GSDLAB TECHNICAL REPORT

Why CART Works for Variability-Aware Performance Prediction? An Empirical Study on Performance Distributions

Jianmei Guo, Krzysztof Czarnecki, Sven Apel,
Norbert Siegmund, Andrzej Wąsowski

GSDLAB–TR–2013–04–02 April 2013

Generative Software
Development Lab

University of
Waterloo

Generative Software Development Laboratory
University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1

WWW page: http://gsd.uwaterloo.ca/

Why CART Works for Variability-Aware Performance Prediction? An Empirical Study on Performance Distributions

Jianmei Guo*, Krzysztof Czarnecki*, Sven Apel†, Norbert Siegmund‡, and Andrzej Wąsowski§
*University of Waterloo, Canada
†University of Passau, Germany
‡University of Magdeburg, Germany
§IT University of Copenhagen, Denmark

***Abstract*—This report presents follow-up work for our previous technical report "Variability-Aware Performance Modeling: A Statistical Learning Approach" (GSDLAB-TR-2012-08-18). We try to give evidence why our approach, based on a statistical-learning technique called Classification And Regression Trees (CART), works for variability-aware performance prediction. To this end, we conduct a comparative analysis of performance distributions on the evaluated case studies and empirically explore why our approach works with small random samples.**

I. Introduction

In our previous technical report [2], we proposed an incremental and variability-aware approach to performance prediction for configurable software systems. We use a statistical learning technique called Classification And Regression Trees (CART) to build the correlation between feature selections and performance, to be used for performance prediction. Empirical results on six real-world case studies show that our approach achieves an average of $93.8\,\%$ prediction accuracy based on small random samples. However, we still want to give evidence why our approach works for variability-aware performance prediction, especially with small random samples.

CART and its variants (e.g., Random Forests and Boosting) have been widely used in statistics and data mining, because CART's algorithm is quick and reliable, and its tree structure can provide insight into the relevant input variables for prediction [1], [3]. Some studies have demonstrated the effects of CART on performance prediction for different case studies [5], [6], but they did not explicitly evidence why CART does or does not work.

A general explanation from the statistical-learning theory is that a regression model works well when the problem it addresses or the data it evaluates does fit the regressive pattern it builds [1]. CART builds a tree-like regression model M that recursively partitions a sample S and makes the total prediction errors in each partition minimal (refer to [2] for more details); this way, the regression model always fits the sample well. Furthermore, if the sample can represent the whole population WP or reflect the important characteristics of the whole population, then the prediction model built on the sample also fits the whole population well and makes accurate predictions. This can be formalized as follows:

$$M \vdash S \ \wedge \ S \sim WP \ \Rightarrow \ M \vdash WP \tag{1}$$

Since our prediction targets numeric performance values, the performance distribution is an important characteristic for performance prediction. Hence, we conduct a comparative analysis of performance distributions on six real-world case studies and empirically explore why our approach works with small random samples. Here, a performance distribution denotes the frequency distribution of all performance values in a sample or in the whole population. A key finding is that our approach based on CART works well when the sample it uses has a similar performance distribution as the whole population

In the remaining of this report, we first describe the subject systems used in our case studies. Then, we analyze and compare performance distributions for each system. All experimental data are available on the project's web site: `http://cpm.googlecode.com`.

II. Subject Systems

We performed the case studies on a publicly-available dataset, deployed with the SPLConqueror tool.[1] The dataset covers a reasonable spectrum of practical application scenarios. As shown in Table I, there are six existing real-world customizable systems with different characteristics: different sizes (45 thousand to 300 thousand lines of code, 192 to millions of configurations), different implementation languages (C, C++, and Java), and different configuration mechanisms (conditional compilation, configuration files, and command-line options). Moreover, the dataset contains the whole population of each system, i.e., all configurations of each system and their performance measurements (The exception is SQLite, for which the original developers measured $4,553$ configurations for prediction modeling and 100 additional random configurations for prediction evaluation [4]). For each system, the performance has been measured using a standard benchmark either delivered by its vendor (e.g., Oracle's standard benchmark for

[1]The dataset is available from here: http://fosd.de/SPLConqueror.

TABLE I
OVERVIEW OF THE SIX SUBJECT SYSTEMS. LANG.—LANGUAGE; LOC—LINES OF CODE; $|\mathbf{X}|$—NUMBER OF ALL CONFIGURATIONS; N—NUMBER OF ALL FEATURES; M—NUMBER OF CONFIGURATIONS REQUIRED BY THE PAIR-WISE HEURISTIC USED IN SPLCONQUEROR.

	System	Domain	Lang.	LOC	$\lvert\mathbf{X}\rvert$	N	M
1	Apache	Web Server	C	230,277	192	9	29
2	LLVM	Compiler	C++	47,549	1,024	11	62
3	x264	Encoder	C	45,743	1,152	16	81
4	Berkeley DB	Database	C	219,811	2,560	18	139
5	Berkeley DB	Database	Java	42,596	400	26	48
6	SQLite	Database	C	312,625	3,932,160	39	566

Berkeley DB) or used widely in its application domain (e.g., AUTOBENCH and HTTPERF for Apache Web Server).

To better understand each subject system, we present the feature model of each system and analyze it as follows.

Figure 1 shows the feature model of the Apache Web server (called Apache, for short). Apache has nine features other than the root feature, which is used to denote the system itself. It contains one mandatory feature, eight optional features, and one cross-tree constraints.

Figure 2 shows the feature model of the LLVM compiler infrastructure (called LLVM, for short). LLVM has 11 features other than the root feature. It contains one mandatory feature, 10 optional features, and no cross-tree constraints.

Figure 3 shows the feature model of the x264 video stream encoder (called x264, for short). x264 has 16 features other than the root feature. It contains three mandatory feature, seven optional features, two alternative groups, and no cross-tree constraints.

Figure 4 shows the feature model of the Berkeley database's C version (called Berkeley DB C, for short). Berkeley DB C has 18 features other than the root feature. It contains two mandatory features, seven optional features, two alternative groups, and no cross-tree constraints.

Figure 5 shows the feature model of the Berkeley database's Java version (called Berkeley DB Java, for short). Berkeley DB Java has 26 features other than the root feature. It contains 13 mandatory feature, five optional features, four alternative groups, and one cross-tree constraints.

Figure 6 shows the feature model of the SQLite database (called SQLite, for short). SQLite has 39 features other than the root feature. It contains nine mandatory feature, 15 optional features, five alternative groups, and no cross-tree constraints.

III. COMPARATIVE ANALYSIS OF PERFORMANCE DISTRIBUTIONS

In this section, we analyze and compare the performance distributions between a random sample S and the whole population WH on our evaluated case studies. The hypothesis is that our CART-based approach works well with a small random sample when the sample has a similar performance distribution as the whole population.

For each subject system in our case studies, we first visualize and analyze the performance distribution of the whole population. The exception is SQLite, for which the original developers cannot measure the whole population in reasonable time. So, we use all $4,553$ measured configurations to represent the whole population of SQLite, but we are aware of that it is a threat to validity.

Then, we collected all random samples generated in the previous experiment on the prediction fault rate [2]. We visualize the average performance distribution for each sample size of N, $2N$, $3N$, or M to mitigate the influence of a specific performance distribution of a certain random sample.

Furthermore, we compare the similarity between the performance distribution of a random sample (size N, $2N$, $3N$, or M) and that of the whole population. We analyze the similarity to the corresponding fault rate observed in the previous experiment [2].

We use a *histogram* to visualize a performance distribution. A histogram provides a quick and intuitive visualization of the distribution of the data [7]; it consists of two parts: the *vertical bars*, each of which displays the frequency of each value range; and the *density estimate curve*, which shows a more accurate display of the distribution of the data.

Figure 7 shows the Apache performance distributions. The performance distribution of the whole population is roughly a mixture distribution with two peaks. Compared to the whole population, we can see that the N sample has the least similarity, and it produces a prediction fault rate of $26.9\,\%$. The $2N$ sample already has a similar performance distribution as the whole population, and the prediction fault rate reduces dramatically to $11.6\,\%$. The $3N$ and M samples are more similar to the whole population than the $2N$ sample, and their prediction fault rates reach $8.4\,\%$ and $9.7\,\%$. In this case, the $3N$ sample has a lower fault rate than the M sample, and its performance distribution is also more similar to the whole population. It is because $3N = 27$ is very close to $M = 29$ in this case as well as the fluctuations caused by the random generation of samples.

In most cases, a larger random sample is more similar to the whole population, if such a sample is not skewed to some undesirable characteristics (e.g., always missing some features).[2] To reduce the fluctuations of the prediction fault rate caused by random generation, we performed five repetitions for each system and each sample size. That is, for each subject system, we repeated five times generating a random sample of a certain size and subsequently measured the prediction fault rate after applying our approach to the sample. We took only the average of these measurements for analysis. However, the fluctuations may still exist in our experimental results, which is a threat to validity.

Figure 8 shows the LLVM performance distributions. The performance distribution of the whole population is roughly a single distribution. The N sample captures the correct peak and the rough trend of the performance distribution, and it produces a quite low fault rate of $5.7\,\%$. The $2N$, $3N$, and M samples identify two trivial subpeaks gradually, and they

[2]An exploratory experiment on missing features and skewed configurations can be found in our technical report: http://gsd.uwaterloo.ca/node/484.

achieve a robust decreasing trend of the fault rate from $4.5\,\%$, $4.0\,\%$, to $3.3\,\%$.

Figure 9 shows the x264 performance distributions. The performance distribution of the whole population is roughly a mixture distribution with two peaks. The N sample identifies the two peaks, but misses the correct locations of the two peaks; and it produces a fault rate of $15.1\,\%$. Next, the $2N$, $3N$, and M samples gradually move and form the two peaks approximately to the precise locations, as shown in the performance distribution of the whole population. With such a gradual process that makes the more similar performance distribution as the whole population, the prediction fault rate shows a robust decreasing trend from $15.1\,\%$ to $8.5\,\%$, $7.2\,\%$, and $6.4\,\%$ when the sample size increases from N to $2N$, 3N to M.

Figure 10 shows the Berkeley DB C performance distributions. The performance distribution of the whole population is roughly a mixture distribution with four peaks. The N sample roughly identifies three peaks, and it produces a quite high fault rate of $112.4\,\%$. The $2N$ and $3N$ samples further move the three peaks gradually to the precise locations, but they both miss the fourth peak, and their fault rates still remain at $98.3\,\%$ and $46.8\,\%$. Only when the M sample identifies the fourth peak, which is next to the leftmost peak, the fault rate reduces sharply to $7.8\,\%$.

Furthermore, from this case, we can see that the intuitive comparison of the shape and trend of performance distributions is not sufficient. For Berkeley DB C, the whole population has $2,242$ distinct performance values and these values spread over a large value range. That demands many measurement points to guarantee reasonable prediction accuracy. Hence, it is necessary to consider more influencing factors for the similarity between a random sample and the whole population and further quantify the similarity, which will be explored in future work.

Figure 11 shows the Berkeley DB Java performance distributions. The performance distribution of the whole population is roughly a mixture distribution with four peaks. Despite this, the whole population of Berkeley DB Java has only 171 distinct performance values. The N sample identifies all four peaks, and it already produces a fault rate of $3.2\,\%$. Subsequently, the fault rates of the $2N$, $3N$, and M samples fluctuate smoothly from $2.2\,\%$, $2.6\,\%$, to $2.7\,\%$.

Figure 12 shows the SQLite performance distributions. We use all $4,553$ measured configurations to represent the whole population of SQLite. The performance distribution of the whole population is roughly a single distribution. The N sample already has a similar performance distribution as the whole population, and its fault rate reaches $8.0\,\%$. Then, the $2N$, $3N$, and M samples gradually refine the performance distribution and produce the fault rate from $8.1\,\%$, $7.6\,\%$, to $7.2\,\%$.

IV. Conclusion

The comparative analysis of performance distributions between random samples and the whole populations reveals that our approach works well with a small random sample when the sample has a similar performance distribution as the whole population. In fact, we found an explicit evidence that a sample does reflect some important characteristics of the whole population when we can produce accurate predictions based on it. However, we are aware of that the performance distribution is just one of the important characteristics for performance prediction. These characteristics may involve, for example, the number and dispersion of distinct values as well as the feature coverage. A quantitative study on the similarity between a random sample and the whole population, involving more characteristics for performance prediction, will be explored in future work.

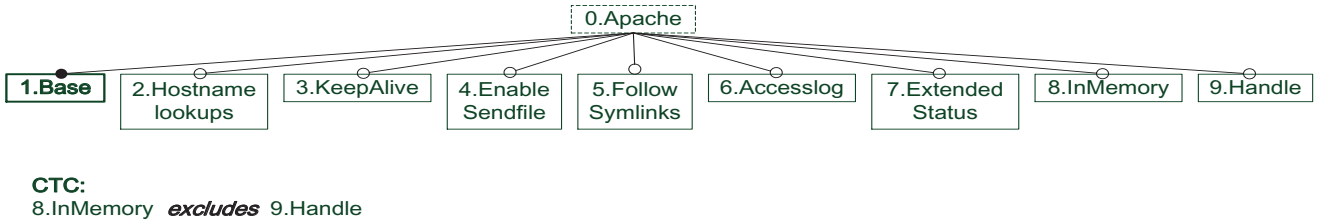


Figure 1. The feature model of Apache

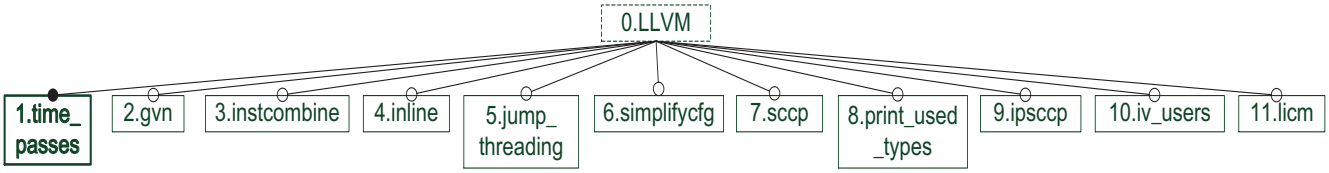


Figure 2. The feature model of LLVM

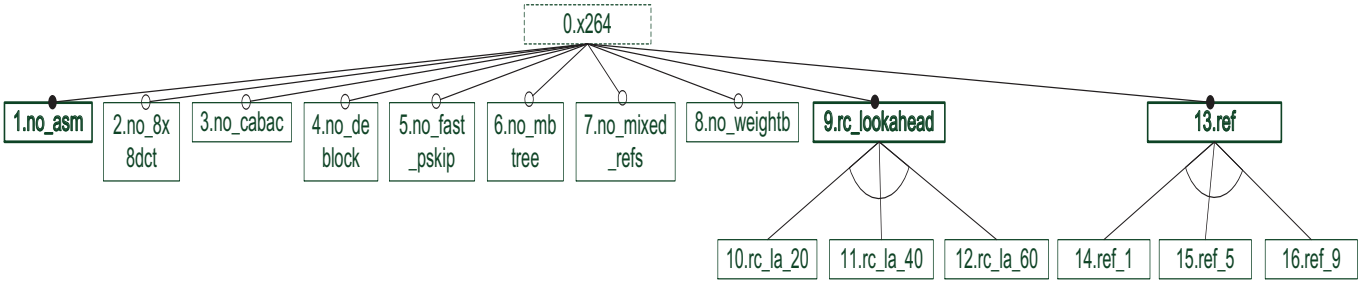


Figure 3. The feature model of x264

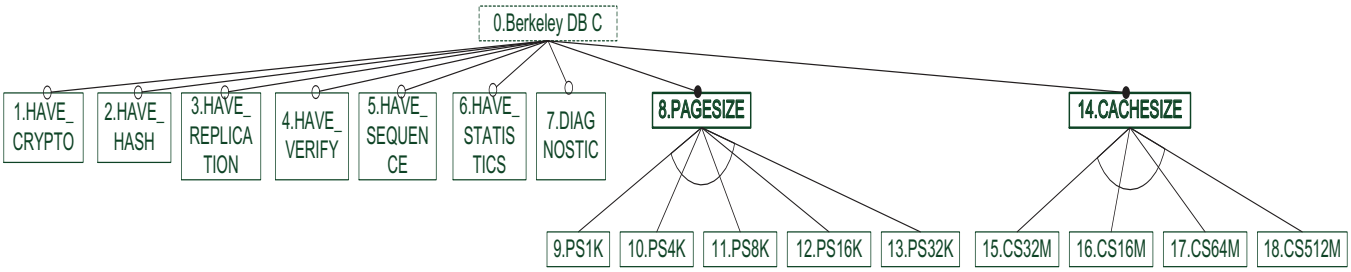


Figure 4. The feature model of Berkeley DB C

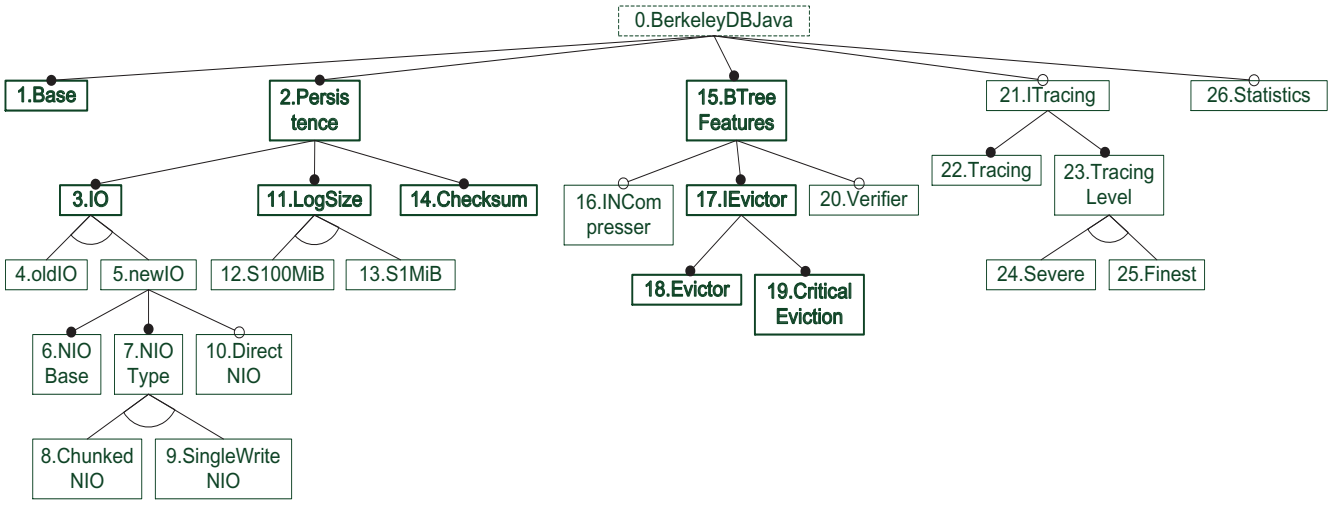


Figure 5. The feature model of Berkeley DB Java

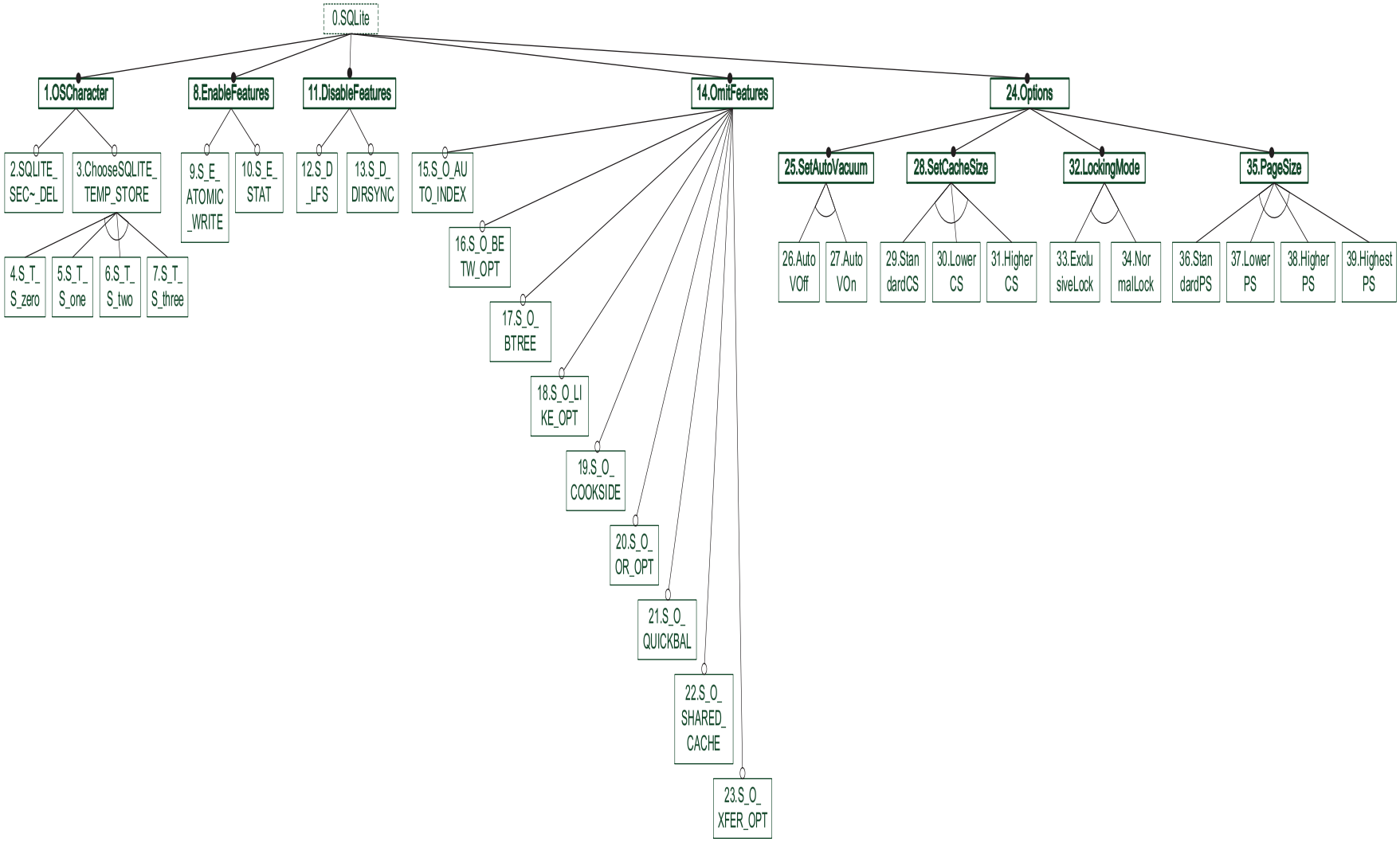
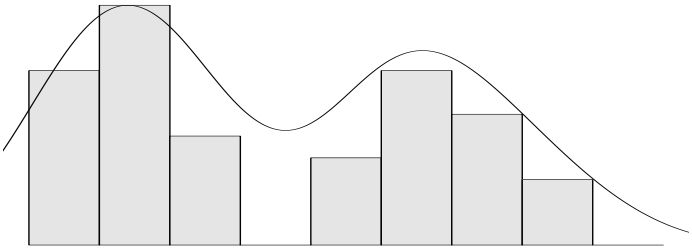
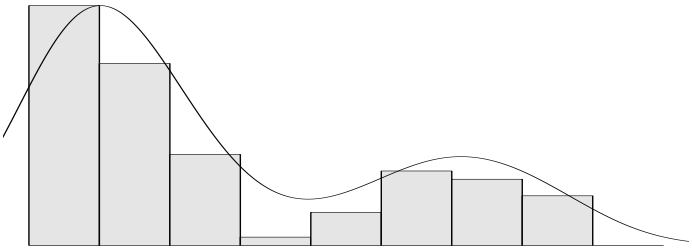


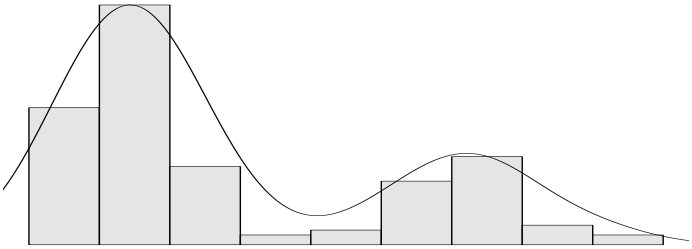
Figure 6. The feature model of SQLite



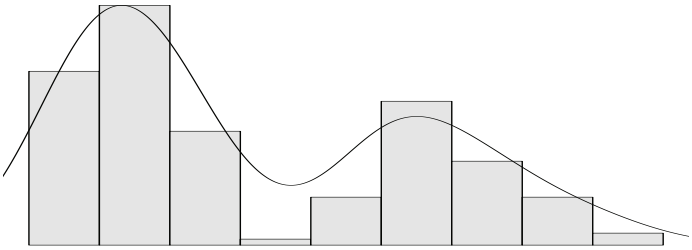
(a) $|S| = N$



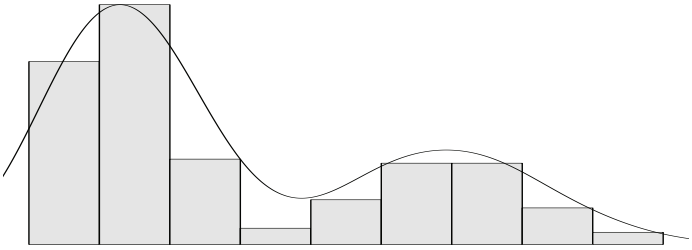
(b) $|S| = 2N$



(c) $|S| = 3N$

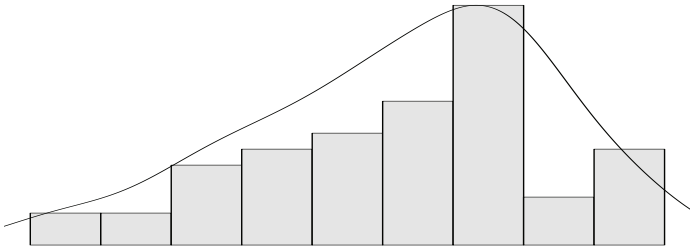


(d) $|S| = M$

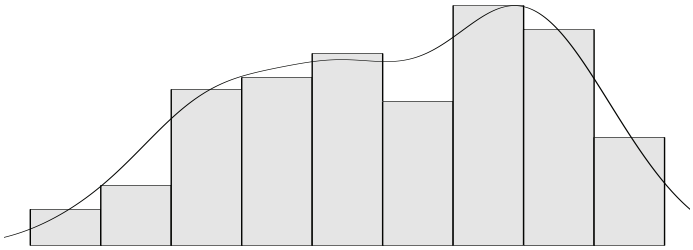


(e) WH

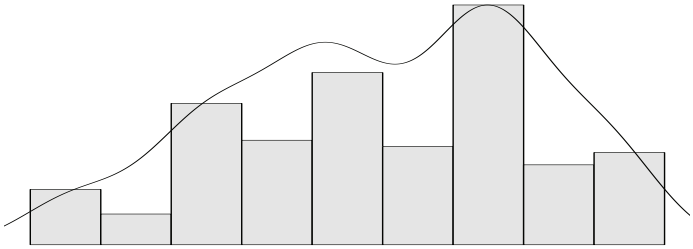
Figure 7. Histograms of the performance distributions of the random samples of four sizes ($N \sim M$) and of the whole population of Apache. (X-axis: Performance (seconds); Y-axis: Relative Frequency.)



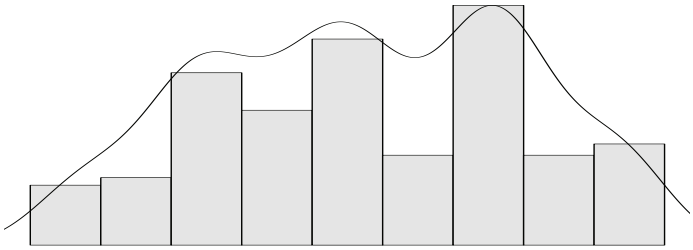
(a) $|S| = N$



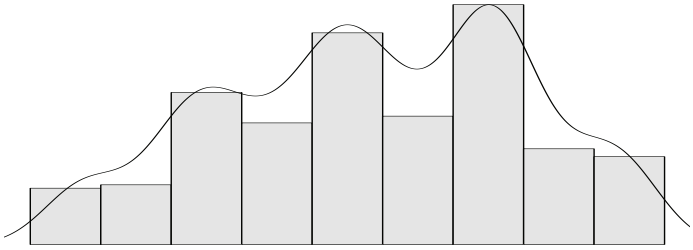
(b) $|S| = 2N$



(c) $|S| = 3N$

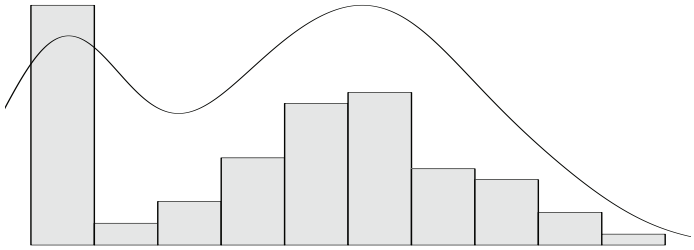


(d) $|S| = M$

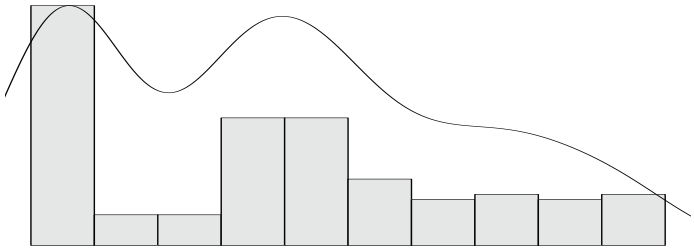


(e) WH

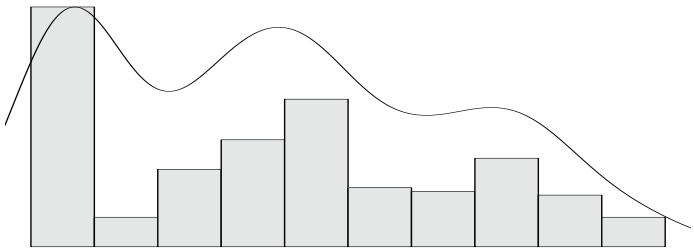
Figure 8. Histograms of the performance distributions of the random samples of four sizes ($N \sim M$) and of the whole population of LLVM. (X-axis: Performance (seconds); Y-axis: Relative Frequency.)



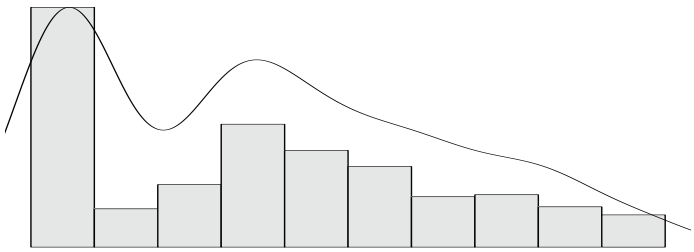
(a) $|S| = N$



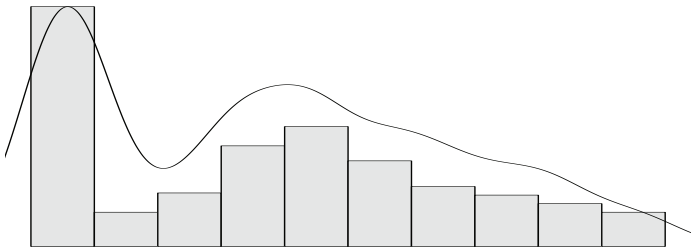
(b) $|S| = 2N$



(c) $|S| = 3N$

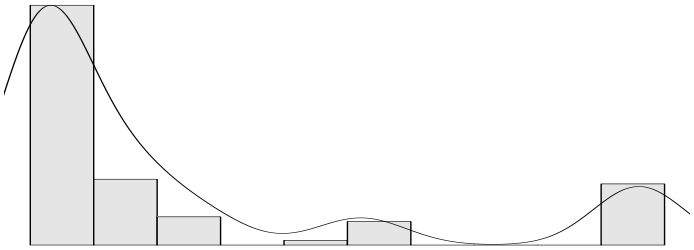


(d) $|S| = M$

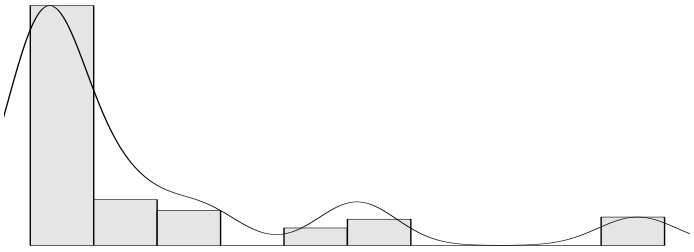


(e) WH

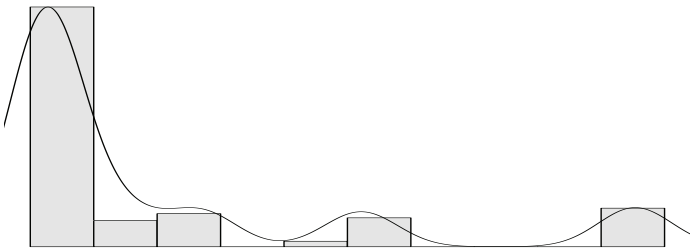
Figure 9. Histograms of the performance distributions of the random samples of four sizes ($N \sim M$) and of the whole population of x264. (X-axis: Performance (seconds); Y-axis: Relative Frequency.)



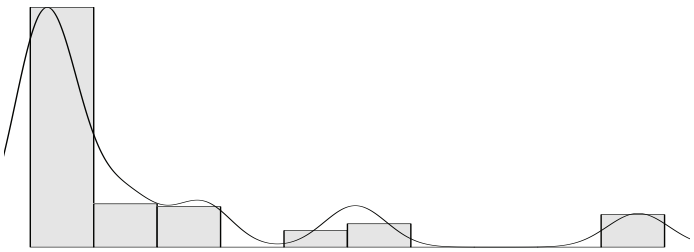
(a) $|S| = N$



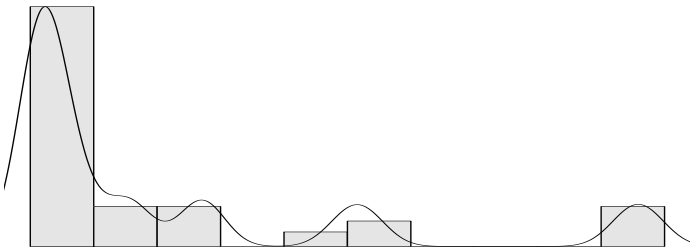
(b) $|S| = 2N$



(c) $|S| = 3N$



(d) $|S| = M$

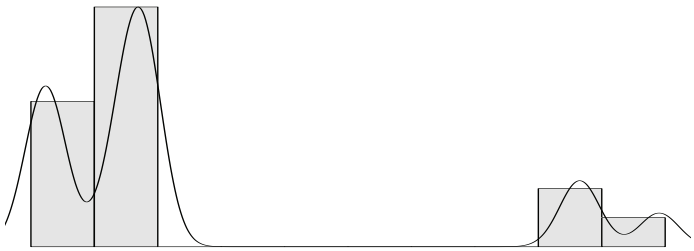


(e) WH

Figure 10. Histograms of the performance distributions of the random samples of four sizes ($N \sim M$) and of the whole population of Berkeley DB C. (X-axis: Performance (seconds); Y-axis: Relative Frequency.)

(a) $|S| = N$

(b) $|S| = 2N$

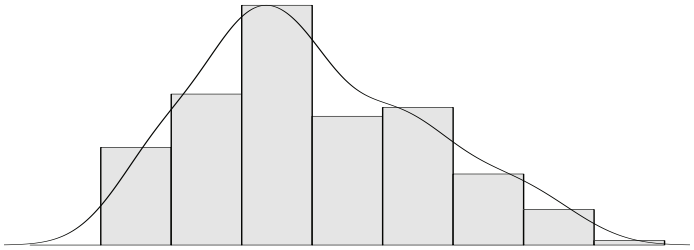


(c) $|S| = 3N$

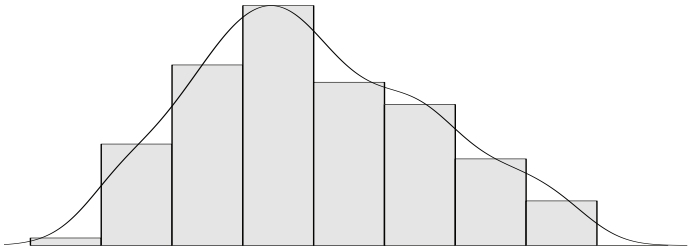
(d) $|S| = M$

(e) WH

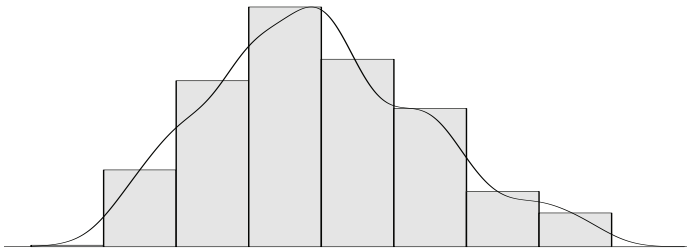
Figure 11. Histograms of the performance distributions of the random samples of four sizes ($N \sim M$) and of the whole population of Berkeley DB Java. (X-axis: Performance (seconds); Y-axis: Relative Frequency.)



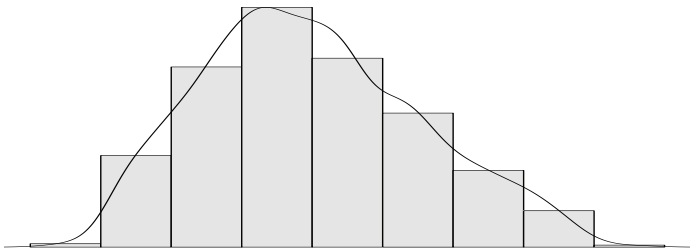
(a) $|S| = N$



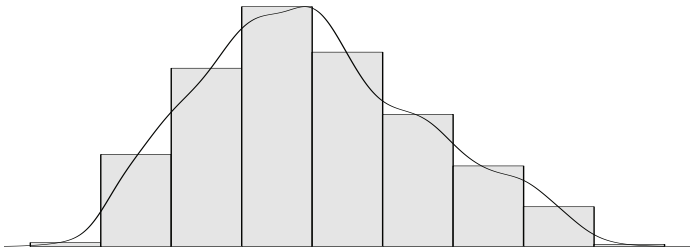
(b) $|S| = 2N$



(c) $|S| = 3N$



(d) $|S| = M$



(e) WH

Figure 12. Histograms of the performance distributions of the random samples of four sizes ($N \sim M$) and of the whole population of SQLite. (X-axis: Performance (seconds); Y-axis: Relative Frequency.)

REFERENCES

[1] R. A. Berk, *Statistical Learning from a Regression Perspective*. New York, NY, USA: Springer, 2008.

[2] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wąsowski, "Variability-aware performance modeling: A statistical learning approach," Generative Software Development Laboratory, University of Waterloo, Tech. Rep. GSDLAB-TR-2012-08-18, 2012.

[3] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.

[4] N. Siegmund, S. S. Kolesnikov, C. Kastner, S. Apel, D. S. Batory, M. Rosenmuller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *ICSE*. Zürich, Switzerland: IEEE, 2012, pp. 167–177.

[5] E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel, "Practical performance models for complex, popular applications," in *SIGMETRICS*. New York, NY, USA: ACM, 2010, pp. 1–12.

[6] D. Westermann, J. Happe, R. Krebs, and R. Farahbod, "Automated Inference of Goal-Oriented Performance Prediction Functions," in *ASE*. Essen, Germany: ACM, 2012, pp. 190–199.

[7] G. Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*. New York, NY, USA: Springer, 2011.