Preprint submitted to Innovations in Systems and Software Engineering

Maximal Software Execution Time: A Regression-based Approach

Ayoub Nouri · Peter Poplavko · Lefteris Angelis · Alexandros Zerzelidis · Saddek Bensalem · Panagiotis Katsaros

Abstract This work aims at facilitating the schedulability analysis of non-critical systems, in particular those that have soft real-time constraints, where Worst-Case Execution Times (WCETs) can be replaced by less stringent probabilistic bounds, which we call Maximal Execution Times (METs). To this end, it is possible to obtain adequate probabilistic execution time models by separating the non-random dependency on input data from a modeling error that is purely random. The proposed approach first utilizes execution time multivariate measurements for building a multiple regression model and then uses the theory related to confidence bounds of coefficients, in order to estimate the upper bound of execution time. Although certainly our method cannot directly achieve extreme probability levels that are usually expected for WCETs, it is an attractive alternative for MET analysis, since it can arguably guarantee safe probabilistic bounds. The method's effectiveness is demonstrated on a JPEG decoder running on an industrial SPARC V8 processor.

Keywords WCET · Linear Regression · Stepwise Regression · Principle Components Analysis · JPEG

The research leading to these results has received funding from the European Space Agency project MoSaTT-CMP, Contract No. 4000111814/14/NL/MH

A. Nouri · S. Bensalem
Univ. Grenoble Alpes, CNRS, Grenoble INP*, VERIMAG, 38000 Grenoble, France
700 avenue centrale, Saint Martin d'hères 38401, France
E-mail: ayoub.nouri@univ-grenoble-alpes.fr

P. Poplavko
Mentor® A Siemens Business
F-38334 Inovallee Montbonnot, France (The presented research was done while working at UGA/VERIMAG)

L. Angelis · A. Zerzelidis · P. Katsaros
Information Technologies Institute, Centre of Research & Technology - Hellas 6th km Xarilaou - Thermi, 57001, Thessaloniki, Greece

L. Angelis · P. Katsaros
Department of Informatics, Aristotle University of Thessaloniki, Greece

*Institute of Engineering Univ. Grenoble Alpes

1 Introduction

We propose a new statistical approach, for the timing analysis of embedded software programs. Such methods aim at highly-probable execution time overestimations, as opposed to the 100% certain upper bounds given by common worst-case execution time (WCET) estimation techniques.

This option can be justified in many practical situations. Actually, only a small subset of safety critical applications (*e.g.,* avionics) require conservatively that for the system to operate correctly, all its tasks need to run up to their WCET. In such cases, an over-provisioned design – entailing costly hardware resources – is assured. On the other hand, it often happens that infrequent and non successive deadline misses can be tolerated, and may be even preferable to excessive hardware costs. This reasoning is applicable for systems that do not have safety requirements (*e.g.,* car infotainment) that are characterized by weak, soft or firm real-time constraints. For all these systems, we can rely on statistical (over-)estimations based on extensive measurements that we call probabilistic *maximal* execution times (MET).

The proposed method is measurement-based, *i.e.,* it relies on executing the program code and collecting

measurements, as opposed to static approaches[1]. Such methods are referred to as *measurement-based timing analysis* (MBTA). When relying on probabilistic or statistical analysis techniques, the latter are called *probabilistic* MBTA.

In the recent research literature, the reliability of probabilistic MBTA techniques has been improved, even to the level of considering them eligible —under some restrictive hardware assumptions (*e.g.,* cache randomization) —as WCET estimates for hard real-time systems. Such estimates provide an upper bound on the program execution time with a certain probability (α) that the actual program execution time can exceed it. Probabilistic WCETs are METs that hold at an extremely high probability ($1 - \alpha$) with $\alpha = 10^{-15}$ per program execution [4] or 10^{-9} per hour, which corresponds to the most stringent requirements in safety-critical standards.

Analyses aiming to 'true' WCET (with $\alpha = 0$) are costly to adapt to new application domains and processor architectures, as they require the construction of complex exact models that have to be verified. The techniques based on extreme value theory (EVT) can ensure the levels of probability that render them suitable for WCET. However, these techniques assume that the execution times are random and identically distributed, an overly strong assumption that does not generally hold in practice. Execution times typically show significant autocorrelations and their probability distribution varies, due to the input data dependencies.

For non safety-critical systems, one can settle for METs characterized by an α a few orders of magnitude larger than that claimed by EVT methods ($\alpha = 10^{-15}$). In this case, it is possible to rely on a rich set of mature statistical model fitting tools, such as *linear regression*, which can handle the input data dependencies. In the current article, we propose a novel probabilistic MET analysis technique that builds upon linear regression and the associated statistical analyses.

Linear regression was proposed in [10], as a means for conservative execution time analysis, but without having profited from the rich statistics associated with it. More specifically, that work aimed at 100% conservative estimates (without probabilities) and for this reason it focused on non-statistical linear model fitting techniques. However, targeting 100% conservative estimates may result in a costly analysis, losing the advantage of regression. Moreover, their technique for calculating the regression parameters is rather *ad hoc* and it is not described in detail. On the other hand, an important connection between linear regression and WCET analysis methodologies is established, which is based on implicit path enumeration.

In the same article, some interesting possibilities are also shown for the explicit modeling of hardware effects, such as pipelining, which could be used as well in our work. However, for simplicity, in this article we do not address the hardware modeling issue directly, but undoubtedly this is an important future work matter. Since our analysis is based on measurements on real hardware and since the variability attributed to hardware is a consequence of the variability of input data, we believe that hardware effects are covered indirectly up to a level of accuracy that may be appropriate for non safety-critical applications.

The present paper is an extended version of the work published in [11]. The main contributions are the following. In Section 2, we discuss the MBTA method and we recall some of the basics of Linear Regression. Section 3 groups all the theoretical contributions of the paper. It first introduces a new regression model, called the Maximal Regression Model (Section 3.1) that yields probabilistic upper bounds for METs estimation using confidence intervals. A great challenge in this context is to find a good hypothesis model, that is to identify the most relevant variables to accurately explain and predict the execution time. For this purpose, we propose, Stepwise Regression, an iterative procedure for building a compact regression model (Section 3.2.1). In addition, we propose in this extended version, a new method for variables identification based on Principal Component Analysis (PCA) in Section 3.2.2. We provide, in Section 3.3, a pragmatic method to compute the MET estimate. We note that, following this method, the obtained MET estimate is independent from input data. Finally, since we follow a measurement-based approach, a statistical technique is also proposed in Section 3.4 for the assessment of the quality of the input data in order to obtain pertinent measurements.

In Section 4, we detail the instrumentation and measurement techniques necessary to collect the required input data for the above analyses techniques. With this respect, the scalability of the approach in [11] is further improved with a new preprocessing phase based on I-point Graphs (IPG). In Section 5, all these techniques are assembled together in an automated analysis flow for METs estimation. In Section 6, the overall approach is demonstrated on a JPEG decoder case study with a significant input data dependency, which runs on a state-of the art industrial SPARC V8 architecture with caches reset at every execution start. Additional experiments and analysis results using Stepwise Regression and PCA are reported compared to [11]. At the end of

[1] Classifying the different methods is beyond of the scope of this paper. A detailed survey of the different approaches and their classifications can found in [12]

Section 6 we aslo provide a comparative discussion of Stepwise Regression and PCA. The related work is further discussed along with the conclusions, in Section 7.

2 Common Probabilistic Techniques

We first review the probabilistic MBTA setting, and then we recall the basics of linear regression, while interpreting it in the context of probabilistic MBTA.

2.1 Measurement-based Probabilistic Timing Analysis

MBTA consists of initially performing multiple execution time measurements of the program's blocks of code and/or the program's execution time, and subsequently analyzing them to combine the results and thereafter to calculate the MET bound (see Fig. 1). The probabilistic variant of MBTA utilizes statistical methods for the analysis phase [4, 2].

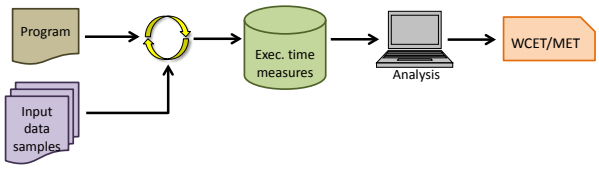


Fig. 1: Overview of MBTA steps

Let us denote Y the execution time, which in general depends on a set of variables X_i. A MET bound with probability $(1-\alpha)$ can be obtained by finding the minimal y such that $\mathcal{P}r\{Y < y\} \geq (1-\alpha)$.

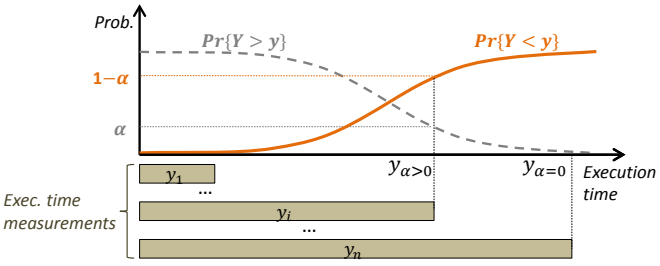


Fig. 2: Illustration of the probability of MET bound

Such a bound is related with $\mathcal{P}r\{Y > y\}$, as it is shown in Fig. 2 (dashed line). For small values of y the probability is high and it decreases when y increases. The curve converges to 0 for very high values of y, which corresponds to the WCET case with $\alpha = 0$ ($y_{\alpha=0}$ is very unlikely to be observed in the set of measurements). The $\mathcal{P}r\{Y < y\}$ can be characterized by the symmetrical curve (solid line) and corresponds to $1-\alpha$. We are interested to find the smallest y – not necessarily from measurements – that matches a sufficiently small α (respectively high $1-\alpha$).

Suppose that Y is random with a known continuous distribution f, denoted as $Y \sim f$. A possible solution is given through the *quantile function* of that distribution, *i.e.,* for $y = Q_f(1-\alpha)$ we have, by definition, $\mathcal{P}r\{Y < y\} = (1-\alpha)$. If Y is normally distributed, *i.e.,* $Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$, we have $y = \mu_Y + \sigma_Y \Phi^{-1}(1-\alpha)$, where Φ^{-1} is the quantile function for $\mathcal{N}(0,1)$. In order to calculate METs by using this formula, the 'mean' μ_Y and the 'standard deviation' σ_Y have to be estimated from measurements with sufficient precision, for which a large enough number of measured Y samples may be required. The normal distribution can describe many random physical variables, like for example noise and measurement error, and provides access to a rich set of mature statistical tools for reliably deriving estimates from measurements.

Unfortunately, neither normal nor any other distribution law can be justified to *directly* describe execution times. Therefore, the probabilistic MBTA techniques do not consider the execution time itself as a random variable, but only some of its characteristics. For example, the normal distribution can be adequate if we assume that an 'oracle model' exists, which for each program run can predict its execution time Y almost perfectly, but still makes a small error due to various independent factors ignored by the 'oracle'. In this case, it is reasonable to apply the normal distribution law to characterize the error of the 'oracle'. This is, in fact, the underlying idea of our method, though the normal distribution is only adequate for the values of α that are not too small. As a consequence, our approach can be applied only to soft real-time systems.

For estimates with very small α, MBTA analyses use EVT [4]. They apply EVT probability distribution laws, again, not to the execution times directly, but to their upper bounds. However, an important requirement for justifying the EVT-based techniques is that the execution times will have to be independent and identically distributed (*iid*) random variables. This requirement is typically violated, due to the dependency on input data via multiple conditional branches and loops in the program. The input data parameters are not *iid* and in a certain sense they are even 'non-random' (no practically adequate distribution law can characterize them). Therefore, for programs with complex control flow the applicability of EVT-based techniques is difficult to justify. By contrast, using linear

regression as an 'oracle model', our method separates the non-random factors from the modeling error.

2.2 Linear Regression in the Nutshell

Linear regression is mostly used to predict *average* estimates [6, 8]. Though our goal is to produce *upper bounds*, the same approach is used as the starting point.

The ultimate aim of linear regression is to model a variable of interest Y, called *dependent* variable, with *explanatory* variables (or *predictors*) X_i. In the context of MBTA, the dependent variable Y is the program execution time, and $Y(n)$ is its n^{th} observation in a series of measurements. The concrete values of X_i represent the possibility to 'explain' (or 'predict'), with some precision, the concrete value of Y.

The fundamental assumptions for the validity of a linear regression are: (i) X_i have approximately linear contribution to Y and (ii) the approximation error is normally distributed. The first assumption is realistic, since it is always possible to decompose execution time as a linear combination of contributions by the executed blocks of code (see Example 1 for an illustration). The second assumption is motivated in the previous subsection and it is further confirmed by experiments (Section 6). To produce an MET estimate, if we can obtain bounds for the predictors X_i this will allow us to derive a bound on Y as well.

In linear regression [5], the dependence of Y on X_i is given by

$$Y(n) = \beta_0 + \beta_1 X_1(n) + \ldots + \beta_{p-1} X_{p-1}(n) + \epsilon(n) \quad (1)$$

where coefficients β_i are *parameters* that have to be *fitted* to the measurements $Y(n)$ for minimizing the *regression error* $\epsilon(n)$ as shown in Fig 4. The dependent variable Y, the error ϵ, and the parameters β_i are all modeled as real numbers, since they represent (components of) the execution time. Their probability distributions are assumed to be continuous, as it is usually the case for timing metrics in statistical MET methods [4, 1]. On the other hand, the predictors X_i are non-negative integers that count the number of times that some important branch or loop iteration in the program is taken (or skipped). The corresponding parameter β_i can be either positive, to reflect the processor time spent per unit of X_i, or negative, to reflect the economized time.

Example 1 (Linear regression on a small program code) Let us consider the following artificial program that computes the product of two input vectors of integers (`v1` and `v2`). Then, based on the computed value and on a third input parameter `u`, it computes the final result `out`. The execution time Y of the program is given as: (1) the time required to compute the product of the input vectors (denoted β_0), plus (2) the time required to execute the instruction in the 'if' body (denoted β_1) – whether the 'if' branch is executed or not, this is captured by a predictor X_1 – plus (3) the time required to execute the 'while' loop. We denote β_2 the execution time for one iteration, with the total time given as a product with a predictor X_2 that captures the number of performed iterations.

```
Input: v1, v2 (integer vectors), u (integer)
Output: out

out = v1 * v2;
if(out < 0) {
 out = -out;
}
while(out > u) {
 out = out * 0.9;
}
return out;
```

Fig. 3: A sample program code

It is worth to note that X_1 depends on the product of the input vectors `v1` and `v2`, *i.e.,* it captures a dependency on the program's inputs. Similarly, the predictor of the number of iterations of the 'while' loop X_2 depends also on the same product, and on the value of `u`. Hence, these dependencies are represented by

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

where ϵ is the regression error, which aggregates the effects of factors that are not measured such as pipelining, cache hit ratio, etc. ■

From a probabilistic MBTA perspective, equation (1) captures the 'non-randomness' of Y by building a model $\sum_i \beta_i X_i(n)$, which 'explains' its dependence on the factors X_i with weights β_i that reflect the complexity of the program. Ideally, the remaining 'non-explained' part is a random variable $\beta_0 + \epsilon(n)$ with β_0 representing the mean value and $\epsilon(n)$ the random deviation, whereby $\epsilon(n)$ are hopefully independent and normally distributed by $\mathcal{N}(0, \sigma_\epsilon)$.

The accuracy of the bounds proposed here depends on the validity of the aforementioned assumption, though the bounds are generally robust with respect to deviations from the normal distribution. For justifying the 'randomness' of ϵ, we consider that all non-random factors are captured by X_i. Finally, the normality of ϵ is justified using the central limit theorem based on the intuitive observation that the sources of execution time

variation, *e.g.,* non-linearity of X_i, are additive in nature and independent.

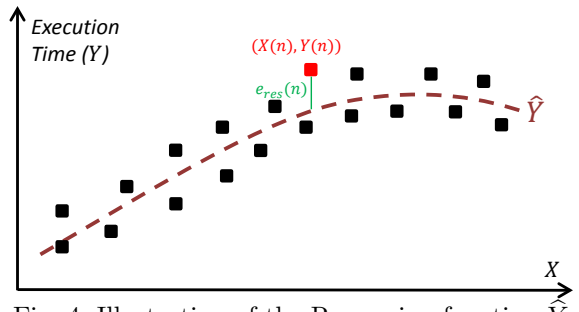


Fig. 4: Illustration of the Regression function $\widehat{Y}$

The exact values of parameters β_i in equation 1 are unknown, and can only be estimated based on measurements, *e.g.,* with the *least-squares method*. Let us denote b_i the estimate of β_i and $\widehat{Y}$ the estimate of Y. When ϵ is 0, we get an *unbiased regression model*

$$\widehat{Y}(n) = b_0 + b_1 X_1(n) + \ldots + b_{p-1} X_{p-1}(n) \tag{2}$$

and the difference $e_{\text{res}}(n) = Y(n) - \widehat{Y}(n)$, called *residual*, serves as an estimator of the error $\epsilon(n)$: $\epsilon(n) \approx e_{\text{res}}(n)$ (see Fig. 4).

For more convenience, we use a vector notation. Let $\mathbf{x} = [X_0 \ldots X_{p-1}]$ be the vector of predictors, where $X_0 = 1$ is an artificial constant predictor that corresponds to b_0, and $\mathbf{b} = [b_0 \ldots b_{p-1}]$ the vector of parameter estimators. We denote $\mathbf{x}(n) = [X_0(n) \ldots X_{p-1}(n)]$ the n^{th} observation of predictors X_i. The regression model in (2) can thus be rewritten as the vector product

$$\widehat{Y}(n) = \mathbf{b} \cdot \mathbf{x}(n)$$

The model parameters $\mathbf{b}$ are obtained from a set of measurements – the so-called training set – through a process known as *model training* (or *fitting*). In our case, the training set consists of N measurements of execution time and predictors. In practice, N is recommended to be $N \gg p$, *i.e.,* at least $N > 5p$ [10]. We consider a training-set with predictor measurements organized into a $N \times p$ matrix $\mathbf{X}^{\text{train}}$ and the corresponding N-dimensional vector of execution time measurements $\mathbf{y}^{\text{train}}$.

$$\mathbf{X}^{\text{train}} = \begin{pmatrix} \mathbf{x}(1) \\ \mathbf{x}(2) \\ \vdots \\ \mathbf{x}(N) \end{pmatrix}, \quad \mathbf{y}^{\text{train}} = \begin{pmatrix} Y(1) \\ Y(2) \\ \vdots \\ Y(N) \end{pmatrix}$$

3 Linear Regression for MET

The Maximal Regression Model is introduced, as a means to compute conservative MET estimations. For identifying the most relevant predictors, we propose two techniques, and we then discuss the technique used to compute the MET estimate. To ensure accuracy, it is essential to evaluate the representativeness of the input data, since we rely on measurements. An appropriate technique to this end is also proposed here.

3.1 The Maximal Regression Model

The least-squares method provides a closed form formula for computing the vector $\mathbf{b}$ from $\mathbf{X}^{\text{train}}$ and $\mathbf{y}^{\text{train}}$ (see [5] for details). However, each least-square parameter b_i is itself a random variable, since it is obtained from a training-set $\mathbf{y}^{\text{train}}$ 'perturbed' with a random error ϵ. It turns out, from theoretical studies, that each estimate b_i can be seen itself as a sample from a normal distribution, since different training sets would lead to distinct samples b_i from the distribution shown in Fig. 5. This distribution has as mean value the unknown parameter β_i and therefore, the samples b_i are likely to be close to β_i.

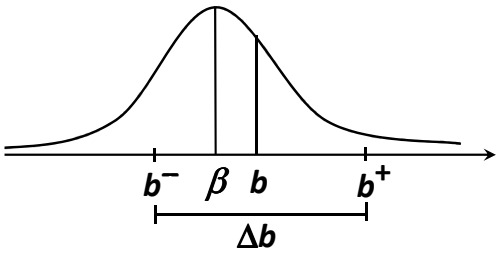


Fig. 5: Parameter Confidence Interval

However, the model parameters b that simply are '*close*' to β are not appropriate for the estimation of METs. We opt for a conservative model consisting of parameters b^+ that are likely to *overestimate* β. Such parameters can be obtained using the notion of confidence interval, which is an interval $\Delta b = [b^-, b^+]$ that likely contains β (cf. Fig. 5), such that

$$\mathcal{P}r\{\beta \in \Delta b\} = (1 - \alpha) \tag{3}$$

where α is a sufficiently small value, usually specified in percents, *e.g.,* $\alpha = 5$ %. By symmetry with the distribution of b, if we use b^+ as coefficient estimator, then our aforementioned model is conservative with probability $(1 - \alpha/2)$. We therefore conclude to a *maximal*

regression model

$$\widehat{Y}^{+}(n) = b_0^{+} + b_1^{+} X_1(n) + \ldots + b_{p-1}^{+} X_{p-1}(n) + \epsilon^{+} \quad (4)$$

where ϵ^{+} is the maximal error, as opposed to the usual unbiased regression model given by equation (2). The b_i^{+} are trivially obtained from formulas used to compute b and its confidence interval, which are provided in popular mathematical packages. Heuristically, we opt for computing b_i^{+} with the *same* probability bound $(1-\alpha/2)$ for all i, in order to avoid 'favoring' any predictor over the other predictors.

For the calculation of ϵ^{+}, *i.e.,* the probabilistic upper bound of the residual, and in order to avoid 'favoring' it over the other bounds, we use the same probability such that $\mathcal{P}r\{\epsilon(n) < \epsilon^{+}\} \geq (1-\alpha/2)$. Since $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$, we would ideally calculate ϵ^{+} using the quantile $Q_{\mathcal{N}(0,\sigma_\epsilon)}(1-\alpha/2) = \sigma_\epsilon \cdot \Phi^{-1}(1-\alpha/2)$. However, the exact value of σ_ϵ is not available and therefore we use the formula: $\epsilon^{+} = \widehat{\sigma}_\epsilon^{+} \cdot \Phi^{-1}(1-\alpha/2)$ where $\widehat{\sigma}_\epsilon^{+}$ is the calculated probabilistic upper bound on σ_ϵ based on measurements. This bound is obtained using a commonly known property (*e.g.,* [5]) of multivariate linear regression, according to which $\hat{\sigma}_\epsilon^2 = S^2/K$ where $S = \sum_{i=1}^{N} (\widehat{Y}(i) - \widehat{Y}^{+}(i))^2$ is the sum of squares of the residual over the N samples of the training set, and K is a real random variable distributed by χ^2 distribution with $(N-p)$ degrees of freedom. This leads to the following formula:

$$\epsilon^{+} = \left(\sqrt{\frac{\sum_{n=1}^{N} (Y(n) - \widehat{Y}(n))^2}{Q_{\chi^2(N-p)}(\alpha/2)}} \right) \cdot \Phi^{-1}(1-\alpha/2) \quad (5)$$

where $Q_{\chi^2(N-p)}$ is the quantile function of the distribution of K.

By comparison of equations (1) and (4) all the terms of the first are likely to be inferior to the corresponding terms of the second, and therefore $\widehat{Y}^{+}(n)$ is a probabilistic bound of $Y(n)$ (an illustration is provided in Fig. 6).

To estimate the reliability of the maximal regression model, let us calculate the chances that, contrary to expectations, $Y(n) > \widehat{Y}^{+}(n)$. To this end, we conservatively assume that this will happen when *at least* one of the upper bounds used in (4) is violated. There are p bounds for the regression coefficients b_i^{+} and two bounds used for ϵ^{+} in (5). Recall that we have chosen all bounds to admit the same violation probability $\alpha/2$. Therefore, our model ensures maximal execution time with the following guarantee:

$$\mathcal{P}r\{Y(n) < \widehat{Y}^{+}(n)\} \geq \left(1 - \frac{(p+2)\alpha}{2}\right) \quad (6)$$

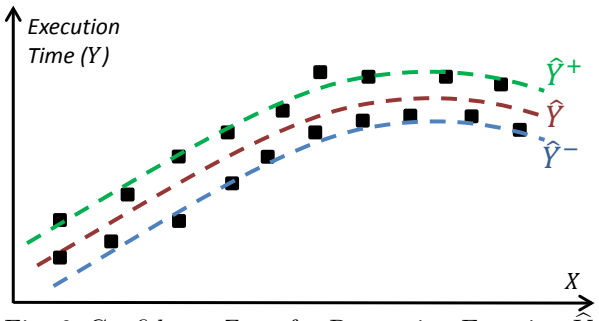


Fig. 6: Confidence Zone for Regression Function $\widehat{Y}$

3.2 Identifying the Predictors

The predictors of a regression model for software execution times count the number of executions, for each executed block of code [10]. To this end, every programming construct that introduces branching, *e.g.,* loop and 'if' statements, contributes with at least one potential predictor. This results in a relatively large set of *potential predictors* that is denoted as P_{pp}. We are interested to *identify* only a sufficiently small set $P \subseteq P_{pp}$ for the regression model; for this purpose, the simple rule of thumb $N > 5 \cdot p$ with $p = |P|$ is used, as in similar studies.

The identification process is essential to exclude redundant variables, due to interdependent predictors as is the case for nested loops, where the (total) number of inner-loop iterations is potentially strongly dependent on the number of outer-loop iterations. From each pair of dependent variables it suffices to keep only one, while attributing the small additional effect of the other variable to random error ϵ. Such a process aims to avoid a potential over-fitting, if overly many variables are kept in the set of predictors, in which case the model fits perfectly to the training set, but it cannot reliably predict any other program execution. This happens when the model fits exactly not only the 'true' linear dependence $\beta_i X_i$, but also the particular sample of random noise ϵ encountered in the training set, but not in other samples.

In applied mathematical studies, the identification of useful predictors in a set of candidates is an important problem to solve ([see 5, Chap. 15] and [7, Chap. 3]). In other studies for execution time modeling this process is either manual or *ad hoc*. Here, we propose a practical and mathematically sound algorithm for identifying the subset P of P_{pp}.

3.2.1 Method 1: Stepwise Regression

The main criterion of any strategy for identifying the right set of predictors is the reduction of the model

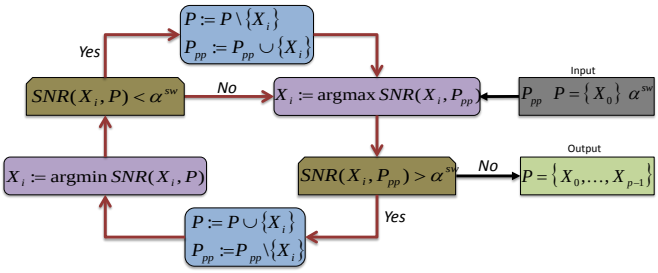


Fig. 7: Illustration of the Stepwise Regression procedure

error, when adding a new predictor to a previously selected set of predictors. It is thus expected that for a certain number of selected predictors, new variables do not reduce the error significantly anymore. The identification process is then stopped by adopting the hypothesis that the remaining error represents a 'random noise'. One of the most well-established methods that we propose for using in the MET analysis is the *stepwise regression* ([see 5] for details).

A tentative set P of predictors is maintained along the procedure. Initially, P contains the constant predictor $X_0 = 1$ (*i.e.*, $p = |P| = 1$), which remains in the set throughout all subsequent steps. The algorithm proceeds by adding a variable that is worth to be included in P, while removing from P a variable that is not worth to keep; the same step is repeated until no progress can be made. When a variable is added, it is temporarily moved from P_{pp} to P, and when a variable is removed it is moved backwards. The algorithm terminates, whenever no other variables are worth to be added or removed. At the end, we still consider that $P \subseteq P_{pp}$, since all the found predictors come from the set P_{pp}.

Whether a variable is considered 'worth' to be included in P or not depends on the variables that are already in P; the decision is taken by evaluating the least-squares regression $\widehat{Y}$ with and without the candidate predictor. A variable is 'worth' if its 'signal to noise ratio' is significantly large, where the 'noise' is the total model error (evaluated by the residual sum of squares) and the 'signal' is the contribution of the variable to the variance of $\widehat{Y}$. Fig. 7 illustrates the described procedure. If the change in variance by keeping the variable is negligible compared to the total error, then the variable is not 'worth'. The whole procedure is controlled by a parameter α^{sw} that sets a threshold for variable acceptance/rejection, and is based on statistical hypothesis-testing procedures under the assumption that the modeling error is normally distributed.

3.2.2 Method 2: Principal Component Analysis

Principal component analysis (PCA) is a well known technique for dimensiality reduction [9]: given a vector $\mathbf{x} = [X_0 \dots X_{|P_{pp}|-1}] \in \mathbb{R}^{|P_{pp}|}$, the method produces a new vector $\mathbf{z} = [Z_0 \dots Z_{p-1}] \in \mathbb{R}^p$ such that $p < |P_{pp}|$. The new variables - components can be extracted so as to be uncorrelated with each other. This addresses the problems of multicollinearity that may exist among the original variables. We use PCA to identify regression predictors. Let us consider $\mathbf{x} = [X_0 X_1] \in \mathbb{R}^2$, a two dimension vector, and assume that X_1 partially captures some information that is already captured by X_0. It is then possible to 'compress' the information in $\mathbf{x}$ in a new vector $\mathbf{z} = [Z_0] \in \mathbb{R}$ that contains only the useful information. A graphical interpretation of this reduction is shown in Fig. 8.

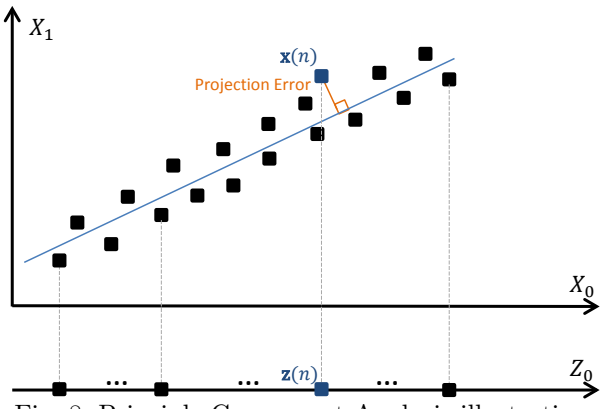


Fig. 8: Principle Component Analysis illustration

Unlike the stepwise regression method, PCA does not identify predictors by *selection* from set P_{pp}. It is assumed *a priori* that all predictors in P_{pp} are equally 'interesting', but since they are dependent on each other all the information conveyed in P_{pp} can be 'compressed' into a smaller set P' of variables. The so-called *principal components* Z_j are linear combinations of predictors from P_{pp} (here $p = |P'|$):

$$Z_{j=1\dots p-1}(n) = \gamma_1^{(j)} \breve{X}_1(n) + \dots \gamma_{|P_{pp}|-1}^{(j)} \breve{X}_{|P_{pp}|-1}(n) \quad (7)$$

where $\breve{X}(n) = (X(n) - \mu_X)/\sigma_X$ is a 'normalized' and 'scaled' version of X, with μ_X being the average and σ_X the standard deviation of X in the training set (X_i are normalized to make their ranges compatible). Furthermore, in contrast to stepwise regression this method ignores measurements of Y and focuses only on those of X_i (there are though some method variants [see 9, Chap. 8], which take into account Y to some extent). Thus, PCA has the advantage that predictor identification can be performed without execution time measure-

ments on any platform, and can be reused on different platforms.

As in the previous method, the 'worthiness' of a potential predictor is also measured by a variance metric, but this time its own variance instead of its impact on the variance of $\widehat{Y}$. The larger variance implies that more 'information' is conveyed. First, the most 'worthy' predictor Z_1 is identified, which 'absorbs' as much as possible of the variance from variables $\breve{X}_i$. The second predictor Z_2 then absorbs the remaining variance to the largest possible extent, and the process continues until the required percentage of variance, $1-\alpha^{pc}$ (*e.g.,* 99%), has been absorbed by $p-1$ variables, while the rest remains 'almost constant' without potential to bring much information (α^{pc} is a user-provided parameter). The vectors $\gamma^{(1)}, \gamma^{(2)}, \ldots$ are calculated as normalized eigenvectors of the matrix of Pearson correlation coefficients for the variables X [9], sorted by decreasing eigenvalues. Intuitively, they represent vectors (characterizing a lower dimension surface) onto which to project data, so as to minimize the projection error (see Fig. 8)[2].

The regression of Y is thus done with predictors Z_j (for numerical reasons, we prefer to normalize not only X_i, but also Y). An advantage of predictors Z_j is that they are not correlated with each other in the training set; in fact, they are orthogonal, which is favorable for doing the linear regression on them. We note that these variables may also take negative values.

3.3 Pragmatic MET.

Our maximal regression model could be used within the context of the *implicit path enumeration technique* (IPET) by following the approach in [10]. In this case, the MET would be computed by

$$\epsilon^+ + \max_{\mathbf{X}\in\mathbb{X}} \left(\sum_{i=0}^{p-1} b_i^+ X_i \right) \tag{8}$$

with $\mathbb{X}$ the set of all vectors $\mathbf{X}$ that result from the feasible program paths. This is achieved by solving the integer linear programming (ILP) problem with a set of constraints on the variables X_i. The constraints are derived from a *static program analysis*, which requires sophisticated tools, as well as from user provided hints, such as loop bounds.

The IPET method has not yet been implemented in our analysis framework. For each predictor, we assume that its minimal and maximal bounds X^- and X^+ are available, either from measurements or as user hint, and we then calculate the pessimistic estimate:

$$\epsilon^+ + b_0^+ + \sum_{i=1}^{p-1} (bX)_i^+ \tag{9}$$

where $(bX)^+$ is b^+X^+ if $b^+ > 0$ or b^+X^- otherwise. We refer to this estimate as the *pragmatic MET*.

In the case of PCA, since Z_j can be negative, the pragmatic MET is calculated as follows:

$$\epsilon^+ + \sum_j \max(b_j^+ Z_j^+, b_j^- Z_j^+, b_j^+ Z_j^-, b_j^- Z_j^-) \tag{10}$$

The pragmatic MET, in general, can be too pessimistic; for example, in a switch-case branching it may associate with every case a separate predictor, and can then assume that they all take the maximal value simultaneously. Nevertheless, the pragmatic MET is safe with the probability bound in (6), if the regression model itself is safe with this bound.

3.4 Quality of Input Data: Cook's Distance

The set of measurements must include all important scenarios that may occur at runtime. To ensure this, the engineer has to discover the most influential algorithmic complexity parameters of the program that may vary at run time. Then, an input data set has to be found, where every *combination* of these factors is represented *fairly*.

For the linear regression, a useful mathematical metric of input-data quality is the *Cook's distance*. Given a set of measurements, this metric ranks every measurement n by a numeric 'distance' value $D(n)$ that indicates the extent to which the measurement influences the whole regression model. The model should not be dominated by 'odd' measurements; it is generally recommended to have $D(n) < 1$ or even $D(n) < 4/N$. For convenience, let us refer to the measurements with $D(n) > \theta$ as the *bad samples*, for some threshold θ. These samples should be examined, and one should either add more similar samples (so that they are not exceptional anymore) or remove them from the training set (keep them for testing).

4 Instrumentation and Measurements

We explain now the instrumentation procedure to get the measurements, for the set of potential predictors P_{pp} upon which the previous analysis can be performed.

In some MBTA approaches, it may be necessary to instrument multiple blocks of program code to measure their individual contributions to execution time [1].

[2] This should not be confused with the error $e_{\text{res}}(n)$ in the case of Linear Regression (compare with Fig. 4 to see the difference).

This instrumentation approach can be intrusive, whereas it is likely to obtain inaccurate results when adding the block contributions, due to various hardware effects (e.g. pipelining). However, in a regression-based approach with end-to-end measurements taken over the entire program, the instrumentation is not intrusive. For the measurements $Y(n)$ the program has to be instrumented only at its start and end. As for the measurements needed for the set of potential predictors P_{pp} and in order to obtain their values $X_i(n)$, the instrumented program can run on any workstation, instead of the target platform, but it has to run with the same input data, as those used for the $Y(n)$ measurements. We refer to these measurements as *functional simulations*.

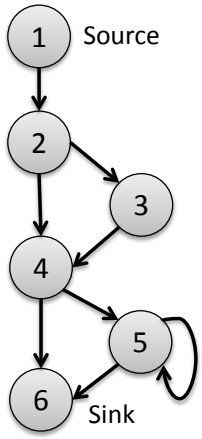
Instrumentation for end-to-end execution time measurement is straightforward and can be easily performed in different ways. In the remainder of this section, we focus on the instrumentation and measurement for the set of predictors P_{pp}, *i.e.,* the functional simulation.

```
ipoint(1);
out = v1 * v2;
ipoint(2);

if(out < 0) {
 out = -out;
 ipoint(3);
}
ipoint(4);

while(out > i) {
 out = out * 0.9;
 ipoint(5);
}
ipoint(6)
```

(a) Instrumented Source Code



(b) I-point graph

$X_1 = f(2,3)$ – 'if' body counter
$X_2 = f(5,5)$ – 'while' loop counter

(c) Associated Predictors

Fig. 9: Instrumentation and i-point graph

4.1 Instrumentation Points and Traces

The instrumentation for functional simulations consists of inserting *instrumentation points (i-points)* into the source code of the program [3] [4]. The i-points are inserted at every point, where the control flow diverges or converges, *e.g.,* at the start/end of the conditional and loop blocks, at the branches of the conditional statements etc. An i-point is a subroutine call with a parameter q that specifies the i-point's unique identifier. For instance, Fig. 9a shows the instrumented version of the program presented in Fig. 3 in Example 1, where we have points with $q = 1, 2, \ldots, 6$. As shown in the code, these i-points are inserted, so as to identify the different paths followed during any simulation. For instance, the points $q = 2$ and $q = 3$ enable the detection of whether the 'if' branch was taken. The sequence of i-points visited during the n^{th} simulation run is called *i-point trace* and is denoted as $\mathcal{T}r(n) = (q_1, q_2, q_3, \ldots)$. Examples of traces for the instrumented program in Fig. 9a are $(1,2,4,6)$, $(1,2,3,4,5,6)$, and $(1,2,4,5,5,5,6)$, but, in general, the number of possible traces can be exponential in the size of the program.

[3] For a higher precision, it may be possible to instrument the binary code for the target platform in a separate binary executable used only for the construction of P_{pp}, and still use the non-instrumented version for the end-to-end execution time measurements on the target platform.

[4] Industrial tools, such as the one in [**?**], can be used to automate this intrumentation process. We actually used this tool for the JPEG experiments presented in Section 6.

4.2 Potential Predictors

As the program executes, the subroutines of the i-points insert their q identifiers into the execution trace. When the program exits, the trace is saved in a trace file. For the linear regression, a training set of size N consists of N i-point traces.

The i-point traces are used to: (i) construct the so-called *instrumentation point graph* (IPG) that characterizes the program's control flow [3] (Fig. 9b) and (ii) define the set of potential predictors P_{pp} (the n^{th} trace is used to obtain the n^{th} value of all predictors $\mathbf{x}(n) = [X_0(n) \ldots X_{|P_{pp}|-1}(n)]$).

The set of all i-points that occur in the training set yields the vertices Q of the IPG graph and the set of all simple blocks yields the set P_{pp} of its directed edges (the edges are denoted as P_{pp}, because they correspond one-to-one to the potential predictors). Thus, the IPG graph is a digraph (Q, P_{pp}) with exactly one *source* and one *sink* (vertex that has no incoming, respectively no outgoing edges). Fig. 9b shows the IPG graph for the program example (with 'if' and 'while' statements) in Fig. 9a.

The value of a potential predictor is computed from a trace as follows. A pair of points (q, r) that may occur successively in a trace defines a *simple block*, *i.e.,* the part of the program that is executed in between. In Fig. 9a, the simple blocks are $(1,2)$, $(2,3)$, $(2,4)$, $(4,5)$, $(5,5)$, and $(4,6)$. For instance, $(1,2)$ corresponds to the **v1** by **v2** multiplication operation. We define the *flow counter* $f(q,r)$ as the number of times that a simple block (q,r) occurs in the trace. The n^{th} value, $X_i(n)$

of a potential predictor X_i that corresponds to an IPG edge, $X_i \rightarrow (q, r)$, is given by the value of the flow counter $f(q, r)$ for the measured trace $\mathcal{T}r(n)$. In the current example, the predictor X_1 is associated to the body of the 'if' statement, which corresponds to the edge $(2, 3)$ in the IPG (*i.e.,* $f(2, 3)$). Predictor X_2 is associated to the body of the 'while' loop, *i.e.,* the edge $(5, 5)$ in the IPG (*i.e.,* $f(5, 5)$) as shown in Fig. 9c.

In general, we could consider complex code blocks, a subtrace $(q_1, q_2, q_3, \ldots)$, and also count their occurrence in the trace, which would give us knowledge about sequences of simple blocks that can be useful to handle hardware effects. For example, knowing pairs of subsequent simple blocks can help to take into account their temporal overlaps in pipelining [10]. However, for simplicity, in this work we only use simple blocks.

It is worth mentioning that the parameter β_0 (of the artificial predictor X_0) is in general associated to the part of the program executed unconditionally[5]. In the previous example, β_0 is associated to the cost of code block $(1, 2)$. Hence, as already stated in Example 1 the following regression model is obtained for this program $Y(n) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$. In the given set P_{pp}, only the specified predictors are linearly independent. The others are removed as explained in Section 5.1.

5 A Work Flow for MET

We build upon the techniques presented in the previous sections to introduce an automated work flow for MET estimation[6]. The proposed work flow consists of three major phases: (1) input program instrumentation and measurements, in order to obtain the set of potential predictors P_{pp}, (2) identification of the most relevant predictors P (or P' for PCA), and (3) model instantiation and actual estimation of the MET.

As sketched in Fig. 10, after having completed the code instrumentation and measurements (*i.e.,* i-point traces and end-to-end execution times), an IPG is built (cf. Section 4) that represents the set of all potential predictors P_{pp} observed in the i-point traces. The IPG can be used for performance optimization by removing useless predictors from P_{pp}, *e.g.,* constants (cf. Section 5.1).

The set of potential predictors P_{pp} is then further purified during the predictors' identification phase towards obtaining the final set of predictors $P \subseteq P_{pp}$ (or P' for PCA), using one of the methods in Section 3.2.

[5] In general, this execution time is never 0 due to different reasons *e.g.,* initialization; β_0 captures this cost.

[6] Sources are made available at `www-verimag.imag.fr/~nouri/met-estimation`

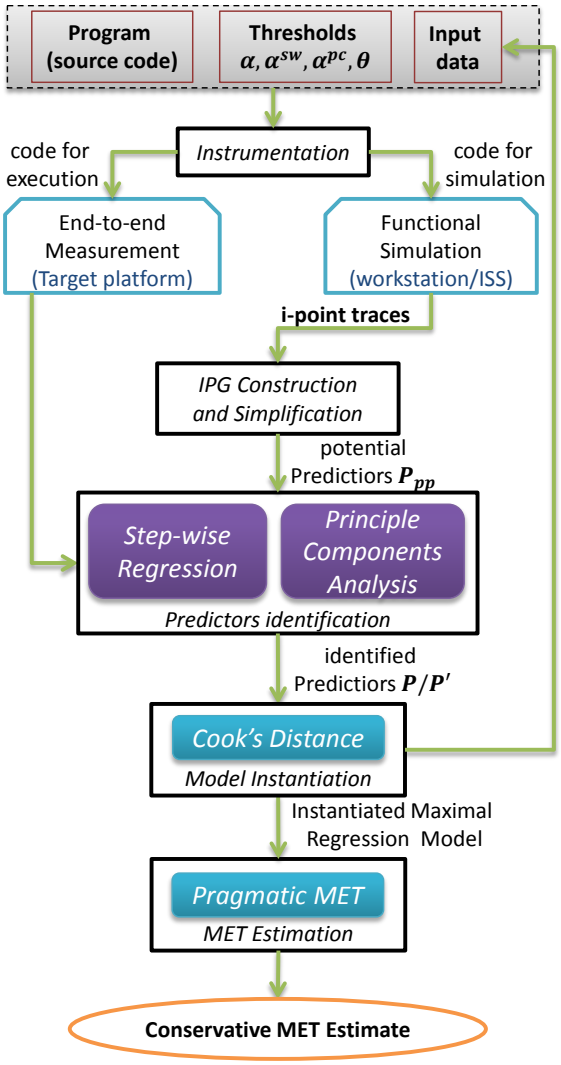


Fig. 10: A simplified view of the MET work flow

Later, during the model instantiation, it is primordial to perform a quality control on the considered input data using the Cook's distance of Section 3.4), in order to make sure that the obtained parameter estimations are good *e.g.,* they are not influenced by outliers. Otherwise, additional observations with respect to new inputs data need to be considered or outlier observations should be removed (at least form the training set). Once, the quality of the input data and thus the considered observations (measurements) is established, the maximal regression model is built as described in Section 3.1. Finally, a MET bound is calculated using the pragmatic MET technique of Section 3.3.

It is worth mentioning that measurements are separated into two sets, a training set $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$ and a test set $(\mathbf{X}^{\text{test}}, \mathbf{y}^{\text{test}})$[7], and that only the former is used

[7] A common practice is to consider 70% for the training set and 30% for the test set.

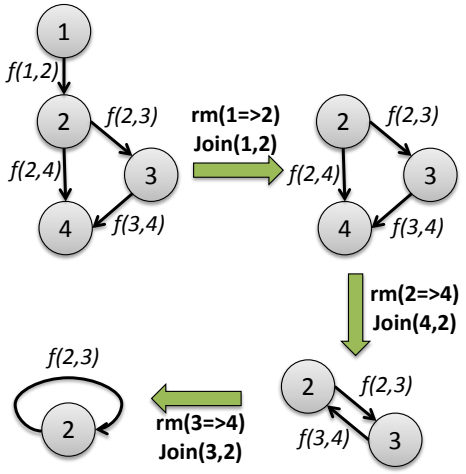


Fig. 11: IPG simplification example (cont. Fig 9)

to construct the model, whereas the latter is used to evaluate its quality.

5.1 IPG Simplification

Before performing the predictors' identification, the IPG 'simplification' procedure reduces the set P_{pp} of potential predictors by removing linearly dependent predictors. This could be very useful when the number of obtained predictors is very large. The IPG simplification relies on the following observation. For a given vertex r, which is neither a source nor a sink, we can write the following '*structural constraint*':

$$\sum_{q \in I} f(q,r) = \sum_{s \in O} f(r,s) \tag{11}$$

where I and O are respectively the set of predecessors and successors of r in the IPG. The structural constraints can be used as part of the IPET model to calculate the IPET MET by solving the ILP [10]. In our work, we use them to perform graph simplifications.

This process can be seen as a graph transformation, assuming that the IPG graph is a multi-graph, *i.e.,* it may contain multiple edges between the same pair of nodes. The transformation can be roughly represented by the repetitive selection of an edge between two different nodes r_1 and r_2; for each one of them, we can write (11). First, we substitute the expression for $f(r_1, r_2)$ from one equation to another. Next, we join the two nodes into one (because of the preserved equality), and finally we remove or redirect the underlying edge. Fig. 11 illustrates this procedure on a fragment covering the sub-graph starting in vertex 1 and ending in vertex 4 of the IPG in Fig. 9b.

The IPG simplification procedure also removes from P_{pp} all predictors measured as constants, and it adds an artificial predictor $X_0 = 1$ that is needed for regression.

6 A JPEG Decoder on a SPARC Platform

We use a JPEG decoder program written in C[8] to illustrate our method. The JPEG decoder processes the header and the main body of a JPEG file. Basically, the main body consists of a sequence of compressed MCUs (Minimum Coded Units) of 16×16 or 8×8 pixels. An MCU contains pixel blocks also referred to as 'color components', as they encode different color ingredients. In the color format '4:1:1' an MCU contains six blocks. For monochromatic images, the MCU contains only one pixel block. The pixel blocks are represented by a matrix of Discrete Cosine Transform (DCT) coefficients, which are encoded efficiently over few bits, so that a whole pixel block can fit in only a few bytes.

The execution time measurements took place on an FPGA board featuring a SPARC V8 processor with a 7-stage pipeline, a double-precision FPU, a 4 KB instruction cache, a 4 KB data cache, a 256 KB Level-2 cache, and an SDRAM. The data caches were reset at every new program run (*i.e.,* after loading a JPEG image), so that they are always empty at the beginning.

6.1 Instrumentation and Measurements

We used 99 different JPEG images of different sizes and color formats, which yields 99 execution traces and their execution times Y[9]. After the IPG simplification procedure, 103 potential predictors were detected from the generated traces. We then randomly split the set of 99 measurements into the training set consisting of $N = 70$ cases, and the test set with 29 cases. In the training set, 8 predictors showed up as constants and they were therefore eliminated, thus ending up with $|P_{pp}| = 95$ potential predictors, plus one constant $X_0 = 1$ added by default. Since we had a training set size $N = 70$, by the rule of thumb, we should not exceed $N/5 = 14$ variables, to avoid over-fitting.

It is worth mentioning that the maximal observed execution time over the whole set of measurements corresponds to an image of a particularly large size, yielding *maximal measured time* of 23643 Mcycles, while the *mean time* was only 1000 MCycles. In the remaining

[8] Downloaded from Internet, presumably authored by P. Guerrier and G. Janssen 1998

[9] We could not obtain more measurements because the FPGA card was available for a limited period of time, and loading data into it required some manual work.

discussion, all timing values (*e.g.,* errors) are reported in Megacycle units. We used $\alpha = 0.05$ for the maximal regression parameters and MET, but we also present the obtained estimate for $\alpha = 0.00005$.

6.2 Predictors Identification and Model Construction

6.2.1 Basic Model

The simplest model to build is when $p = 1$, *i.e.,* when the execution time is modeled as a purely random variable without non-random contributors, of the form $\beta_0 + \epsilon(n)$. This corresponds to a naïve measurement-based method, where the execution time does not account for the non-random factors. With such a strong input data dependency as in JPEG decoders, we obviously do not expect adequate results when using the aforementioned model. Indeed, we carried out a normality test for Y using the Kolmogorov-Smirnov procedure that reported only a mere 2% likelihood (normality was rejected); this was not surprising as the histogram for Y was considerably skewed and had a few extreme values, due to images of exceptionally large size. The obtained error was large compared to the mean $\epsilon^+ = 6650$ and the *pragmatic MET* was found to be ≈ 8000, which underestimates the maximal measured time. This adversity is the consequence of the relatively large model error whose distribution was not normal and which was actually not random (it could be easily controlled, *e.g.,* by using additional large JPEG images).

In line with our methodology, these observations point to a need of adding more predictors into the model (*e.g.,* those characterizing the image size), in order to ensure a smaller, random and normally distributed error, so that the computation of MET is more accurate.

6.2.2 Extended Model using Stepwise Regression

The α^{sw} was first tuned ($\approx 20\%$), in order to obtain $p = 6$, *i.e.,* to have 5 predictors. Table 1 shows the identified variables – in the order of their identification – and the corresponding MET calculation on the training set. The first predictor $f(271, 244)$ corresponds to the *byte count* in the 'main body' of JPEG. The second counter $f(90, 30)$ gives the *pixel block count* specifically for those blocks that had correct prediction of the 0-th DCT coefficient. Typically, such blocks are not costly in terms of needed bytes for encoding. At the same time, the contribution of the costly blocks can be captured by the first predictor. Hence, using the $f(90, 30)$ as second predictor can account for the additional computations that were not accounted for by the first predictor; a similar variable in P_{pp}, the *total* pixel block count, $f(406, 26)$, would give less additional information and hence was not identified by our method.

The remaining predictors have less impact on the execution time. The third predictor, $f(101, 101)$, corresponds to the number of *elements in the color format* minus one, *e.g.,* $5 = 6 - 1$ for the 4:1:1 format and 0 for monochromatic images. Equivalently, it gives the number of pixel blocks per MCU block minus one. We note that this predictor has a negative regression coefficient. The JPEG decoding is characterized by two related cost components: a cost per pixel block (reflected by the first two predictors) and a highly correlated cost per MCU block. The more pixel blocks fit into one MCU, the less overhead per pixel block has the MCU processing and this presumably explains why the found coefficient is negative. The fourth identified predictor, $f(80, 81)$, counts the number of *'padded' image dimensions*, X and Y, *i.e.,* the dimensions which are not exactly proportional to the MCU size (16 or 8 pixels). When an image has such dimensions, less processing is required and less data copying for 'partial' MCU blocks, which presumably explains the negative coefficient for this predictor. Finally, the predictor $f(409, 410)$ is zero for colored images. This predictor counts the total number of MCUs in monochromatic images and its impact is presumably complementary to that of $f(101, 101)$.

The obtained *pragmatic MET* is 26696, which, as expected, exceeds the *observed maximal time* 23643. For the MET, we used the X^+ and X^- observed in the measurements. Compared to $p = 1$, we see a significantly smaller error $\epsilon^+ = 240$. In the test set, we saw reasonably tight overestimations from $\widehat{Y}^+(n)$, however, two pronounced underestimations were detected as shown in Fig. 12a. These results were actually obtained without the Cook's input quality assurance procedure described in Section 3.4.

By a more careful analysis of the input samples, we saw that some of them have a Cook's distance significantly larger than all other samples. Our quality assurance procedure has moved the two samples from the training set to the test set and we re-constructed the model for $p = 6$ (results are reported in Table 1). The obtained error was then reduced to $\epsilon^+ = 52$ and we observed a tight overestimation for all samples in the test set as shown in Fig. 12b. The normality test of the residual returned 26% likelihood on the training set. The MET has become less accurate, reaching 28048. This is presumably explained by the degraded stability of regression accuracy for the bad samples (*i.e.,* the ones with the large Cook's Distance); the sample that provided X^+ and maximal Y was among such samples. This sample corresponded to a monochromatic image of exceptionally large size, whereas a vast majority of

p	b^-		b^+		X^-		X^+		$(bX)^+$	
	×	✓	×	✓	×	✓	×	✓	×	✓
(Constant)	409.660	143.83	637.29	194.18	1	1	1	1	637	194
$f(271, 244)$	0.010	0.0011	0.011	0.0011	3688	60480	1818500	1852200	19752	2204
$f(90, 30)$	0.055	0.0064	0.070	0.0069	28	2581	27215	129330	1917	901
$f(101, 101)$	−49.506	0.0516	−11.530	0.1058	0	6	5	1635	0	173
$f(80, 81)$	−113.010	−14.454	−26.009	−5.958	0	0	2	5	0	0
$f(409, 410)$	0.013	0.9401	0.022	3.368	0	8	192280	37	4150	124
ϵ^+	–	–	–	–	–	–	–	–	240	52
Pragmatic MET	–	–	–	–	–	–	–	–	26696	28048

Table 1: Stepwise Regression results in the training set for $p = 6$ and $\alpha = 0.05$; results are obtained with (✓) and without (×) outliers handling using the Cook's Distance metric

other samples were color images of much smaller size. In practice, such a situation should be avoided by well prepared measurement data. For technical reasons, we could not repair the situation by adding more measurements, but we decided to keep the bad samples for illustrative purposes. An observation that should be made, though, is that the instability did not result in unsafe underestimation, but instead in a safe overestimation.

By experimenting with larger values of p, we found that the model with $p = 9$ was optimal. The error ϵ^+ was reduced to 35 and stopped improving, thus showing saturation. With more variables, a degradation of model tightness was observed, probably because the new parameters b started getting 'blurred', showing a Δb much larger than b. The optimal $p = 9$ yielded 97% error normality likelihood, with tight overestimations for all measured samples except for the bad ones; the resulting MET was 56538, not particularly tight due to bad samples, but still safe. From (6), this estimate corresponded to $\mathcal{P}r > 0.725$ – for $\alpha = 0.05$. The MET estimations using the same model at $\mathcal{P}r > 0.999725$ amounts to 58859. As it is shown in Fig. 13a, the corresponding maximal regression model showed tight overestimations over the measurements not only for $\alpha = 0.05$ but also for $\alpha = 0.00005$. In Fig. 13b, the histogram of residual error is shown that is close to the normal distribution. This is in line with the 97% estimate of normality test and it justifies the use of statistical formulas associated with linear regression.

6.2.3 Extended Model using PCA

In this section, we discuss the results of additional analyses performed on the same data (from the JPEG execution time measurements) using the PCA approach. Our goal was to get more insights on the execution time bounds, and to study the strength and weaknesses of the two methods. A detailed discussion on this matter is provided later in Section 6.2.4.

We follow the same analysis steps performed for stepwise regression. The α^{pc} was first tuned ($\approx 15\%$), in order to obtain $p = 6$, *i.e.,* to have 5 predictors. Recall that for PCA, $(1-\alpha^{pc})$ expresses the percentage of variance we want to retain in our model, *i.e.,* in this case, we retain 85% of the original variance. Table 2 shows the identified variables and the corresponding error ϵ^+ and MET. It is worth mentioning that in contrast to stepwise regression, predictors identified by PCA have no clear meaning with respect to the original ones, *i.e.,* they correspond to combinations of predictors compressed into one, as explained in Section 3.2.2[10]. The obtained *pragmatic MET* is 14828, which underestimates the *observed maximal time* (23643). As expected, the observed error was also quite high $\epsilon^+ = 1155$. This potentially suggests that the number of predictors to consider for PCA should be higher than stepwise regression to enhance the quality of the obtained MET.

Further explorations showed that a model with $p = 18$ variables is optimal with respect to the value and the quality of the obtained error. More precisely, the obtained error was $\epsilon^+ = 41$ and the normality test on the residuals was $\approx 96\%$ in this case (see table 3 for the complete set of predictors). Adding more variables enhances the quality of the model (reduces the error), but degrades the normality of the residuals as shown in Fig. 14. The latter shows the impact of increasing the number of predictors on the value of the error (ϵ^+) and on the normality likelihood of the residuals. For instance with 20 variables we observed an error $\epsilon^+ = 34$, however the normality likelihood of residuals was only 22%. The figure confirms that $p = 18$ is a good compromise regarding the two criteria related to the error (value and normality). Note that with this choice we violate the rule of thumb (14 variables). We believe

[10] Matching PCA predictors back to the original ones (or to the ones obtained by using stepwise regression) is feasible, but requires some additional work. For simplicity, we choose not to show it here.

p	b^-	b^+	Z^-	Z^+	$(bZ)^{\pm}$
(Constant)	677	924	1.0	1.0	924
Z_1	−35.12	23.98	−3.73	8.82	211
Z_2	−9.74	61.07	−5.2	15.66	956
Z_3	−65.98	33.93	−6.06	8.86	400
Z_4	−107.6	10.9	−6.7	5.9	725
Z_5	273.4	460.7	−2.05	5.2	2437
ϵ^+	–	–	–	–	1155
Pragmatic MET	–	–	–	–	14828

Table 2: PCA Results in the Training Set for $p = 6$ and $\alpha = 0.05$; results are obtained with outliers handling

that this can be tolerated, as far as the difference is not too important.

It is worth mentioning that since the normality likelihood of residuals is not monotonically decreasing, we cannot be sure that $p = 18$ maximizes it. We can argue that this choice is inline with the principle of parsimony, *i.e.,* it ensures lower predictors. Fig. 14 indeed shows that: (i) less variables cannot provide better performance (error value/normality trade-off), and (ii) even if with more variables we may obtain a better performance (which is very unlikely given the experiment results), we can settle for the best performance providing the least predictors, *i.e.,* $p = 18$. The pragmatic MET using $\alpha = 0.05$ was equal to 39153 in this case, and we obtained tight overestimation as shown in Fig. 15a. From (6), this estimate corresponded to $\mathcal{P}r > 0.5$ (for $\alpha = 0.05$)[11]. The MET estimations using the same model at $\mathcal{P}r > 0.9995$ (for $\alpha = 0.00005$) amounts to 46786 and it shows a tight overestimation as illustrated also in Fig. 15a. Similarly, in line with the obtained normality likelihood, the histogram of residuals in Fig. 15b shows a normal-like shape.

6.2.4 Observations and Discussion

Different observations can be made out of the previous analyses regarding the two proposed methods.

Number of predictors. One can see that compared to PCA, stepwise regression retains less variables in the model (*e.g.,* $p = 8$ out of $|P_{pp}| = 95$) with tight overestimations and a small and normally distributed error ($\epsilon^+ = 35$). The reason is that the stepwise regression algorithm takes into account execution time measurements Y when identifying the most relevant predictors. Recall that, in this algorithm, the decision to keep a predictor or to remove it, is performed mainly with respect to its impact on Y. In contrast, PCA requires more variables (*e.g.,* $p = 18$ for the same example) to produce tight overestimations. This number of variables was the best observed trade-off, albeit it produces a higher error $\epsilon^+ = 41$ than stepwise regression. This could be explained by the fact that PCA does not take into account Y for predictors identification. It performs a dimensiality reduction on the predictors independently of their impact on the execution time Y.

Experiments easiness. The last point can be seen as an advantage in favor of PCA when computing different MET estimations for different target platforms. In the case of PCA, since it is independent from Y, predictors identification is performed once (on $\mathbf{X}^{\text{train}}$, obtained by functional simulation). However, since in such situation Y will be measured on different target platforms, stepwise regression will be repeated for each new measurement. Recall the setting explained in Section 4.

Estimation tightness. Another advantage of PCA is that it produces tighter MET than stepswise regression. For instance for $\alpha = 0.00005$ in the JPEG case study, the obtained MET using PCA was 46786, whereas it was much larger (58859) with stepswise regression. As stated earlier, adding more variables in the case of stepwise regression, impacts the quality of the new parameters b, *i.e.,* they get 'blurred', showing a Δb much larger than b. However, since these variables are computed to be always orthogonal in PCA, the impact on the interval Δb is almost neglected and the pragmatic MET estimation do not overshoot.

Results interpretability. The two methods can be further distinguished with respect to the interpretability of the obtained model. From this perspective, models produced by stepwise regression are easier to interpret as they contain predictors that represent a subset of the original set and which can be straightforwardly matched with their code blocks counter-parts. This is less evident when using PCA since predictors are obtained, in this case, by combining together original predictors. Thus, more work is required to match them back to the underlying blocks of code.

[11] The small probability for the same α in the case of stepwise regression is due to the fact that (6) takes into account the number of predictors p, which is greater in this case.

p	b^-	b^+	Z^-	Z^+	$(bZ)^\pm$
(Constant)	1011	1203	1	1	1203
Z_1	−1.53	1.21	−3.73	8.84	10
Z_2	22.28	25.63	−5.17	15.71	402
Z_3	−7.19	−3.12	−6.10	8.52	43
Z_4	29.66	34.10	−6.50	6.72	229
Z_5	549.09	620.60	−1.56	6.68	4151
Z_6	145.27	160.46	−2.73	2.53	406
Z_7	−153.04	−122.85	−1.98	4.80	304
Z_8	−58.85	−44.10	−1.22	3.56	71
Z_9	−85.02	−43.11	−2.41	2.24	205
Z_{10}	39.97	72.23	−1.43	1.50	109
Z_{11}	7.14	26.06	−1.77	0.99	25
Z_{12}	124.59	481.33	−1.57	0.37	179
Z_{13}	−215.17	−83.59	−0.88	0.97	190
Z_{14}	−226.68	−107.66	−0.50	1.22	114
Z_{15}	62.76	409.86	−0.78	0.24	101
Z_{16}	−85.04	184.82	−0.35	0.43	79
Z_{17}	−630.34	−191.97	−0.23	0.14	148
ϵ^+	–	–	–	–	41
Pragmatic MET	–	–	–	–	39153

Table 3: PCA Results in the Training Set for $p = 18$ and $\alpha = 0.05$; results are obtained with outlier handling

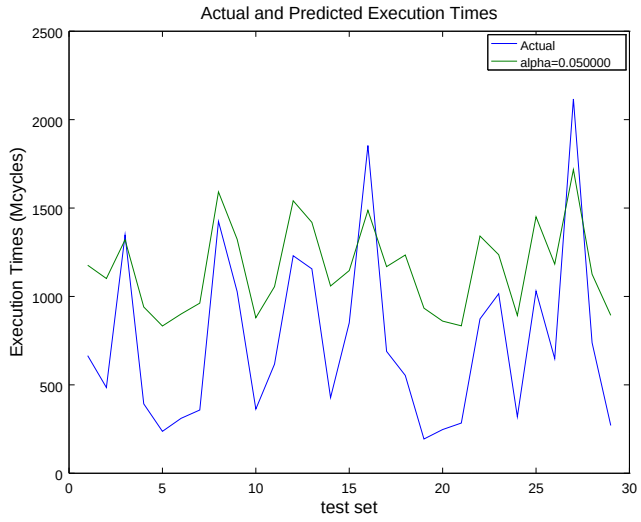
Common shortcoming. Finally, we recall that the pragmatic MET calculation for the two methods is likely to incur extra overestimation by including unfeasible paths. In fact, this is presumably the case for the model obtained with stepwise regression for $p = 6$, as the calculation in Table 1 may combine a relatively large byte and block count that is typically required for colored images with pessimistic contributions of the predictors representing monochromatic images. With the IPET approach this possibility would be excluded and a more realistic worst-case vector $\mathbf{X}$ would have been obtained. A lower bound on hypothetical IPET results with that model is 25764, which is calculated as the observed maximum value of $\widehat{Y}^+(n)$.

7 Related Work

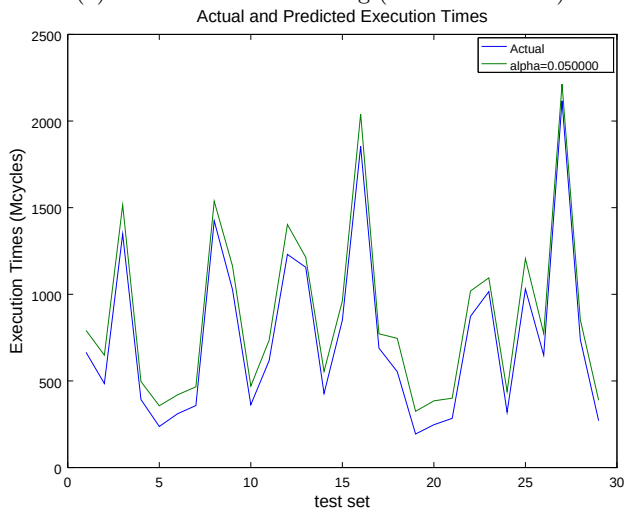
Historically, linear regression and other model fitting techniques have been mostly used to predict *average*, not conservative, software performance in terms of execution time, *e.g.,* [6], and energy consumption. A regression for *maximal* execution time was proposed in [10], but, unlike our work, their regression model is not based on statistical techniques. Instead, the authors sketch an *ad hoc* linear programming based approach and they admit that additional future work is still required. In contrast to our work, *all* potential predictors are included in the model, instead of a small subset of the significant ones, and therefore their techniques presumably require many more measurements to avoid overfitting, and more costly calculations to estimate all parameters. The coverage criteria are based on existence of an hypothetical exact model with a large enough number of variables, which should be known, whereas we tolerate presence of error and estimate the coverage probabilistically. On the other hand, they have showed how a maximal regression model, such as ours, could be combined with existing complementary WCET techniques for calculating tighter execution time bounds than our pragmatic MET formula.

In [8], regression analysis is used in the context execution time prediction. The proposed method, called SPORE, considers polynomial regression models, as opposed to our work. Although it fundamentally differs from our work, the SPORE method is faced with similar challenges, namely, identifying a relevant compact set of predictors. Two ways are proposed in [8], which are both variants of the LASSO (least absolute shrinkage and selection operator) [7] statistical technique. However, since used for prediction, the selection method seems to give an important weight to the computation cost of each predictor. This may result in eliminating relevant predictors. Furthermore, no clear indication is given regarding the choice of the input data sample and its impact on the accuracy of the obtained model.

Among the works on statistical WCET analysis, we only consider those that take into account non-random input data parameters. One of the methods proposed in [4] is to enumerate execution paths of the program and treat them separately, however this approach is appropriate only for programs with simple control flow structure. Another approach is proposed in [1]. In that

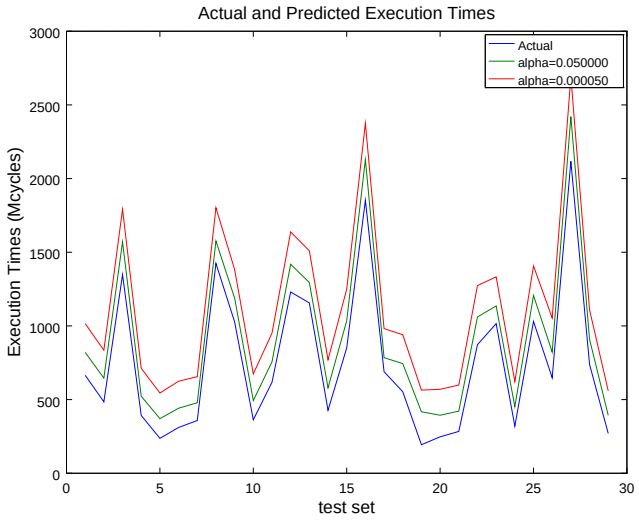


(a) without outlier handling (Cook's Distance)

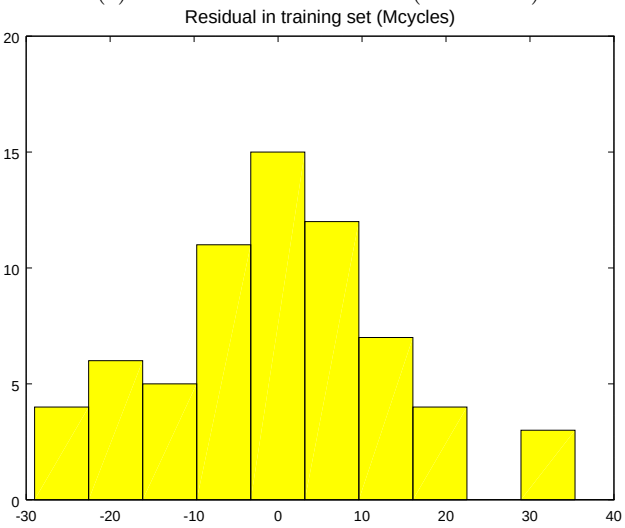


(b) with outlier handling (Cook's Distance)

Fig. 12: Stepwise Regression results for $p = 6$ and $\alpha = 0.05$; results are obtained on the test set



(a) Obtained execution times (on test set)



(b) residuals (on training set)

Fig. 13: Stepwise Regression results for $p = 8$; results obtained with outlier handling

work, program paths are modeled using 'timing schema', which split the program into code blocks. The WCET distributions of each block are measured separately and then the results for the different blocks are combined. However, this approach requires executing instrumentation points together with timing measurements, which introduces the unwanted probe effect.

8 Conclusions

In this paper, we have presented a new regression-based technique for the estimation of probabilistic execution time bounds. Unlike WCET analysis techniques, it cannot ensure safe estimates at very high probability levels, but it can be utilized for preliminary WCET estimates and in the context of non safety-critical systems. We have described a complete methodology for model construction, which includes two algorithm for identifying the proper model variables and an algorithm for finding conservative model parameters. So far, this technique was tested with only one program, a JPEG decoder, through a limited set of measurements. Nevertheless, it has shown promising results, by giving tight overestimations in the tests.

In future work, it would be interesting to combine the presented regression technique with a complete WCET analysis flow using implicit path enumeration techniques and to study how to model hardware effects using specially defined predictors, similarly to [10]. An investigation of possible connections between regression and

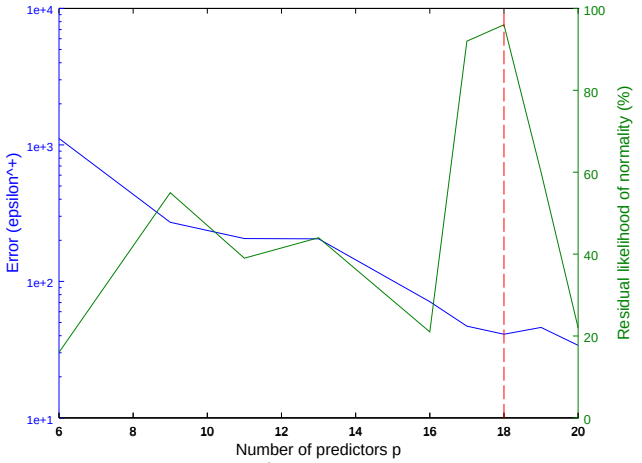
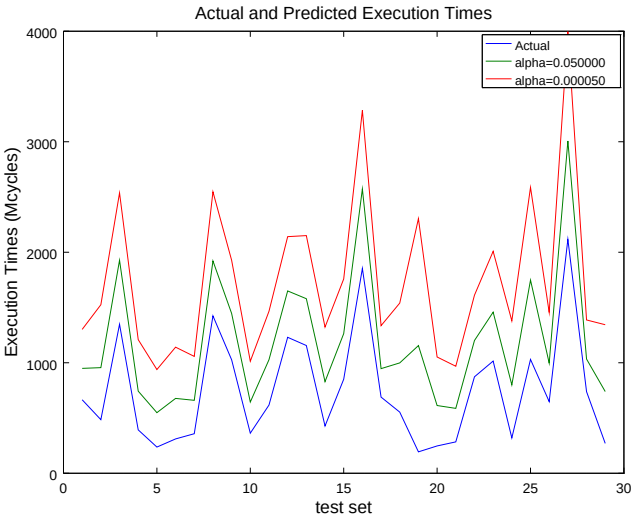
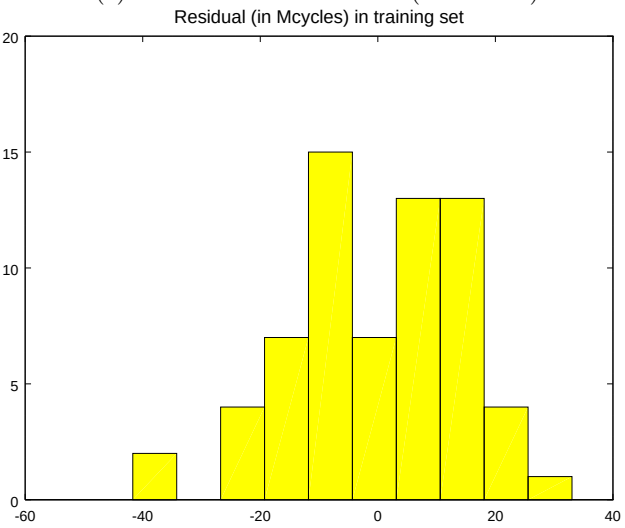


Fig. 14: Value of the ϵ^+ (Log scale) and residuals normality likelihood for different numbers of predictors



(a) Obtained execution times (on test set)



(b) residuals (on training set)

Fig. 15: PCA results for $p = 18$; results obtained with outlier handling (Cook's Distance)

extreme value theory is also needed, in order to produce high-probability bounds, as in [4]. Finally, we observed that by putting too many variables into the multi-variate regression analysis the estimation of model parameters is weakened, which manifests in 'blurred' parameter confidence intervals. Therefore, it is interesting to investigate splitting the program into blocks characterized by a smaller set of variables and combining the results by their joint distributions, as in [1].

References

1. Bernat G, Colin A, Petters SM (2002) WCET analysis of probabilistic hard real-time system. In: Proc. RTSS'02, IEEE, pp 279–288
2. Bernat G, Burns A, Newby M (2005) Probabilistic timing analysis: An approach using copulas. J Embedded Comput 1(2):179–194, URL `http://dl.acm.org/citation.cfm?id=1233760.1233763`
3. Betts A, Bernat G (2006) Tree-based WCET analysis on instrumentation point graphs. In: Proc. ISORC'06, IEEE, pp 558–565
4. Cucu-Grosjean L, Santinelli L, Houston M, Lo C, Vardanega T, Kosmidis L, Abella J, Mezzetti E, Quiñones E, Cazorla FJ (2012) Measurement-based probabilistic timing analysis for multi-path programs. In: Proc. ECRTS'12, IEEE, pp 91–101
5. Draper NR, Smith H (1981) Applied regression analysis (2nd edition). Wiley
6. Eskenazi EM, Fioukov AV, Hammer DK (2004) Performance prediction for component compositions. In: CBSE'04, Springer, pp 280–293
7. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference and prediction (2nd edition). Springer, URL `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`
8. Huang L, Jia J, Yu B, Chun BG, Maniatis P, Naik M (2010) Predicting execution time of computer programs using sparse polynomial regression. In: Proc. NIPS'10, Curran Associates Inc., USA, pp 883–891, URL `http://dl.acm.org/citation.cfm?id=2997189.2997288`
9. Jolliffe I (2002) Principal Component Analysis. Springer Series in Statistics, Springer, URL `https://books.google.fr/books?id=_olByCrhjwIC`
10. Lisper B, Santos M (2009) Model identification for WCET analysis. In: Proc. RTAS'09, IEEE, pp 55–64

11. Poplavko P, Nouri A, Angelis L, Zerzelidis A, Bensalem S, Katsaros P (2017) Regression-based statistical bounds on software execution time. In: Verification and Evaluation of Computer and Communication Systems - 11th International Conference, VECoS 2017, Montreal, QC, Canada, August 24-25, 2017, Proceedings, pp 48–63, DOI 10.1007/978-3-319-66176-6_4
12. Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenström P (2008) The worst-case execution-time problem – overview of methods and survey of tools. ACM Trans Embed Comput Syst 7(3):36:1–36:53