Statement-Level Timing Estimation for Embedded System Design Using Machine Learning Techniques - Additional Material

VITTORIANO MUTTILLO, PAOLO GIAMMATTEO, VINCENZO STOICO, DEWS Centre of Excellence, University of L'Aquila, L'Aquila, Italy

1 INTRODUCTION

This document presents additional material related to the paper "Statement-Level Timing Estimation for Embedded System Design Using Machine Learning Techniques" submitted to The International Conference on Performance Engineering (ICPE 2021).

2 ADDITIONAL MATERIAL

The approach proposed in ICPE work performs statement-level time performance estimation using statistical analysis and approximate predictions. Several features have been chosen to provide a statistical analysis of data collected (i.e., distribution parameters). Two different Python scripts have been created in order to automatize the statistical analysis of all CSV file, integrated into the ML activity. To guarantee unbiased data and correct ML training activities, a feature analysis is applied to the output framework dataset (i.e., 33 features) that can be clustered w.r.t. number of code line (SLOC), input data type (IDT), Halstead Complexity (HC), Cyclomatic Complexity (CC), functions reachability (FR) and program profiling (PP). The considered dataset features are the following:

(1) *SLOC*: the number of lines in the text of the program's source code;
(2) *Scalar Input (IDT)*: the number of scalar input in the code;
(3) *Range Scalar Values (IDT)*: a custom scalar data type range classification (int8, int16, int32 and float) ranging from 1 (low value), 2 (medium value) to 3 (high value);
(4) *Scalar Index Input (IDT)*: the number of scalar used as array index in the code;
(5) *Range Scalar Index Values (IDT)*: a custom scalar index data type range classification (int8, int16, int32 and float) ranging from 1 (low value), 2 (medium value) to 3 (high value);
(6) *Array Input (IDT)*: the number of input array variables inside the code;
(7) *Range Array Input (IDT)*: a custom feature which indicates whether the array variables range from negative values or not;
(8) *Total Operators (HC)*; the total number of operators in the code;
(9) *Distinct Operators (HC)*: the number of unique operators in the code;
(10) *Total Operands (HC)*; the total number of operands in the code;
(11) *Distinct Operands (HC)*: the number of unique operands in the code;
(12) *Program Length (HC)*: the total number of operator occurrences and the total number of operand occurrences;
(13) *Vocabulary Size (HC)*: the total number of unique operator and unique operand occurrences;
(14) *Program Volume (HC)*: it represents the size, in bits, of space necessary for storing the program;
(15) *Effort (HC)*: is the amount of mental effort required to recreate the software;

Author's address: Vittoriano Muttillo, Paolo Giammatteo, Vincenzo Stoico, DEWS Centre of Excellence, University of L'Aquila, L'Aquila, Italy, {vittoriano.muttillo,paolo.giammatteo}@univaq.it,vincenzo.stoico@graduate.univaq.it.

(16) *Bugs Delivered (HC)*: it estimates how many bugs you are likely to find in the system;
(17) *Time to Implement (HC)*: it gives the time in seconds that it should take a programmer to implement the code;
(18) *Difficulty Level (HC)*: this parameter shows how difficult to handle the program is;
(19) *Program Level (HC)*: is used to rank the programming languages and consider the level of abstraction provided by the programming language;
(20) *Decision Point (CC)*: the number of decision point in the code;
(21) *Global Variables (CC)*: the number of global variables in the code;
(22) *Loop (CC)*:the number of loops in the code;
(23) *Goto (CC)*: the number of Goto in the code;
(24) *Assignment (CC)*: the number of Assignment in the code;
(25) *Exit Point (CC)*: the number of decision point;
(26) *Function Call (CC)*: the number of function call in the code;
(27) *Pointer Dereferencing (CC)*: total number of dereference operators;
(28) *Cyclomatic Complexity (CC)*: classical metric used to indicate the complexity of a program;
(29) *Coverage (FR)*: the reachability coverage analysis, which over-approximates the number of functions that can be called from the considered function;
(30) *Total Functions (FR)*: total number of functions called;
(31) *Executed C Instr (PP)*: the number of C instructions executed by the processor;
(32) *Executed Ass Instr (PP)*: the number of Assembly instructions executed by the processor;
(33) *Clock Cycle Effective (PP)*; the number of clock cycles performed by the processor, the desired output of the model.

Once the dataset is created, four regressor models are considered in order to perform the feature analysis.

Random Forest Regressor: Random Forest is a supervised learning algorithm [1] which uses ensemble learning method both for classification and regression. Random forest is a bagging and not a boosting technique. The trees in random forests are run in parallel. There is no interaction between these trees while building the trees. It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the prediction (in this case a regression) of the individual trees.

Extra Trees Regressor: Extremely Randomized Trees Regressor (Extra Trees Regressor) [2] is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a forest, in order to output it's prediction result. Conceptually, it is very similar to a Random Forest Regressor and it only differs from the latter in the manner of construction of the decision trees in the forest. The two ensembles have a lot in common. Both of them are composed of a large number of decision trees, where the final decision is obtained taking into account the prediction of every tree. Furthermore, both algorithms have the same growing tree procedure. The two main differences with Random Forest ensemble method are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample to grow the trees (rather than a bootstrap replica).

Gradient Boosting Regressor: Gradient Boosting is a machine learning technique both for regression and classification problems that produce a prediction model $\hat{y} = F(x)$ in the form of an ensemble of weak prediction models [3]. This technique builds a model F in a stage-wise manner and generalizes the model by allowing optimization of an arbitrary differentiable loss function L. Gradient boosting basically combines weak learners into a single strong learner in an iterative way. As each weak learner is added, a new model is fitted to provide a more accurate estimate of the response variable. The new weak learners are maximally correlated with the negative gradient of the loss function, associated with the whole ensemble. The idea of gradient boosting is to combine a group of relatively weak prediction models to build a stronger prediction model. Gradient Boosting Regressors are additive models whose prediction y_i for a given input x_i is of the following form:

$$\hat{y}_i = F_M(x_i) = \sum_{m=1}^{M} h_m(x_i) \tag{1}$$

where the h_m are estimators called weak learners in the context of boosting. Using a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of known values of x and corresponding values of y, the aim is to find an approximation $\hat{F}(x)$ to a function $F(x)$ that minimizes the expected value of some specified loss function $L(y, F(x))$:

$$\hat{F}(x) = \arg\min_{F} \mathbb{E}_{x,y}[L(y, F(x))]. \tag{2}$$

Ada Boost Regressor: The concept of Ada Boost [4] is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) in order to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights $w_1, w_2, ...w_N$ to each of the training samples. Initially, those weights are all set to $w_i = 1/N$ so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the re-weighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence. It can be used both for classification and regression problems.

The ML techniques considered, among the most used in literature [**? ?**], range from regression trees to SVM, and linear regressors, with the aim to identify which of these can better predict the timing performance metric. At this stage of the work, the Matlab app Regressor Learner [5] has been used. In particular, we considered five ML methodology, according to those available: Linear Regression (LR), Fine Trees (FT), Boosted Trees (BOT), Bagged Trees (BAT), and Support Vector Machine (SVM).

Linear Regression Models - Linear - LR Linear regression models [6] have predictors that are linear in the model parameters, are easy to interpret, and are fast for making predictions. However, the highly constrained form of these models means that they often have low predictive accuracy compared to the others. At the moment, this analysis consider a simple linear regression of the kind:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{3}$$

where β_0 is the intercept, β_1 is the slope (or regression coefficient) and ϵ is the error term. In terms of flexibility, the model doesn't have much but it is a preliminary approach. Regression Learner App uses the *fitlm* function to train *Linear* model option.

Regression Trees - Fine Tree - FT Regression trees [6] are easy to interpret, fast for fitting and prediction, and low on memory usage. The Matlab App Regression Learner gives the possibility to choose between different kind of regression trees. In this work the *Fine Tree* option was chosen, which means high flexibility, with many small leaves for a highly flexible response function (the minimum leaf size is 4). A Fine Tree with many small leaves is usually highly accurate on the training data. However, the tree might not show comparable accuracy on an independent test set. A very leafy tree tends to overfit, and its validation accuracy is often far lower than its training (or resubstitution) accuracy. The Regression Learner App uses the *fitrtree* function to train regression trees, with the parameter *Minimum leaf size* equal to 4 and no splitting criteria for surrogate nodes.

Ensembles of Trees - Boosted Trees - BOT As described in previous paragraph, this ensemble model combine results from many weak learners into one high-quality ensemble model. The approach involves a least-squares boosting methodology with regression tree learners [6]. Regression Learner App uses the *fitrensemble* function to train ensemble models and gives the possibility to set three different parameters: the *Minimum leaf size*, the *Number of learners* and the *Learning rate*, which are respectively 8, 30 and 0.1.

Ensembles of Trees - Bagged Trees - BAT It is another kind of ensemble model with regression tree learners, this time with a bootstrap aggregating or bagging approach [6]. Regression Learner App uses the *fitrensemble* function to train ensemble models. Here the parameter to set are two: the *Minimum leaf size* and the *Number of learners*, which are respectively 8 and 30.

Support Vector Machines - Linear - SVM Support vector machines (SVMs) are a set of supervised learning methods used both for classification and regression [6]. Among the advantages of support vector machines approach, it is effective in high dimensional spaces and in cases where number of dimensions is greater than the number of samples. Furthermore, it uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. On the other hand, among the disadvantages it suffers in performance when the number of features is much greater than the number of samples. Linear SVMs are easy to interpret, but can have low predictive accuracy. While, non-linear SVMs are more difficult to interpret, but can be more accurate. Regression Learner App uses the *fitrsvm* function to train SVM regression models and, in this case, the parameter *Kernel function* is set to *Linear*. Three General Purpose Processors

(GPPs), and their related ISS, have been analyzed that allows to consider a more performing architecture that relies on external memory and cache:

- *LEON3*, a 32-bit synthesizable RISC soft-microprocessor compatible with SPARC V8 architecture. It has a configurable seven-stage pipeline, an Harvard architecture, and a default simulated system clock of 50 MHz (in our work it has been used a 75 MHz clock). The evaluation version of TSIM/LEON3 implements 2*4 KiB caches (not removable), with 16 bytes per line with Least-Recently-Used (LRU) replacement algorithm. It has 8 register windows, a RAM size of 4096 KiB and a Rom size of 2048 KiB. Cobham Gaisler offers TSIM System Emulator [7] as an accurate emulator of LEON3 processors. Benchmark functions have been compiled, with the Bare-C Cross-Compiler (BCC) for LEON3 processors [8]. It is based on the GNU GCC C/C++ compiler tools and the New-lib standalone C-library [9] (in this work the default optimization flag $-O0$ has been used);
- *Intel 8051 micro-controller*, a 8-bit micro-controller with MCS-51 CISC instruction set and Harvard Architecture; 8051 presents a PROM non-volatile memory which contains program instruction and a RAM memory for data. I8051 registers are 8-bit registers. The University of California has developed a project centered on 8051 microprocessor, which provides a number of tools useful for simulating C code on Intel 8051 microprocessor.

The project name is Dalton and was developed by the Department of Computer Science of the University of California [10]. The Dalton Instruction Set Simulator (ISS) allows a user to simulate programs written for the 8051 and provides statistics on instructions executed, instructions executed per second, clock cycles required by the 8051, and average instructions per second for an 8051 executing the same program. The functions were compiled, with SDCC (Small Device C Compiler) [11]. SDCC is a free open source C compiler suite designed for 8 bit Microprocessors. To do a proper simulation, during the compilation two options was specified: *−mmcs51* and *−iram-size 128*. The first one refers to the family of the microprocessor while the second to the dimension of the internal ram.

- *picoPower CMOS 8-bit AVR ATmega328/P*, a single-chip microcontroller created by Atmel in the megaAVR family. It has a modified Harvard architecture with a 8-bit RISC-based processor core, 32KB ISP flash memory with read-while-write capabilities, 1024B EEPROM, 2KB SRAM, and 32 general purpose working registers. The SimulAVR [12] program, part of the GNU Project, is a simulator for the Atmel AVR family of microcontrollers. The core of SimulAVR is functionally a library. This library is linked together with a command-line interface to create a command-line program. The functions have been compiled with the command: *avr-gcc -o0 -mmcu=atmega329*.

Results have been presented in the ICPE PAPER.

ACKNOWLEDGMENTS

This work has been partially supported by the ECSEL RIA 2017-783162 FitOptivis and the ECSEL-JU 2018-826610 COMP4DRONES projects.

REFERENCES

[1] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.
[2] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr 2006.
[3] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001.
[4] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.
[5] *Regression Learner App*, 2020 (accessed: 15.03.2020). https://it.mathworks.com/help/stats/regression-learner-app.html.
[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.
[7] *TSIM2 ERC32/LEON simulator*, 2020 (accessed: 15.03.2020). https://www.gaisler.com/.
[8] *LEON Bare-C Cross Compilation System (BCC)*, 2020 (accessed: 15.03.2020). https://www.gaisler.com/index.php/products/operating-systems/bcc.
[9] *GCC, the GNU Compiler Collection*, 2020 (accessed: 15.03.2020). http://gcc.gnu.org/.
[10] *Dalton Project: 8051 microcontroller, University of California*, 2020 (accessed: 15.03.2020). http://www.ann.ece.ufl.edu/i8051/.
[11] *SDCC - Small Device C Compiler*, 2020 (accessed: 15.03.2020). http://sdcc.sourceforge.net/.
[12] Martin Becker, Ravindra Metta, Rentala Venkatesh, and Samarjt Chakraborty. Scalable and precise estimation and debugging of the worst-case execution time for analysis-friendly processors. *International Journal on Software Tools for Technology Transfer*, 02 2018.