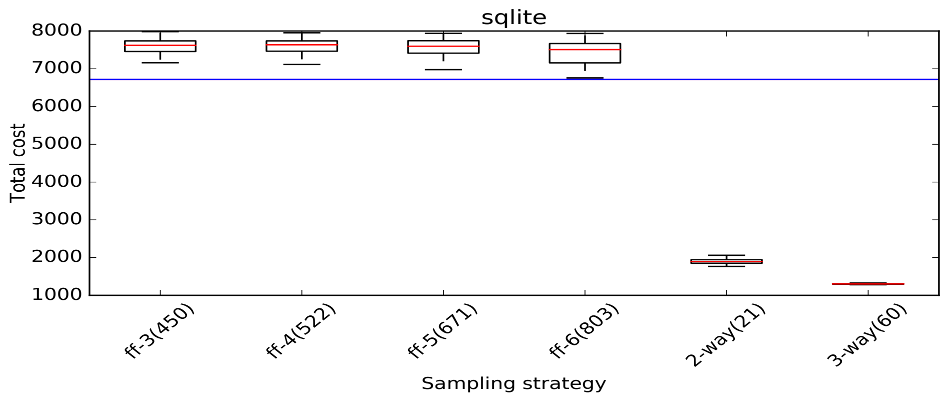
(c) Berkeley DB JAVA



(d) LLVM



(e) x264



(f) SQLite

Figure 7.2: Total cost of prediction calculated by different sampling strategies (100 runs). ▬ - Global minimum cost of prediction; ▬ - Minimum total cost calculated by progressive sampling

7.3 RQ3: T-Way vs. Feature Frequency

In this section, we compare the t-way heuristic to select the initial sample set, which is widely used in combinatorial [19] and product-line testing [11], to our proposal of a feature-frequency heuristic. We use the tool JENNY to generate 2-way and 3-way feature-coverage configurations used as the initial sample points Λ_δ[1], and compare it to feature-frequency sampling with *thresh_-freq* 3 to 6 (*ff*=3,4,5,6).

Figure 7.2 compares feature-frequency (ff=3,4,5,6) sampling to t-way (t=2,3) with respect to total cost of prediction, after 100 runs of the sampling process. For reference, the figures also shows the global minimum cost (blue line) and cost estimated through progressive sampling (magenta line). The size of the Λ_δ for each sampling strategy, is indicated in braces.

As shown in table 7.3, the size of Λ_δ is smaller for 2-way sampling compared to 3-way, since the number of samples to achieve t-way coverage increases exponentially to the strength of coverage (t). For most system (apart from SQLite), the size of Λ_δ for feature-frequency, is comparable to 2-way and 3-way sampling. This enables us to control for the size of Λ_δ while evaluating the cost estimation of the strategies. The only exception is SQLite, where the size generated by feature frequency (450-803) is significantly greater than t-way (21-60). This difference throws light on the drawback of t-way sampling for large systems. Lower strength t-way might not be able to generate a Λ_δ set which is an accurate estimator of the projected curve, like we see in the case on SQLite. To remediate this, a higher strength t-way can be used, which comes with the trade-off of additional computation complexity. For example, most tools for generating t-way coverage support upto t=6 strength [36]. Feature-frequence, on the other hand relies on random sampling, which is computationally cheaper. This enables it to scale efficiently for large systems like SQLite.

In term of total cost of sampling, feature-frequency's estimate of cost is closer to the global minimum for 5 out of 6 systems. The exception being Berkeley DB (C), where 3-way produces a better estimate. Even then, feature frequency is a close second with a total cost of 1372, compared to 3-way's 1342. Similar to our previous evaluation, the cost ratio was set to 1. Section 8 compares the accuracy of cost estimation of the sampling strategies on a broader range of R values.

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

Chapter 8

Sensitivity analysis

Sensitivity analysis (SA) is broadly defined as the investigation of potential changes in a multi-dimensional model's output with respect to variation in its input parameter values [26]. For the purpose of sensitivity analysis, a model can simply be considered as a mapping between the input and output parameters. If we denote $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n$ as the input parameters and $\mathbf{y}_1, \mathbf{y}_2, ...\mathbf{y}_d$ as the output values, then the model M can be represented as

$$(\mathbf{y}_1, \mathbf{y}_2, ...\mathbf{y}_d) = M(\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n)$$

Essentially, sensitivity analysis gives us a better understanding of the model, by reducing uncertainty in the model's response. Thus, decision makers use this as a powerful tool to evaluate the stability and utility of the model.

The first step in sensitivity analysis (SA) is identifying the input factors (including parameters and their respective values), model and the model output [28]. For a general performance prediction process, the entire process described in section 1.1, should be considered to be the model M we want to study for sensitivity analysis. However, in our current analysis we find it reasonable to restrict the model under study to the cost model of equation 3.2. This restriction of the model helps in eliminating two factors from the SA process : the prediction model (CART, SVM etc) and the sampling strategy (projective, progressive). Since, existing literature already contain extensive exploration of the effects of these two factors on the overall prediction process [13] [32] [34], we consider it worthwhile to conduct a systematic SA on the prediction cost model instead. This choice is also driven by our aim to help practitioners reduce uncertainty in the cost model.

We determine the input factors of the cost model that needs to be studied for SA from equation 3.2. They are the multiplier (θ), score set size ($|S|$), cost ratio (R), all of which appear as parameters in the cost equation. We consider the output $\mathbf{y}$ of the model to be the total cost of prediction process. In tune with our overall approach, since accuracy is already incorporated in the total cost equation, we consider total cost as a more accurate reflection of the utility of performance prediction.

8.1 Design

Once the input factors, model and output are identified, the next crucial step is to state up front the questions that needs to be answered through sensitivity analysis [10]. To that end, we ask the following.

- How sensitive is cost and accuracy of prediction to variations in cost ratio (R). i.e ratio of the cost of measurement-error to cost of measuring sample configuration.

- How sensitive is cost and accuracy of prediction to variations in training-testing sample size split.

Sensitivity analysis techniques are broadly categorized into *global* and *local* methods. Local methods study the sensitivity of model outputs to perturbations to a given input parameter value, around a *point of interest*. Global methods, on the other hand, measure sensitivity over the entire range of each input parameter. This is often achieved with the help of prior knowledge about its probability distribution.
Our SA experiments follow a local strategy because we are interested in studying model output around a point of interest. For example, for the cost ratio, in absence of any prior information, we take the case when the cost of measurement is equal to the cost of prediction error ($R = 1$), as the point of interest. Also, the fact that we cannot assume an underlying probability distribution of our model parameters, precludes any attempt at a *global* sensitivity analysis.

Relating the questions we want to study, and the factors in the cost model of equation 3.2, we end up with the 3 factors for our sensitivity analysis : cost ratio (R), multiplier (θ) and the size of the score set ($|S|$). For a given sample system, the size of the score set is a fixed value, which can be pre-determined using a model counter [8]. Effectively, this means that for a given system, considering the score set as one of the factors for SA and varying its value, has no practical relevance. Thus, we exclude this factor from our analysis.

Our design of sensitivity analysis uses a One-At-a-Time (OAT) design strategy for local sensitivity analysis and a quasi-random sequence for parameter valuations. We briefly explain each of these design strategies.

8.1.1 One-At-a-Time (OAT) SA method

One-At-a-Time (OAT) is one of the simplest technique for analyzing local sensitivity around a base value (point of interest) for an input parameter. A chosen parameter is given a local perturbation in the vicinity of the base value ($\mathbf{x}^0$), and the corresponding change in model output is noted. The local perturbation can either be positive ($\mathbf{x}^+$) or negative ($\mathbf{x}^-$).

Table 8.1: OAT sensitivity analysis design

	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$
0	x_1^0	x_2^0	x_3^0
1	x_1^+	x_2^0	x_3^0
2	x_1^-	x_2^0	x_3^0
3	x_1^0	x_2^+	x_3^0
4	x_1^0	x_2^-	x_3^0
5	x_1^0	x_2^0	x_3^+
6	x_1^0	x_2^0	x_3^-

Table 8.1 shows the 7 runs of the model that would be required for a model of 3 factors ($\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$). The first run (0) of the model is with the base value of each factor. In subsequent runs (1-7), a single parameter is give a positive or negative perturbation, keeping the rest of the factors at their base value.

To analyze the local sensitivity to the perturbation, we use the *finite-difference approximation* method, where observed changes in the model output is calculated using the following equation [28].

$$\frac{\partial M}{\partial \mathbf{x}_i} = \frac{M(\mathbf{x}_i + \Delta \mathbf{x}_i) - M(\mathbf{x}_i)}{\Delta \mathbf{x}_i}$$

Where $\mathbf{x}_i : i \in 1, 2, ..n$ are the factors of the model and $\Delta \mathbf{x}_i = \mathbf{x}_i^{+|-} - \mathbf{x}_i$, is the local perturbation to the factor. For accuracy of analysis, size of $\Delta \mathbf{x}_i$ typically ranges within $\pm 10\%$ of $\mathbf{x}_i$. In cases where SA deals with calculating slope, or rate of change of the model output, this guideline helps in preserving local linearity of the model. In fact, this condition of linearity often acts as a precondition for local sensitivity analysis [37]. As we would show in the subsequent section, this condition is preserved for all the factors of the cost model.

One of the drawbacks of using OAT method for local sensitivity analysis, is that it cannot capture any interaction that may occur between the parameters. However, in our case, we believe that this is a reasonable trade-off since the two factors; cost-ratio and the multiplier have minimal scope for any interaction.

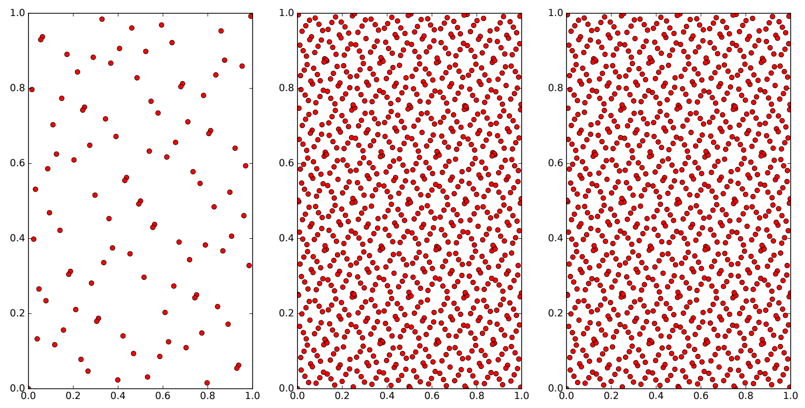
8.1.2 Parameter valuation:

For each factor in our analysis, we set the value of local perturbation as follows:

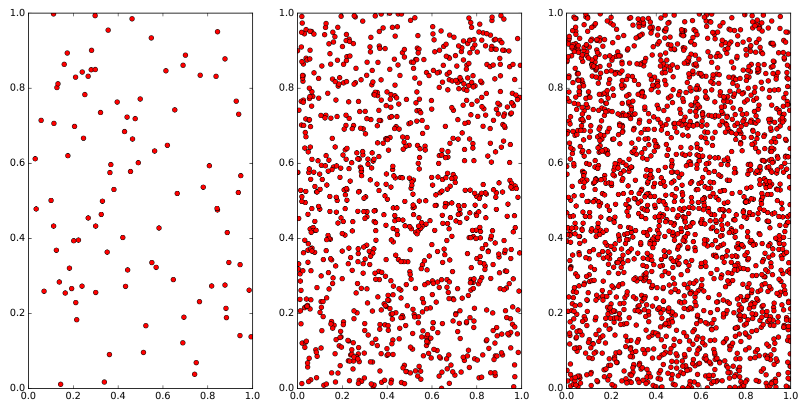
- Cost ratio : $R^0 = 1; R^+ = 10; R^- = 0.1$
- Multiplier : $\theta^0 = 2; \theta^+ = 3$

To further investigate model response, often sampling based techniques can be combined with local SA, by generating a set of sample parameter values in the perturbation interval. In our case, for both parameters, we use Sobol sampling to generate a set of parameter values within the intervals [0.1,1],[1,10] for R and [2,3] for θ.

Sobol sequence: Sobol sequence is an efficient technique for generating low-discrepancy sequence of quasi-random numbers. The advantage of using Sobol sequence is that the random numbers generated are equi-distributed in the given interval. This prevents the numbers from forming a cluster, as often observed in typical pseudo-random number generators. Sobol sequence is widely used for sensitivity analysis, and our choice was also influenced by our aim to achieve an even coverage of the values on the perturbation interval. Figure 8.1 shows a visual comparison of two dimensional sequence of random numbers generated using Sobol sequence, and a widely used pseudo-random generator Mersenne-Twister.



(a) Pseudo-random sequence generated using Sobol



(b) Pseudo-random sequence generated using Mersenne-Twister

Figure 8.1: Sample 2-D pseudo-random sequence : 100,1000,2000 points

8.2 Results

We discuss this section with respect to our two research questions on sensitivity analysis.

Effect of cost ratio(R) on total cost and prediction accuracy

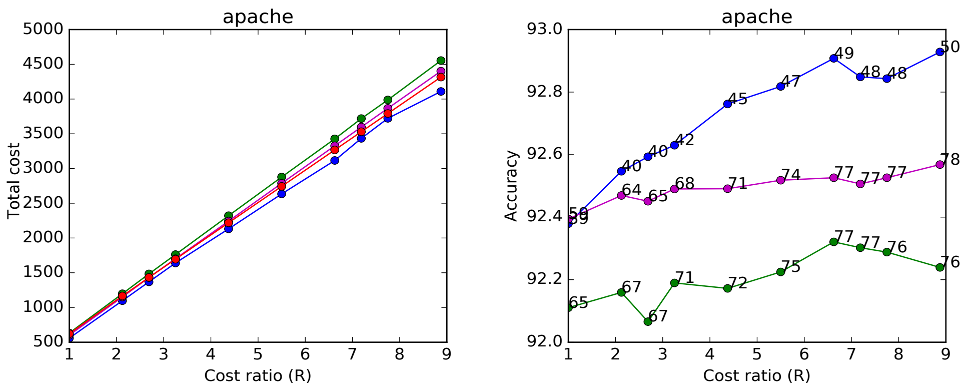
The OAT approach helps us to analyse the effect of one factor, keeping the other constant. Thus, the cost model reduces to a form of:

$$TotalCost(n^*) = c_1 \cdot n^* + \epsilon \cdot c_2 \cdot R \tag{8.1}$$

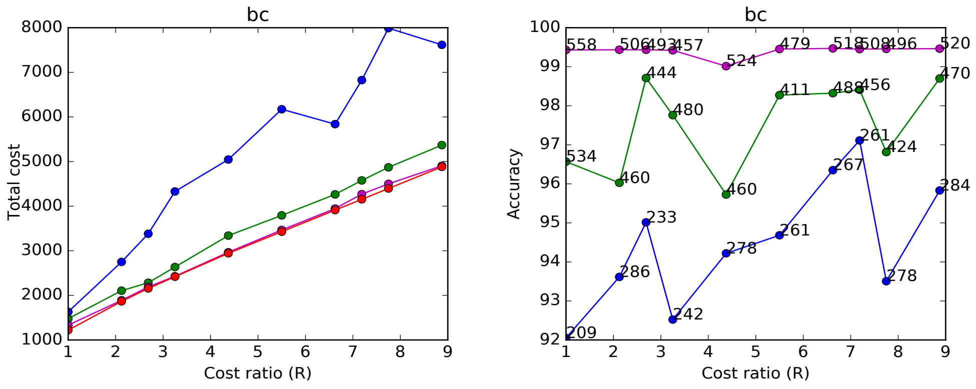
where c_1 and c_2 represent the factors that are kept constant. The equation above calculates the total cost of prediction for a system, at a certain optimal sample size. The optimal sample size (n^*) is estimated through projective sampling, as discussed in section 4.2. An increase in the cost ratio implies that the relative cost to measure configurations for training/testing sample, is lesser than the cost incurred due to error in prediction (section 3). This means that, to minimize cost, more sample measurements can be added to training/testing set, which eventually results in an increase in the optimal sample size (n^*). This in turn, also results in a decrease in the fault rate (ϵ). Thus, in other words, the overall effect of the change in the cost ratio (R), is influenced by the shape of the learning curve. Since this curve can vary from one system to the other, it is futile to analytically estimate the effect of the cost ratio changes on total cost. This leads to an empirical sensitivity analysis being the appropriate choice to study the effect of this change.

Figures 8.2,8.3 show the effect of cost ratio on total cost of prediction. The figures account for positive perturbation of R on the interval [1,10], and negative perturbation in the interval [0,1] respectively. Each point in the curve is the *median* total cost obtained after 100 runs of the sampling process, with the corresponding R value. As we observe from the results, total cost of prediction has close to linear correlation with the cost ratio for all six systems. In other words, the effect on total cost due to an increasing cost ratio and optimal sample size, dominates the decreasing effect of error rate. Table 8.2 compares the rate of change in total cost of prediction with respect to changes in the cost ratio (R). We follow the finite-difference approximation method to calculate the rate of change according to equation 8.1.1. Since the growth rate is positively correlated with the the cost ratio, all the values in table 8.2 are positive.

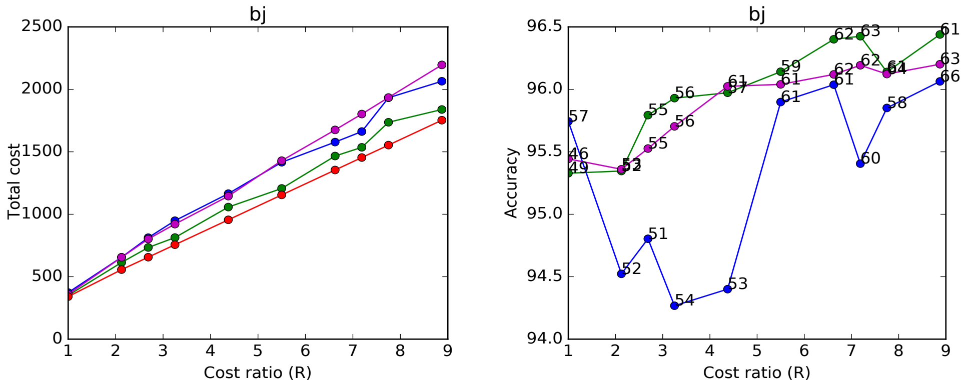
Accuracy plots of figures 8.2,8.3 show sensitivity of optimal sample size (n^*) and corresponding prediction accuracy to variations in R. As discussed earlier, increase in the cost ratio (R) results in a corresponding increase in the optimal sample size, which is evident from these figures. For a well behaved learning curve, this also results in a concomitant increase in prediction accuracy of the model. This accounts for the increasing trend, as seen in the figure, when the cost ratio increases.



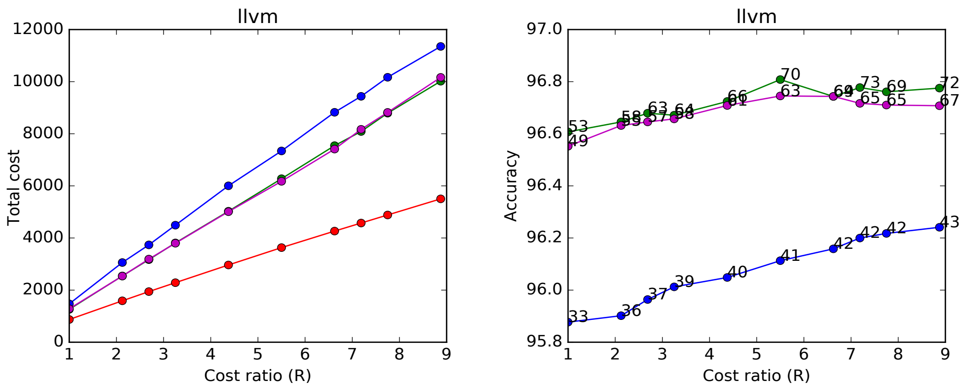
(a) Apache



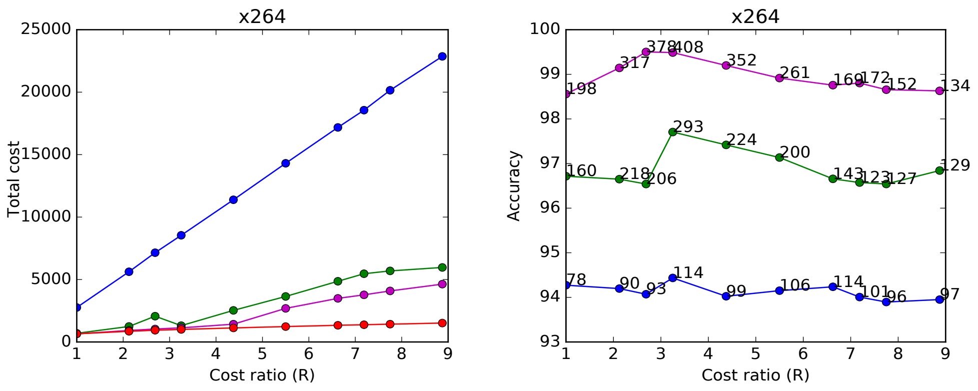
(b) Berkeley DB C



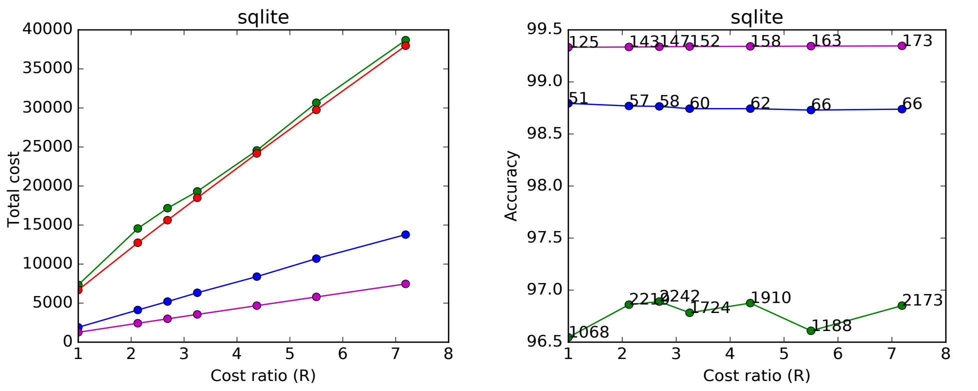
(c) Berkeley DB JAVA



(d) LLVM

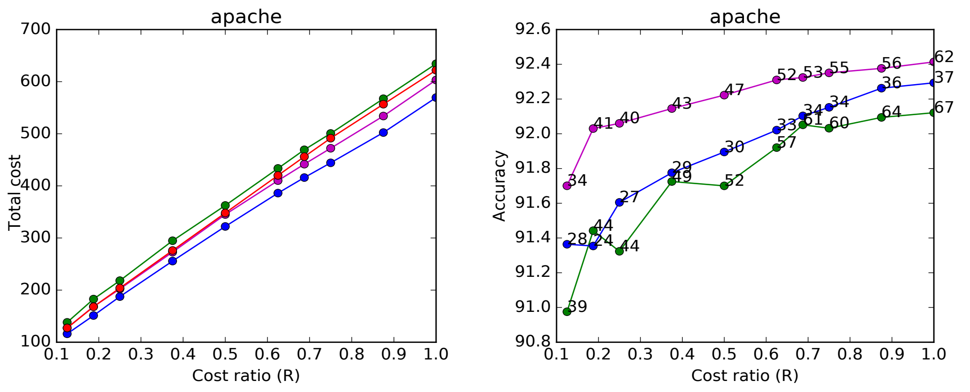


(e) x264

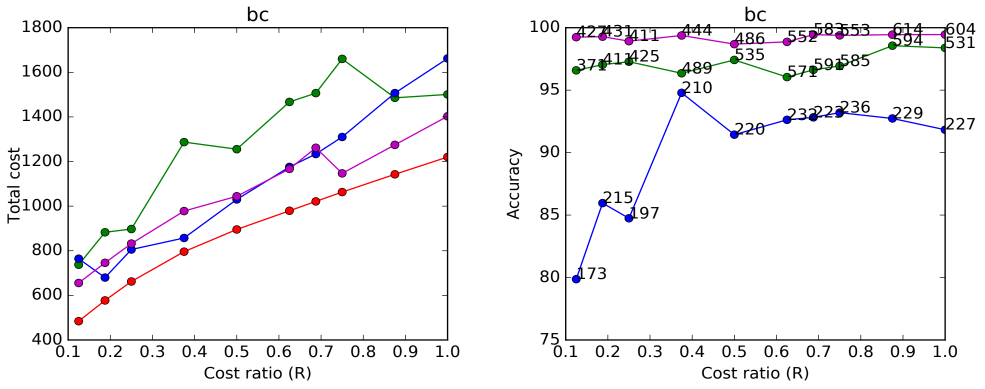


(f) SQLite

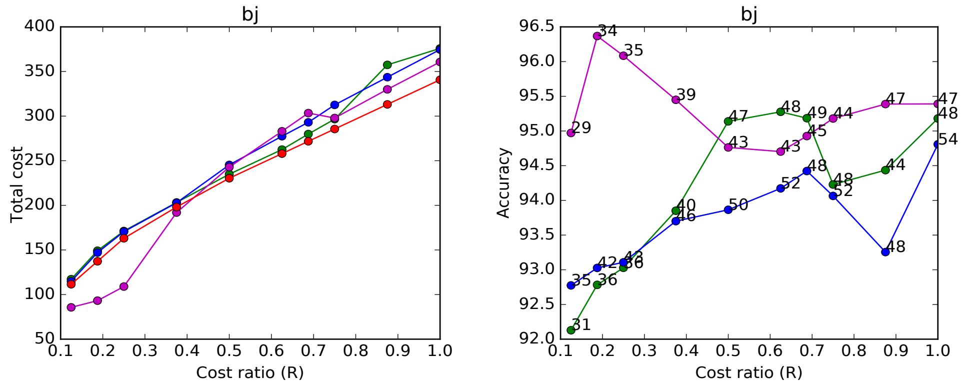
Figure 8.2: Sensitivity of total cost and accuracy to variations in R in the interval 1 to 10. ●—● - 2-way; ●—● - 3-way; ●—● - feature frequency (*ff*=5); ●—● - Actual cost ($TotalCost_{minimum}$)



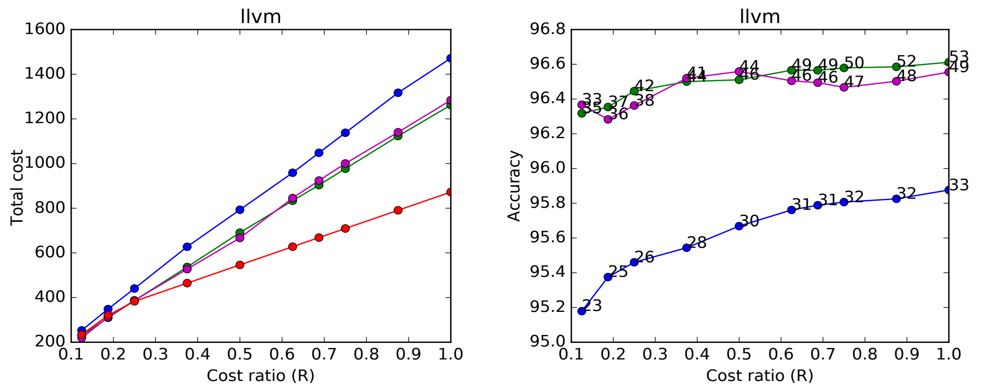
(a) Apache



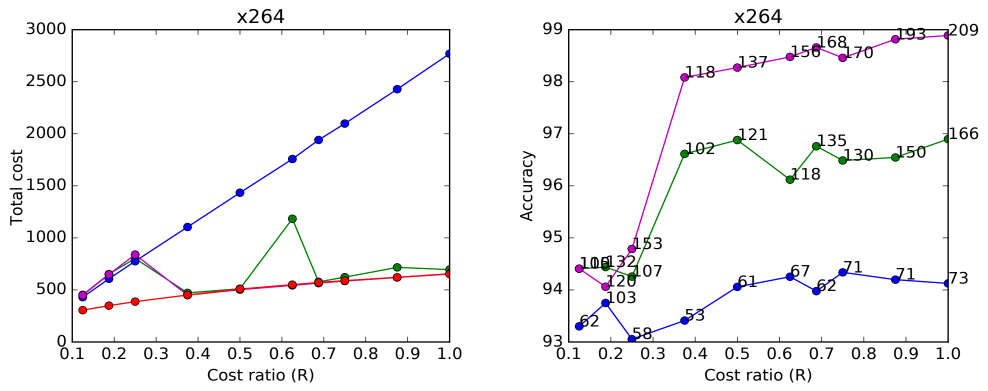
(b) Berkeley DB C



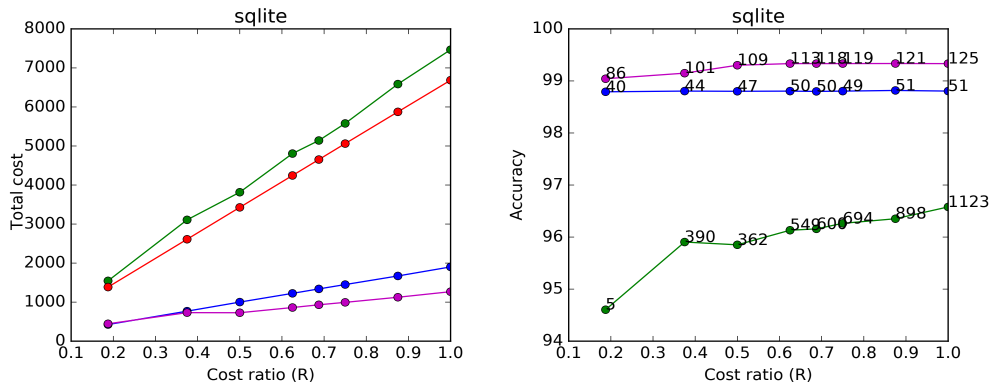
(c) Berkeley DB JAVA



(d) LLVM



(e) x264



(f) SQLite

Figure 8.3: Sensitivity of total cost and accuracy to variations in R in the interval 0 to 1. ●—● - 2-way; ●—● - 3-way; ●—● - feature frequency (*ff*=5); ●—● - Actual cost ($TotalCost_{minimum}$)

Table 8.2: Rate of change of total cost of prediction with respect to unit change in cost ratio (R)

		Apache	Berkeley DB (C)	Berkeley DB (Java)	LLVM	x264	SQLite
$R = [0, 1]$	2-way	507	1241	281	1393	2659	1844
	3-way	543	1054	312	1217	454	1105
	feature-frequency	564	1029	295	1180	389	7315
	actual	565	841	261	729	395	6555
$R = [1, 10]$	2-way	454	890	212	1254	2566	1873
	3-way	482	468	233	500	454	1039
	feature-frequency	496	506	187	1108	737	5005
	actual	469	464	179	587	109	5059

Here, it is worth revisiting that the total cost of sampling as shown in the figures 8.2,8.3, are the global minimum cost estimated by each sampling strategy (2-way, 3-way and feature-frequency). Thus, similar to section 7, we evaluate the utility of the sampling strategies by how close this estimate is to the global minimum. For a given cost ratio, the difference between the global minimum cost, and the estimated minimum cost is apparent by the distance between the red line (actual minimum) to the colored lines. For each cost ratio value, we calculate this error of cost estimation for the three sampling strategies using the formula:

$$Error_{TotalCost} = \frac{|TotalCost_{projective} - TotalCost_{minimum}|}{|Cost_{max} - Cost_{min}|} \tag{8.2}$$

Where $Cost_{max}$ and $Cost_{min}$ are the maximum and minimum total cost of prediction, for a given subject system, using different strategies. Table 8.3 shows the *mean* and *standard deviation* of $Error_{TotalCost}$ after grouping the individual cost ratios into their corresponding interval [1,10] and [0,1]. We evaluate the robustness of a sampling strategy with respect to its $Error_{TotalCost}$ using **winner probabilities** [32]. We calculate winner probability as the number of R values for which a certain sampling strategy has the minimum $Error_{TotalCost}$, divided by the total number of R valuations (120). For example, if 2-way sampling had minimum $Error_{TotalCost}$ for 10 valuations of R, the winner probability of 2-way would be 10/120 = 0.0833. Table 8.3 shows winner probabilities of the three sampling strategies. For feature-frequency, we had set the *freq_threshold* to 5. We can see that feature-frequency (*ff*=5) is the most robust sampling strategy for different values of R, with a winner probability slightly higher than 3-way sampling.

Table 8.3: Mean relative error and standard deviation of cost estimation

(a) A

		Apache	Berkeley DB (C)	Berkeley DB (Java)	LLVM	x264	SQLite
$R = [0, 1]$	2-way	0.063 ± 0.031	0.183 ± 0.108	0.054 ± 0.029	0.221 ± 0.158	0.422 ± 0.261	0.364 ± 0.184
	3-way	0.016 ± 0.014	0.181 ± 0.096	0.101 ± 0.058	0.139 ± 0.114	0.038 ± 0.062	0.400 ± 0.215
	feature-frequency	0.025 ± 0.004	0.248 ± 0.083	0.046 ± 0.029	0.133 ± 0.107	0.043 ± 0.0531	0.062 ± 0.029
$R = [1, 10]$	2-way	0.020 ± 0.009	0.285 ± 0.176	0.101 ± 0.040	0.317 ± 0.160	0.524 ± 0.276	0.389 ± 0.181
	3-way	0.010 ± 0.007	0.009 ± 0.006	0.131 ± 0.069	0.227 ± 0.125	0.055 ± 0.052	0.471 ± 0.229
	feature-frequency	0.029 ± 0.017	0.038 ± 0.017	0.034 ± 0.0267	0.231 ± 0.125	0.100 ± 0.078	0.021 ± 0.009

(b) Winner probability of sampling strategies

	2-way	3-way	feature-frequency (5)
Winner probability	0.0833	0.4333	0.4833

Effect of training-testing split on cost of prediction and accuracy

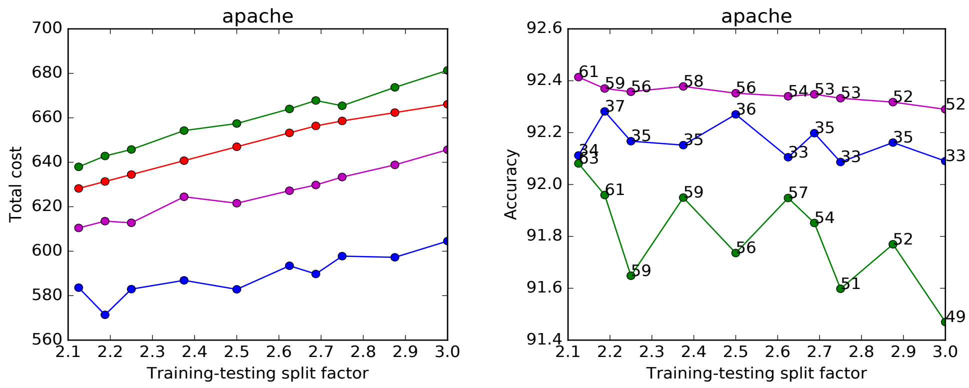
In accordance with our OAT SA design, we analyze the model sensitivity to the second factor i.e training-testing split. As discussed in section 3, the multiplier (θ) accounts for this split. Thus, keeping the rest of the factors constant, the cost model reduces to the form:

$$TotalCost(n^*) = \theta \cdot n^* + \epsilon \cdot c_1 \tag{8.3}$$

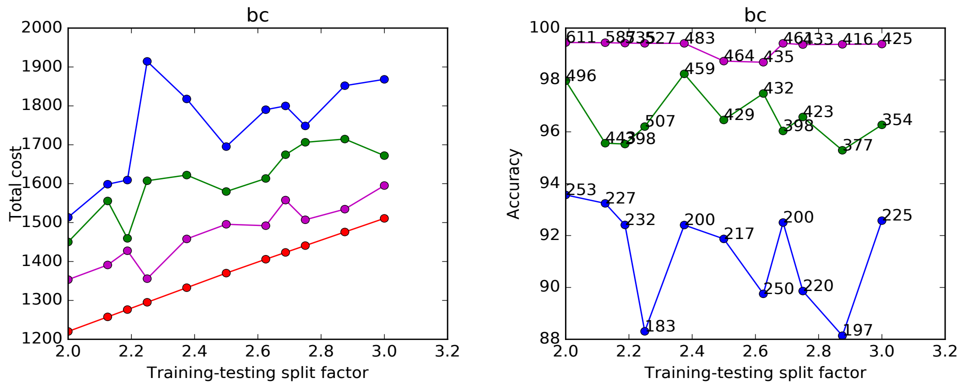
Where c_1 represents the invariant factors in this analysis (cost factor R and score set $|S|$). We only consider positive perturbations to the multiplier (θ) in the range [2,3]. A negative perturbation (or $\theta < 2$) would make the size of testing set lower than the training set size, which is not an acceptable practice in data analysis [14]. Figure 8.4 shows the sensitivity of total cost of prediction and prediction accuracy with respect to changes to θ in the range [2,3]. As we can see from the figure, apart from the Berkeley DB (C and JAVA), all other systems show only marginal increase in total cost. This is in contrast to R, where the rate of increase was much higher. Effect on prediction accuracy is also much subtler. With an increase in *multiplier*, since the size of the testing set is growing, justified pessimism would expect a decrease in prediction accuracy. However, as we see from the figure, the prediction accuracy maintains its stability in the multiplier range.

A key difference between SA analysis of θ and R lies in their influence on optimal sample size (n^*). Varying the values of the cost factor (R) preserves the shape of the learning curve, which helps us judge its influence on the optimal sample size. However, changes to θ, have an effect on the model prediction accuracy, and thus the learning curve of the system is not preserved. This prevents us from deducing a conclusive trend in the optimal sample size, like we were able to for the cost ratio (R).

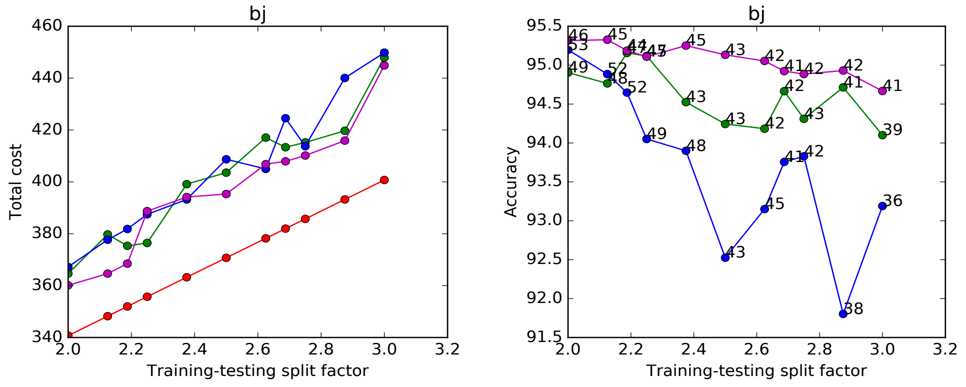
(a) Apache

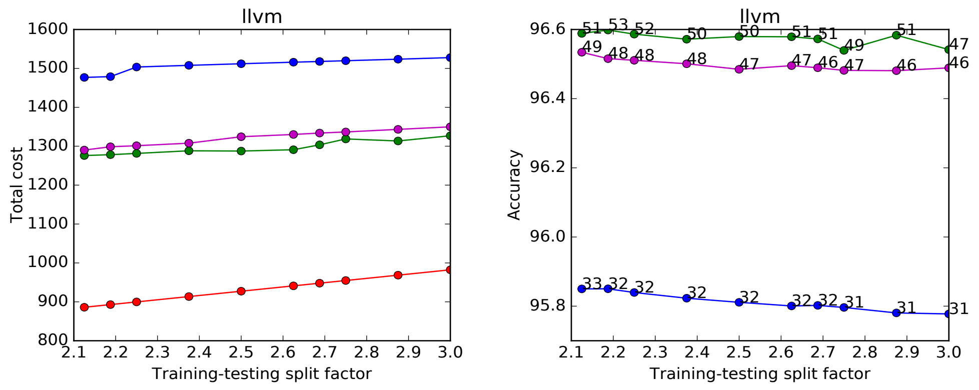


(b) Berkeley DB C

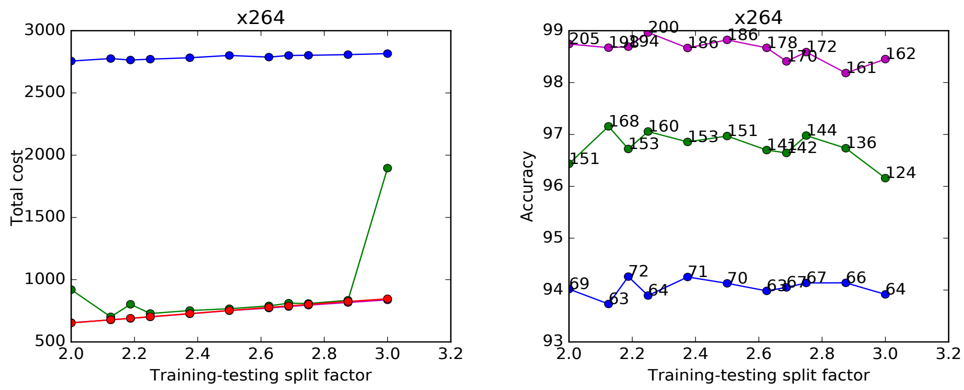


(c) Berkeley DB JAVA

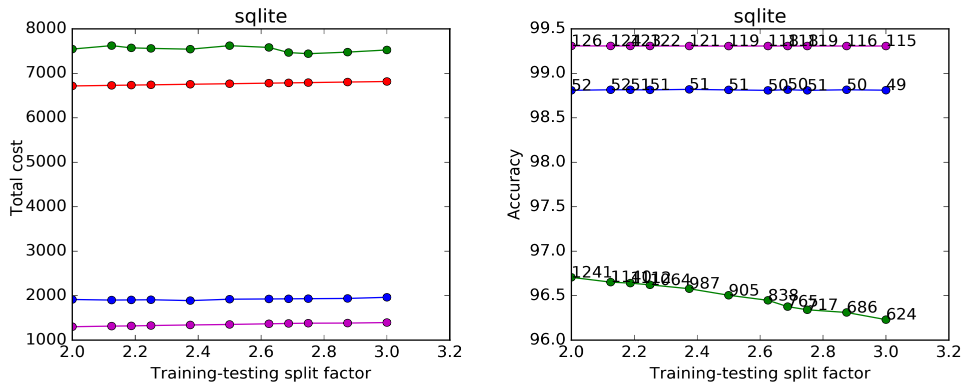




(d) LLVM



(e) x264



(f) SQLite

Figure 8.4: Sensitivity of total cost and accuracy to variantions in θ in the interval 2 to 3. ●—● - 2-way; ●—● - 3-way; ●—● - feature frequency (*ff*=5); ●—● - Actual cost ($TotalCost_{minimum}$)

Chapter 9

Stability of learning curves

We have used four classes of learning curves as listed in table 4.1, to model the learning behavior of a system. These learning curves map a training sample size with the estimated prediction accuracy. The class of learning curve that shows highest correlation with the sampled data-points' prediction accuracy, is selected as the chosen curve, with the following restriction:

- The optimal sample size generated by the chosen curve is less than the total number of configurations of the system.

- The *pearson* correlation-coefficient with the data-points is negative (when curve equations express prediction error) or positive (when they express prediction accuracy).

If the chosen curve violates the first condition, the curve with the next highest correlation value is chosen. Violation of the second condition indicates a less than ideal learning behavior, thus rendering all four classes of learning curves ineffective. However, as we would see further, the likelihood of this violation is minimal.

In this section we investigate the stability of curve selection, both with regards to different sampling strategies and multiple runs of the algorithm. Specifically we are interested in answering the following research questions:

- Is the learning behavior of a system sensitive to sample quality.

- Is the learning behavior of a system sensitive to different sampling strategies i.e 2-way,3-way and feature-frequency.

For the first part of our analysis, we execute multiple runs of our algorithm with feature-frequency sampling (100 runs; *ff*=5). To ensure that the sample data-points used to study the learning behavior is qualitatively different, we use unique seed for each run. The cumulative

Table 9.1: Distribution of curve selection vs random (no. of runs = 100)

	Feature-frequency				**Random**				*p-value*	*success %*
	Exponential	Weiss-Tan	Logarithmic	Power	Exponential	Weiss-Tan	Logarithmic	Power		
Apache	69	1	-	25	25	38	32	-	<0.001	95
Berkeley DB (C)	55	4	4	19	18	21	23	20	<0.001	82
Berkeley DB (Java)	72	5	-	11	33	29	-	26	<0.001	88
LLVM	100	-	-	-	-	-	-	-	-	100
x264	39	-	-	61	60	-	-	40	<0.001	100
SQLite	99	-	-	-	-	-	-	-	-	99

distribution of the selected curve is then compared to a random distribution using chi-square *goodness of fit* test.

Chi-square goodness of fit test is applied when the variable under study is categorical, and it helps to determine if the sample data is consistent with the hypothesized distribution [29]. In our case, the the chosen curve acts as the categorical variable, the population being the independent runs of the experiment. With this information, we formulate the following *null* hypothesis:

H_0 : The distribution of curve selection for feature-frequency sampling is consistent with random distribution

Table 9.1 shows the comparison of the distribution, and the corresponding *p-value*. For each group (*feature-frequency* and *random*), the numbers represent how many times the corresponding curve was selected, out of 100 runs. The *success %* is the percentage of runs where our algorithm was able to select a learning curve. For LLVM and SQLite, we do not compare the distributions, since the same curve (exponential) was selected for all the runs. The *p-value* for all systems is less than 0.001, thus even with conservative significance levels, we can reject the null hypothesis. Thus, we see from this analysis that a) the learning behavior is independent of sample quality and b) exponential curves are the most robust class of learning curves, that can model the learning behavior of our subject systems.

For the second part of our analysis, we use goodness-of-fit test to investigate whether the learning behavior is sensitive to different sampling strategies. We study this by selecting a sampling strategy (*ff*=5) as the reference distribution, and comparing it with 2-way,3-way and *ff*=3,4,6 strategies. Table 9.3 shows the goodness of fit χ^2 and $p-values$ for our subject systems when different sampling strategies are compared to ff=5 distribution, using *chi-square* goodness-of-fit. The table also lists the size of Λ_δ set for each sampling strategy within braces. Statistically significant *p values* at significance level 0.01 are marked in bold. We observe from this table that the distribution of curve selection for 2-way sampling is different from 3-way and feature-frequency, as indicated by their low *p-values*. However, the distribution doesn't change as much between feature-frequency and 3-way.

Table 9.2: Distribution of curve selection with respect to different sampling strategies (100 runs)

		Exponential	Weiss-Tan	Logarithmic	Power
feature-frequency	Apache	69	1	-	25
	Berkeley DB (C)	55	4	4	19
	Berkeley DB (JAVA)	72	5	-	11
	LLVM	100	-	-	-
	x264	39	-	-	61
	SQLite	99	-	-	-
2-way	Apache	94	-	-	6
	Berkeley DB (C)	85	1	2	7
	Berkeley DB (JAVA)	53	9	-	21
	LLVM	100	-	-	-
	x264	83		-	17
	SQLite	100	-	-	-
3-way	Apache	79	-	-	21
	Berkeley DB (C)	76	-	-	21
	Berkeley DB (JAVA)	87	4	-	8
	LLVM	100	-	-	-
	x264	13	-	-	87
	SQLite	100	-	-	-

Table 9.3: Sensitivity of learning curves to sampling strategies

	Apache	Berkeley DB (C)	Berkeley DB (JAVA)	LLVM	x264	SQLite
2-way	χ^2 =66.81, p =2.98e-16 (8)	χ^2 =51.04 p =4.78e-11 (12)	χ^2 =11.78 p =0.002 (10)	-	χ^2 =137.20 p =1.0e-31 (11)	-
3-way	χ^2 =2.02 p =**0.154** (18)	χ^2 =0.717 p =**0.396** (27)	χ^2 =2.860 p =**0.239** (18)	-	χ^2 =59.77 p =1.06e-14 (29) -	
ff=3	χ^2 =3.76, p =**0.052** (13)	χ^2 =5.42 p =**0.143** (13)	χ^2 =4.66 p =0.**096** (16)	-	χ^2 =24.21 p =8.62e-07 (15)	-
ff=4	χ^2 =14.20, p =0.00016 (16)	χ^2 =5.68 p =**0.127** (16)	χ^2 =6.52 p =**0.038**(20)	-	χ^2 =13.61 p =0.0002 (19)	-
ff=5	(19)	(22)	(24)	-	(23)	-
ff=6	χ^2 =3.20, p =**0.0735** (22)	χ^2 =4.57 p =**0.206** (27)	χ^2 =16.98 p =0.0002 (28)	-	χ^2 =0.042 p =**0.837** (27)	-

Among the four classes of learning curves, we see that exponential functions are robust, even across different sampling strategies (Figure 9.2).

Chapter 10

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 100 times for training and testing the prediction models, and we reported median and percentile results for the evaluation metrics, such as sampling cost, prediction accuracy. 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 conducted a systematic sensitivity analysis of our cost model, to the changes in the cost ratio R.

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.

Chapter 11

Related Work

11.0.1 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 [6]. 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 [13] 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. [38] 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. [34] 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. [31] used a measurement-based technique to predict performance by detecting feature interactions. In follow-up work [30], 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.

11.0.2 Sampling Strategies

Provost et al. [27] 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 [33] 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 [20] used the cost function proposed by Weiss and Tian [33] 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.

Chapter 12

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 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 presented approach, empirical findings, and sensitivity analysis, are meant to help stakeholders in designing effective sampling strategies for performance prediction.

References

[1] Compuware. applied performance management survey (october 2006). http://www.cnetdirectintl.com/direct%20/compuware/OvumAPM/APMSurveyReport.pdf. Accessed: 2015-03-23.

[2] Jenny tool. http://burtleburtle.net/bob/math/jenny.html. Accessed: 2015-03-23.

[3] Spec benchmark suite. http://www.spec.org. Accessed: 2015-03-23.

[4] Sqllite. https://sqlite.org/. Accessed: 2015-03-23.

[5] Sven Apel and Christian Kästner. An overview of feature-oriented software development. *Journal of Object Technology*, 8(5):49–84, 2009.

[6] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[7] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. What is a feature?: a qualitative study of features in industrial software product lines. In *Proceedings of the 19th International Conference on Software Product Line*, pages 16–25. ACM, 2015.

[8] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[9] Barry W Boehm. *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ), 1981.

[10] Emanuele Borgonovo and Elmar Plischke. Sensitivity analysis: A review of recent advances. *European Journal of Operational Research*, 248(3):869–887, 2016.

[11] Myra B Cohen, Matthew B Dwyer, and Jiangfan Shi. Coverage and adequacy in software product line testing. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 53–63. ACM, 2006.

[12] Lewis J Frey and DH Fisher. Modeling decision tree performance with the power law. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pages 59–65, 1999.

[13] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. Variability-aware performance prediction: A statistical learning approach. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 301–311. IEEE, 2013.

[14] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Elsevier, 2011.

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

[16] George H John and Pat Langley. Static versus dynamic sampling for data mining. In *Proceedings of the Conference on Knowledge Discovery and Data mining (KDD)*, pages 367–370. ACM, 1996.

[17] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Perform. Eval.*, 67(8):634–658, August 2010.

[18] D Richard Kuhn, Raghu N Kacker, and Yu Lei. Practical combinatorial testing. *NIST Special Publication*, 800(142):142, 2010.

[19] Rick Kuhn, Yu Lei, and Raghu Kacker. Practical combinatorial testing: Beyond pairwise. *IT Professional*, 10(3):19–23, 2008.

[20] Mark Last. Improving data mining utility with projective sampling. In *Proceedings of the Conference on Knowledge Discovery and Data mining (KDD)*, pages 487–496. ACM, 2009.

[21] Aleksandar Lazarevic and Zoran Obradovic. Data reduction using multiple models integration. In *Principles of Data Mining and Knowledge Discovery*, pages 301–313. Springer, 2001.

[22] Rui Leite and Pavel Brazdil. Improving progressive sampling via meta-learning on learning curves. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 250–261. Springer, 2004.

[23] Huan Liu and Hiroshi Motoda. On issues of instance selection. *Data Mining and Knowledge Discovery*, 6(2):115–130, 2002.

[24] Douglas C Montgomery, George C Runger, and Norma F Hubele. *Engineering statistics*. John Wiley & Sons, 2009.

[25] Changhai Nie and Hareton Leung. A survey of combinatorial testing. *ACM Computing Surveys (CSUR)*, 43(2):11, 2011.

[26] David J Pannell. Sensitivity analysis: strategies, methods, concepts, examples. *Agric Econ*, 16:139–152, 1997.

[27] Foster Provost, David Jensen, and Tim Oates. Efficient progressive sampling. In *Proceedings of the Conference on Knowledge Discovery and Data mining (KDD)*, pages 23–32. ACM, 1999.

[28] Andrea Saltelli, Karen Chan, E Marian Scott, et al. *Sensitivity analysis*, volume 1. Wiley New York, 2000.

[29] David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[30] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2015.

[31] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 167–177. IEEE, 2012.

[32] Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. Empirical comparison of regression methods for variability-aware performance prediction. In *Proceedings of the 19th International Conference on Software Product Line*, pages 186–190. ACM, 2015.

[33] GaryM. Weiss and Ye Tian. Maximizing classifier utility when there are data acquisition and modeling costs. *Data Mining and Knowledge Discovery*, 17(2):253–282, 2008.

[34] Dennis Westermann, Jens Happe, Rouven Krebs, and Roozbeh Farahbod. Automated inference of goal-oriented performance prediction functions. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 190–199. IEEE, 2012.

[35] Cemal Yilmaz, Arvind S Krishna, Atif Memon, Adam Porter, Douglas C Schmidt, Aniruddha Gokhale, and Balachandran 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)*, pages 293–302. IEEE, 2005.

[36] Linbin Yu, Yu Lei, Raghu N Kacker, and D Rick Kuhn. Acts: A combinatorial test generation tool. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 370–375. IEEE, 2013.

[37] X-Y Zhang, MN Trame, LJ Lesko, and S Schmidt. Sobol sensitivity analysis: A tool to guide the development and evaluation of systems pharmacology models. *CPT: Pharmacometrics & Systems Pharmacology*, 4(2):69–79, 2015.

[38] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. Performance prediction of configurable software systems by Fourier learning. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 2015.

APPENDICES

A.1 Derivation of optimal sample size (n^*)

To solve for optimal sample size, $\frac{d\,TotalCost(n)}{dn} = 0$

A.1.1 Logarithmic

$$
\begin{aligned}
\frac{d\,TotalCost(n)}{dn} &= 0 \\
\frac{d\,(\theta \cdot n + \epsilon_n \cdot |S| \cdot R)}{dn} &= 0 \\
\frac{d\,(\theta \cdot n + (a + b \cdot \log(n)) \cdot |S| \cdot R)}{dn} &= 0 \\
\frac{d\,\theta}{dn} + \frac{d}{dn} a \cdot |S| \cdot R + b \cdot |S| \cdot R \frac{d}{dn} \log(n) &= 0 \\
\theta + \frac{b \cdot |S| \cdot R}{n} &= 0 \\
\therefore \boldsymbol{n} = -\frac{\boldsymbol{R \cdot |S| \cdot b}}{\boldsymbol{\theta}}
\end{aligned}
$$

A.1.2 Weiss and Tan

$$
\begin{aligned}
\frac{d}{dn} TotalCost(n) &= 0 \\
\frac{d}{dn}(\theta \cdot n + \epsilon_n \cdot |S| \cdot R) &= 0 \\
\frac{d}{dn}(\theta \cdot n + (a + \frac{b\,n}{n+1}) \cdot |S| \cdot R) &= 0 \\
\frac{d\,\theta}{dn} + |S| \cdot R \cdot \frac{d}{dn}(\frac{b\,n}{n+1}) &= 0 \\
\theta + |S| \cdot R \cdot (\frac{(n+1) \cdot b - bn}{(n+1)^2}) &= 0 \\
\theta + \frac{|S| \cdot R \cdot b}{n+1} - \frac{|S|R \cdot b \cdot n}{(n+1^2)} &= 0 \\
(n+1)^2 \cdot \theta = -|S| \cdot R \cdot b \\
\therefore \boldsymbol{n} = \sqrt{-\frac{\boldsymbol{R \cdot |S| \cdot b}}{\boldsymbol{\theta}}} - \mathbf{1}
\end{aligned}
$$

A.1.3 Power law

$$
\begin{aligned}
\frac{d}{dn} TotalCost(n) &= 0 \\
\frac{d}{dn}(\theta \cdot n + \epsilon_n \cdot |S| \cdot R) &= 0 \\
\frac{d}{dn}(\theta \cdot n + (a \cdot n^b) \cdot |S| \cdot R) &= 0 \\
\frac{d\,\theta}{dn} + |S| \cdot R \cdot a \cdot \frac{d}{dn}(n^b) &= 0 \\
a \cdot R \cdot |S| \cdot b \cdot n^{(b-1)} = -\theta \\
\therefore \boldsymbol{n} = (\frac{\boldsymbol{-\theta}}{\boldsymbol{R \cdot |S| \cdot a \cdot b}})^{\frac{\mathbf{1}}{\boldsymbol{(b-1)}}}
\end{aligned}
$$

A.1.4 Exponential

$$\begin{aligned}
\frac{d}{dn} TotalCost(n) &= 0 \\
\frac{d}{dn}(\theta \cdot n + \epsilon_n \cdot |S| \cdot R) &= 0 \\
\frac{d}{dn}(\theta \cdot n + (a \cdot b^n) \cdot |S| \cdot R) &= 0 \\
\frac{d\theta}{dn} + |S| \cdot R \cdot a \cdot \frac{d}{dn}(b^n) &= 0 \\
\theta + |S| \cdot R \cdot a \cdot b^n \cdot \ln(b) &= 0 \\
b^n = -\frac{\theta}{a \cdot R \cdot |S| \cdot \ln(b)} & \\
n \log(b) = \log(\frac{-\theta}{R \cdot |S| \cdot a \cdot \ln(b)}) & \\
\therefore \boldsymbol{n = log_b(\frac{-\theta}{R \cdot |S| \cdot a \cdot \ln(b)})} &
\end{aligned}$$

A.2 Code and implementation

Source is available at : https://github.com/atrisarkar/ASE_extn

A.2.1 FAQ

Q. Where do I feed in the input files?
A. Input files are read from `com.ase.extn.constants.data.input`

Q: How can I execute progressive sampling?
A: Set the following parameters in `com.ase.extn.constants.configs.py`

```
strategy = 'progressive'
system = 'apache'|'bc'|'bj'|'llvm'|'x264'|'sqlite'|'all'
```
To display figures : `plot = True`

To display learning curve: `plot_real_cost = False`
To display cost curve: `plot_real_cost = True`
Execute `com.ase.extn.cart.base.py`
Results are generated under `com.ase.extn.constants.data.output`

Q: How can I execute projective sampling?
A: Set the following parameters in `com.ase.extn.constants.configs.py`

```
strategy = 'projective'
system = 'apache'|'bc'|'bj'|'llvm'|'x264'|'sqlite'|'all'
print_detail = True
```

To display figures : `plot = True`
Execute `com.ase.extn.cart.base.py`
Results are displayed in the console

Q. How to setup the parameters for projective sampling?
A.

Cost-ratio (R) : `r = <value>`
Feature-frequency threshold (*thresh_freq*) : `projective_feature_threshold = <value>`
Multiplier for training-testing set split (θ) : `th = <value>`

Q. How to run t-way sampling?
A. Set the following parameters in `com.ase.extn.constants.configs.py`

```
tway = 2 | 3
```

Execute `com.ase.extn.tway.twaysample.py`
Results are displayed in the console

Q. How to run sensitivity analysis?
A. Set the following parameters in `com.ase.extn.sensitivity.sanalysis.py`

```
sensitivity = 'r' | 'th'
```

`com.ase.extn.constants.configs.r_0_to_1 = True` : For SA in the interval [0,1] for R