This article is published in Real-Time Systems and published form is available at:
https://doi.org/10.1007/s11241-020-09343-2

Estimating WCET using prediction models to compute fitness function of a genetic algorithm

Syed Abdul Baqi Shah[1,2] · **Muhammad Rashid**[3] · **Muhammad Arif**[4]

Abstract
Genetic algorithms can be used to generate input data in a real-time system that produce the worst-case execution time of a task. While generating the test data, the fitness function is normally evaluated using a cycle-accurate simulator of the processor architecture, which consumes a significant computational effort and time. We propose to replace the simulator-based actual execution with a predictive model that is trained using the samples acquired on the simulator. The feasibility of this proposal was evaluated using four distinct predictive models, namely artificial neural networks, generalized linear regression, gaussian process regression and support vector regression. The results obtained on the four benchmarks Bubble sort, Insertion Sort, Gnome sort and Shaker sorts indicate that the proposed use of prediction models can significantly reduce the temporal verification time. The time gain achieved is up to 17.7 times and the best accuracy achieved is 98.5%.

Keywords Temporal verification · Real-time systems · Genetic algorithm · Prediction models · Worst-case execution time · Test data

1 Introduction

Real-time systems (RTS) are conditioned to satisfy stringent timing constraints and their temporal verification is of paramount importance. Nevertheless, RTS do exhibit a certain variation of execution times to accomplish a certain task (Abella

✉ Syed Abdul Baqi Shah
syed_abdul_baqi.shah@mymail.unisa.edu.au; sashah@uqu.edu.sa

[1] School of Electrical Engineering, University of South Australia, Adelaide, Australia

[2] Science and Technology Unit, Umm Al-Qura University, Makkah, Kingdom of Saudi Arabia

[3] Department of Computer Engineering, Umm Al-Qura University, Makkah, Kingdom of Saudi Arabia

[4] Department of Computer Science, Umm Al-Qura University, Makkah, Kingdom of Saudi Arabia

et al. 2015). In this regard, the main factors influencing the execution times are: (a) the underlying hardware complexity (e.g., modern processors incorporating speculative components, shared caches and out-of-order executions), (b) the real-time software system architecture (i.e., the structure of application) and (c) the input data. In a given system setup with fixed hardware and software, the input test data becomes the critical factor for variations in execution time (Kozyrev 2016). Therefore, the generation of input test data that can estimate the worst-case execution time (WCET) plays a vital role for the temporal verification of real-time systems.

Traditionally, WCET estimation is performed by using static analysis or measurement-based analysis. Static analysis employs some formal methods to analytically estimate the WCET and thus suffers from over estimations (Reineke and Wilhelm 2016). On the other hand, the measurement-based analysis requires an extensive testing of the RTS to ensure the worst-case scenario. Therefore, various evolutionary techniques, such as genetic algorithms (GAs), are being incorporated in measurement-based analysis (Surendran and Samuel 2016). In GAs, the problem of searching the worst-case input data from a huge input space is tackled as an optimization problem. Initial random input data points are evolved to worst-case execution-time data, based on certain fitness evaluation criteria (Kudjo et al. 2017).

The GA-evolution process requires several fitness evaluations, which implies the repeated executions of the real-time systems software under test (RTS-SUT). The execution of an RTS-SUT is an expensive activity in terms of time and resources. Moreover, the actual execution is only possible when the target hardware system is finalized and available. In the absence of a target hardware, a cycle-accurate simulator is used. Nevertheless, the timing cost is further multiplied when the simulators are used. The time and resource intensive nature of GA-evolution process provides the motivation for prediction models (Haftka et al. 2016). A prediction model replaces the actual execution of an RTS-SUT for fitness evaluation with an instantaneous prediction of the fitness value against a given input. As a result, the temporal verification time can be reduced drastically by using prediction models.

Various prediction models can be used to assist the GA-evolution process (Agresti 2013). In this article, we present a framework in which an appropriate prediction model can be used to predict the execution time of a problem in hand. Selection of an appropriate model depends on the complexity and type of the application problem. Some typical examples of prediction model are: Artificial Neural Network (ANN) (Hagan et al. 2014; Hagan and Menhaj 1994), Generalized Linear Regression Model (GLM) (Fahrmeir and Tutz 2013), Gaussian Process Regression Model (GPR) (Rasmussen and Williams 2006) and Support Vector Regression Model (SVR) (Vapnik 2000; Smola and Scholkopf 2004). The initial results for ANN as a prediction model, in a GA-based test data generation methodology, are presented in Shah (2017). However, a detailed study of prediction models, to assist the GA-evolution process for temporal verification of real-time systems, has not been performed yet. Moreover, the use of prediction models in GA-based timing analysis is a novel idea and thus requires a detailed evaluation of prediction models.

This article performs a detailed study of various prediction models for temporal verification of real-time systems by using certain defined performance measures. The defined performance measures in this article are: prediction performance, isolated

timing performance, evolution performance, integrated timing performance and the quality of test data. All the aforementioned target prediction models (ANN, GLM, GPR and SVR) are required to be trained through a cycle-accurate simulator of the hardware. The training of a model requires some adequate input-output data, which are obtained by running the simulator with the randomly selected set of input data. Once the prediction model is trained through a simulator, it can be used for the computation of the fitness function during the GA-evolution process. The prediction of the fitness function, during the GA-evolution process, eliminates the need for actual execution of the application program, either on a simulator or hardware.

We evaluate our proposal of estimating execution times using prediction models for fitness evaluation in GAs using four sorting algorithms as benchmarks. Sorting algorithms are used because of their deterministic critical-path and their common applications in real-time systems (Puschner 1999). For example, task schedulers, which are important components of hard real-time systems, employ sorting for reliable scheduling of tasks (Alghamdi et al. 2017; Bambagini et al. 2016). Energy-aware schedulers and multiprocessor schedulers are among other common applications of sorting in real-time systems (Zhang et al. 2019; Baruah et al. 2015). The other reason for our selection of sorting algorithms as benchmarks for this study is that their execution paths are impacted by the input data (Puschner 1999), which has the potential to generate patterns in execution times (Puschner 1999). Those patterns in execution times can be used to train the prediction models. This reason renders the sorting algorithms as suitable candidates for the investigation of WCET estimation in our study. Although the experimentation in this study is limited to the sorting tasks, the proposed approach can be extended through further investigation on other benchmarks.

In our study, two benchmarks *Bubble sort* and *Insertion sort* are used from Mälardarlen benchmarks suite (Gustafsson et al. 2010). Mälardarlen benchmarks are commonly used in the research community to evaluate various WCET analysis methods and tools. Mälardarlen benchmarks classify both, Bubble sort and Insertion sort, into input dependent loops category (Gustafsson et al. 2010). In addition to Mälardarlen benchmarks, two more sorting algorithms with similar complexity, *Gnome sort* and *Shaker sort*, are used (Dharmajee Rao and Ramesh 2012). The key characteristic of all the selected benchmarks is to exhibit the influence of input data on execution time of the program.

The paper is organized as follows: Sect. 2 overviews the temporal verification techniques for real-time systems. Section 3 describes the methodology, target prediction models and benchmarks. Section 4 elaborates the selection of certain specific parameters during experiments. Section 5 evaluates the performance of target prediction models using the defined performance measures. Finally, we conclude the article in Sect. 6.

2 Related work

This section overviews state-of-the-art methods and techniques for the process of temporal verification. First, Sect. 2.1 summarizes the static and measurement-based analysis methods. Then, Sect. 2.2 briefly describes the use of evolutionary techniques for the testing of real-time systems.

Finally, Sect. 2.3 highlights the novelty of the proposed approach.

2.1 Basic analysis methods

The comprehensive reviews of the WCET analysis problem, describing the basic analysis methods as well as stat-of-the-art tools, have been performed in Abella et al. (2015), Kozyrev (2016), Nélis et al. (2015). Generally speaking, the main methods in any timing analysis tool are: static analysis, measurement-based analysis and hybrid analysis.

2.1.1 Static analysis

The basic principle is to combine the abstract timing model of the target hardware (low level analysis) with the structural representation of the application program (high level analysis). The execution of an application program on the target hardware is not required. Therefore, the major activities in static methods are: creation of control flow graphs (CFGs), analysis of CFGs, combining CFGs with some abstract model of the target hardware architecture, and finally the estimation of upper bounds for WCET. The significance of a static technique is that it provides safe WCET without executing the program. Furthermore, it removes the need for complex equipment to simulate the hardware and peripherals of the target system. However, the abstract timing models are becoming more complex due to the increasingly advanced hardware features (Sha et al. 2016). Furthermore, a high level analysis of the application program, requiring structural representation of the program for flow analysis, may require some manual annotations (Kelter et al. 2014). To summarize, static analysis techniques are considered as more trustworthy when the target hardware and application programs are sufficiently simple. In other words, when the hardware features get more complex, the scope for issues in the static model increases which ultimately decreases the confidence in static techniques.

2.1.2 Measurement-based analysis methods

The basic principle in measurement-based analysis methods is to execute the application program on the target hardware or a simulator using some input data. Here, the analysis of processor behavior is not required, which is a critical ingredient of static techniques (Kozyrev 2016). The proposed work in this article is also based on the execution of an application program (similar to measurement-based analysis) for the training of a prediction model. However, the presented work is not pure measurement-based in a sense that it utilizes a GA along with a prediction model. Furthermore, due to the huge number of required executions in a pure measurement-based analysis, underestimation of WCET may result. As a consequence, a measurement-based analysis method/technique is usually combined with a static technique. It has resulted in various hybrid approaches, integrating measurement-based analysis methods with the information of all possible execution paths (Rapita 2017).

2.1.3 Hybrid methods

A comprehensive survey on hybrid methods is recently presented in Cazorla et al. (2019). Hybrid methods enhance the confidence level as compared to measurement-based analysis techniques. Hybrid techniques may indicate which execution paths lead to the highest execution times. Therefore, static analysis and dynamic measurements are used together to achieve safe WCET bounds. Model checking techniques can be employed for the effective path coverage during the generation test suites (Bunte et al. 2011). To assure the execution of each path, feasible paths analysis can be performed before the identification of test vectors using search algorithms (Wenzel et al. 2008). Generating an input that can lead to the desired execution path is a complex task even if the execution path for the maximum execution time is known (Law and Bate 2016). The required input search space is too large to exhaustively explore all the possible executions. Taking measurements for the huge search space, from the target hardware or performing simulations, is computationally expensive and time-consuming. Similarly, determining the worst-case input data analytically is a daunting task for the engineers as the application program under analysis may be legacy code developed by a third party. It is quite possible that the legacy applications may not have the required test cases to validate its functionality. The absence of required test cases may complicate the code when new features are added in the existing code. It is particularly true when the legacy code is employed on a new hardware architecture. Therefore, a method is required to generate the worst-case input test data in an affordable time.

2.2 GA-based techniques for temporal verification

GAs have been frequently employed to handle the huge search space (Surendran and Samuel 2016; Kudjo et al. 2017). Section 2.2.1 overviews the GA-based temporal testing techniques. Shortcomings of GA-based techniques are presented in Sect. 2.2.2. Finally, Sect. 2.2.3 describes that a prediction model-based solution can be effectively utilized to mitigate the problems of existing techniques.

2.2.1 GA-based temporal testing techniques for real-time systems

The initial guidelines for using GAs in the context of real-time systems are provided in Wegener et al. (1996). Based on these guidelines, an automated test data generation framework for the testing of safety-critical software systems is presented in Tracey et al. (2002). The authors of Tracey et al. (2002) concluded that even the GA-search process is not sufficient for a thorough and comprehensive test of real-time systems and a combination with other test procedures is essential to develop an effective test strategy. As a result, the test data generation mechanism is further optimized in Bunte et al. (2011) by using a combination of model checking and GAs.

The requirement of model checking support, presented in Bunte et al. (2011), is replaced by extending the GAs with context information Buret et al. (2014) such as

kernel details, other tasks and hardware states. A GA with complex contexts builds a library of valid constructs which are used during the execution of the GA to find interesting solution candidates. The limitation of this solution is its scalability. Providing the context information for large industrial scale projects is expensive in terms of processing power and required engineering effort. Another approach for enhancing the optimization process is presented in Bate and Khan (2011), where the computations are performed for more than one fitness function. The newly introduced fitness functions in Bate and Khan (2011) include cache misses and loop iterations. However, it is found in Bate and Khan (2011) that no single fitness function provides better results across all the test code items, and that the selection of fitness function is dependent on the target environment. The work in Bate and Khan (2011) is further extended for an industrial scenario using a commercial tool (Law and Bate 2016). It investigates the use of search algorithms for the generation of test cases so that the appropriate execution traces are available to support measurement-based timing analysis.

2.2.2 Limitation of existing GA-based techniques

In all the aforementioned research (Rapita 2017; Law and Bate 2016; Wegener et al. 1996; Tracey et al. 2002; Bunte et al. 2011; Buret et al. 2014; Bate and Khan 2011), it is assumed that there exists a means for evaluating the fitness value of all the individuals in a population during the GA-evolution computations. Generally, the fitness value of an individual is computed using an analytical fitness function, a computational simulation, or an experiment (using hardware). In practice, however, computing fitness values for multiple individuals during the GA-evolution process may become non-trivial. Particularly, such non-trivial situations occur when either the computational simulations for each fitness estimation are highly time-consuming, or the experiments for fitness evaluation are prohibitively costly, or an explicit analytical function for fitness computations does not exist. It provides the motivation to assist the GA-evolution process with prediction models. In other words, prediction models will prune the search space of measurement-based analysis by predicting fitness to generate a set of test data for which the execution time is near to the worst-case.

2.2.3 Prediction models in GA-based solutions

The shortcomings of existing GA-based temporal testing techniques, mentioned in Sect. 2.2.2, sets the stage for the integration of prediction models. The objective of model integration in a GA-based solution is to predict the worst-case input data. The employment of a prediction model reduces the need for required simulation runs and thus the overall temporal testing time is decreased. Prediction models are being used in the literature to facilitate the optimization of expensive computer simulations, by approximating the fitness function(s) for at least two decades (Haftka et al. 2016). The use of prediction models to assist GAs in various industries to solve diverse nature of problems is getting popular (Koopialipoor et al. 2019; Moayedi et al. 2019; Armaghani et al. 2018; Rodriguez-Roman 2018). For example, in Koopialipoor et al.

(2019), overbreaking in tunnel construction induced by drilling and blasting operation is approximated using a hybrid neuro-genetic (GA-ANN) predictive model. Moayedi et al. (2019) use multiple evolutionary and neural network models, including genetic algorithm optimized with ANN (GA-ANN), to predict ultimate bearing capacity of shallow footing on soil. In Armaghani et al. (2018), airblast prediction is done through a hybrid genetic algorithm with ANN model. Similarly, the work in Rodriguez-Roman (2018) uses surrogate model assisted genetic algorithms in project designs for highway safety and travel time improvement. The use of prediction models to assist GAs in multiple areas have shown promising results.

2.3 Novelty of the proposed approach

The execution of benchmarks is a major bottleneck in time-complexity reduction of state-of-the-art GA-based temporal testing techniques (Law and Bate 2016; Wegener et al. 1996; Tracey et al. 2002; Bunte et al. 2011; Buret et al. 2014; Bate and Khan 2011; Aziz and Shah 2015) for real-time embedded systems. Our study eliminates this bottleneck by forecasting the execution time of benchmarks using prediction models, hence saving the execution time during GA-evolution. In other words, a computationally expensive simulation model is replaced with a cheaper-to-run prediction model. We have achieved 10 to 21 times efficiency, as presented in Table 6. Moreover, with the higher execution time complexities of benchmarks and with more number of GA generations, the significance of this efficiency gets more pronounced because the time complexity of our approach only scales up linearly.

3 Materials and methods

This section presents the target prediction models, the methodology used for their evaluation and the benchmarks used for their validation. The block diagram of employed methodology in Fig. 1 shows that it consists of two main components: a genetic algorithm (GA) and a prediction model.

A GA is an evolutionary search algorithm where the initial population of random data is evolved, using certain parameters and fitness values, to find the optimal solution of an optimization problem (Surendran and Samuel 2016). The fitness values used in this methodology are execution times. The working phenomenon of GAs is briefly described in Sect. 3.1. On the other hand, the prediction model is first trained with the training data. The trained model then takes the input data from the GA to provide the corresponding fitness values through predictions. Sections 3.2 and 3.3 provide the brief description of prediction models and the training mechanism respectively. Finally, Sect. 3.4 briefly describes the benchmarks, used for the evaluation of various prediction models.

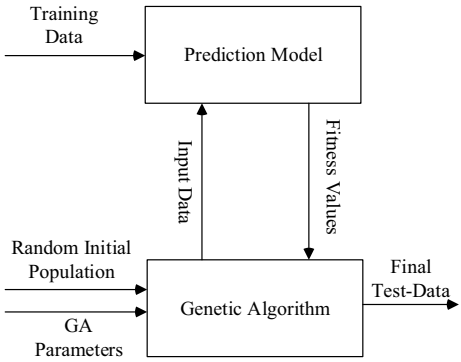


Fig. 1 Block diagram of employed methodology

3.1 Genetic algorithm

Figure 2 shows a flow chart representation of the GA. It can be sub-divided into two parts: initialization phase and evolution phase. The initialization phase starts with the generation of initial population and the corresponding fitness values. The initial population, consisting of individual solutions to the given optimization problem, is generated randomly. The individual solutions in the initial population represent the input test data to a real-time application. The fitness values (execution times) of these individual solutions (input test data) are required to be determined. The employed methodology uses a trained prediction model for the determination of fitness values, instead of the actual execution of the target application on a simulator or real hardware. The trained prediction model provides an estimate of the execution time (fitness value) through predictions. Consequently, the predicted fitness values of a randomly generated initial population are used to determine if more generations of the GA are required or the stopping criteria is met. If the stopping criteria is not satisfied, the GA enters into the evolution phase. Finally, the evolution phase is repeated over generations until the stopping criteria is satisfied.

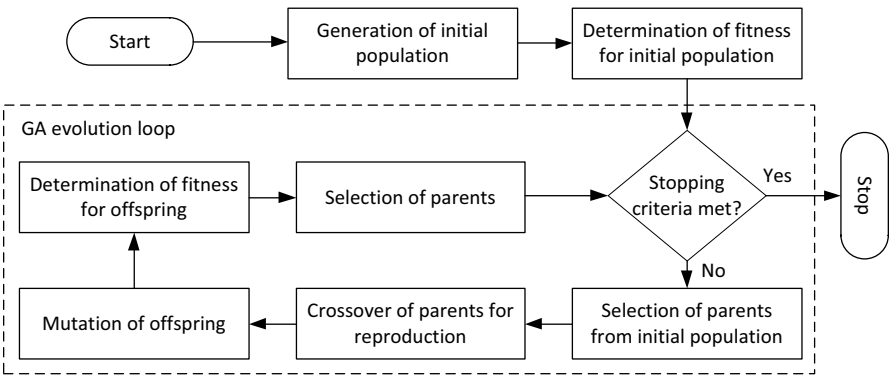


Fig. 2 Flow chart representation of GA

Evolution phase, consisting of a GA loop, starts with the selection of parents from initial population. There are various selection strategies. Typical examples of selection strategies are rank-based and roulette-based (Sastry et al. 2014). Once the parents are selected, they produce offsprings through crossover and the mutation is applied on the offsprings (Srinivas and Patnaik 1994). Thereafter, the fitness evaluation of offsprings is performed through a prediction model. Finally, a selection strategy is applied on offsprings and parents to form a new population.

3.2 Prediction models

A prediction model is used to obtain an estimated fitness value of a given input. It is first trained and then used with the GA as shown in Fig. 1. Consequently, the use of prediction model eliminates the need of a simulator or actual hardware for the fitness evaluation. In this article, we have targeted four prediction models. Two of them, generalized linear regression model and gaussian regression model (probabilistic model) belong to the statistical modeling class (Fahrmeir and Tutz 2013; Rasmussen and Williams 2006) and two models are borrowed from machine learning literature (Hagan et al. 2014; Vapnik 2000). Artificial neural network and support vector regression model are widely addressed in the literature for modeling of the data and successfully used in many real-life applications already (Hagan et al. 2014; Smola and Scholkopf 2004). Prediction of execution times for a problem requires proper selection of the model and its architecture to minimize the prediction error. Furthermore, prediction model is used to guide the GA-search towards the worst-case test data. Even in the presence of prediction error, if the model can successfully guide the GA towards the worst cases, the model is very useful. As we reach close to the worst-case test data, we can always find the actual worst-case execution time from the real system or simulator. The in-depth knowledge of these prediction models is not required to understand the experiments presented here. However, for the sake of completeness, a brief description of each prediction model is given in the following.

3.2.1 Artificial neural network

An artificial neural network (ANN) is a collection of artificial neurons connected together. There can be multiple layers of neurons in an ANN where the first layer is the input layer, the last layer is the output layer and the intermediate layers are known as the hidden layers (Hagan et al. 2014).

The number of inputs and outputs depends on the design of ANN architecture. An ANN architecture can be multiclass (having multiple outputs) or single-class (having only one output). Figure 3 presents a typical single-class ANN architecture with m inputs. In our study, we use a single-class ANN architecture because there is only one desired output, the WCET.

Inside the layered organization of neurons, connected neurons communicate by sending and receiving the signals. The applied inputs are transferred to the hidden layer(s) for the actual processing. Each neuron has an associated weight that

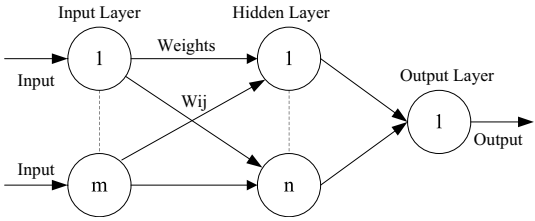


Fig. 3 An artificial neural network with a single hidden layer

influences the value passing through this neuron. The output layer takes the processed values from hidden layer before the final output is produced. Let $X \in \mathbb{R}^m$ is an input to the artificial neural network where $X = [x_1, x_2, \ldots, x_m]$. The output of ANN can calculated as follows:

$$Output = f\left(\sum_{j=1}^{n} v_j g_j\left(\sum_{i=1}^{m} w_{ij} x_i\right)\right) \tag{1}$$

Equation (1) is the analytic form of the neural network in Fig. 3, where v_j and w_{ij} are the weights that are adjusted for optimizations according to appropriate learning rules. There are variety of learning mechanism, available in the literature, for the weights associated with the neurons in every layer (Hagan and Menhaj 1994). Similarly, f and g_j are the activation functions. Based on the requirement of the prediction problem, different types of activation functions can be used. A good list of learning mechanisms and activation functions can be found in Hagan et al. (2014).

3.2.2 Generalized linear regression model

A generalized linear regression model (GLM) is a linear regression model that is flexible and generalized. GLMs were developed by John Nelder and Robert Wedderburn Fahrmeir and Tutz (2013) to handle the data distribution which does not follow the normal distribution.

Given a vector of predictors X and an outcome Y, the conditional expectation is $\mathbb{E}(Y|X)$; where $X \in \mathbb{R}^m$, $Y \in \mathbb{R}$, and $\mathbb{R}^m$ is m-dimensional real space for any positive integer m, $\mathbb{R}$ is the set of real numbers and $\mathbb{E}$ is the expected value. The GLM comprises of three components:

1. *a random component,* specifying the probability density function of the outcome Y (Lower case y is used to differentiate the random variable Y from its realization y):

$$f(y;\mu,\sigma) = exp\left\{\frac{y\mu - b(\mu)}{a(\sigma)} + c(y,\sigma)\right\} \tag{2}$$

   where f is the probability density function, a, b and c are the known functions, μ is the mean of the outcome Y and is also known as natural parameter. σ is the standard deviation of the outcome Y and is also known as dispersion parameter.
2. *a systematic component,* relating the predictors X to a parameter η:

$$\eta = \sum_{j=1}^{m} \beta_j X_j \tag{3}$$

where X_j is the j^{th} predictor, η is the linear combination of predictors, i.e., $\eta = \beta^T X$; such that, $\beta = [\beta_1, \beta_2, ..., \beta_m]$, where β_j is a j^{th} linear coefficient and β is a vector of linear coefficients. T is the non-linear transform of β from an m-dimensional input pattern space ($\mathbb{R}^m$) to a p-dimensional numerical feature space ($\mathbb{R}^p$).

3. *a link function,* connecting *random* and *systematic* components:

$$g(\mathbb{E}(Y|X)) = \eta \tag{4}$$

where g is a function that links (not equate) the probability density function of the outcome Y given in Eq. (2) to the linear combination of predictors X given in Eq. (3) by equating the $\mathbb{E}(Y|X)$ to η. Where $\mathbb{E}(Y|X)$ is the expected value of the outcome Y when the predictor X is given/known.

Equation (4) can be rewritten as:

$$\mathbb{E}(Y|X) = g^{-1}(\eta) \tag{5}$$

For given samples (x_i, y_i), where i=1,...,N, for each $y_i|x_i$ Eq. (5) becomes:

$$\mathbb{E}\left(y_i\right) = g^{-1}\left(\eta_i\right) \tag{6}$$

and

$$g(\mathbb{E}) = \sum_{j=1}^{m} \beta_j x_j \tag{7}$$

Hence, GLM is nothing but a linear model with transformed mean of an independent variable that has distribution from the exponential family (Fahrmeir and Tutz 2013).

3.2.3 Gaussian process regression model

Suppose we have $X_i \in \mathbb{R}^m$ as the i^{th} input patterns and y_i is the output. A linear regression model (Rasmussen and Williams 2006) is defined in Eq. (8), as follows:

$$y_i = X_i^T \beta + \epsilon \quad \text{where} \quad \epsilon \sim N\left(0, \sigma^2\right) \tag{8}$$

In Eq. (8), T represents non-linear input-to-feature transformation, that projects the m-dimensional input pattern space ($\mathbb{R}^m$) to p-dimensional numerical feature space ($\mathbb{R}^p$), β is the linear coefficient, ϵ is the error, tilde ($\sim$) means "has the distribution", i.e., ϵ has a normal distribution of zero mean and σ^2 variance. The error variance σ^2 and linear coefficients β are estimated from the data. A Gaussian process regression (GPR) model introduces a latent variable $f(X_i)$ from a Gaussian process and a set of basis functions $h(x)$. These basis functions project the input patterns coming from $\mathbb{R}^m$ to p-dimensional feature space ($\mathbb{R}^p$). For n input patterns $X_1, X_2, \ldots, X_n$, the

joint distribution of n random variables $f(X_1), f(X_2), \ldots, f(X_n)$ is Guassian. The GPR model is of the form of Eq. (9), as shown in the following:

$$h(X)^T\beta + f(X) \quad \text{where} \quad f(X) \sim GP\left(0, k\left(X, X^{'}|\theta\right)\right) \tag{9}$$

The latent variable $f(x)$ is from a Gaussian process with zero mean and covariance function $k\left(X, X^{'}|\theta\right)$. GPR model is a probabilistic model and output y_i is modelled, as given in Eq. (10).

$$P\left(y_i|f(X_i), X_i\right) \sim N\left(y_i|h\left(X_i\right)^T\beta + f\left(X_i\right), \sigma^2\right) \tag{10}$$

GPR model estimates the linear basis coefficients β , the noise variance σ^2 and the hyper-parameters θ from the training data whereas basis functions and the covariance function are predefined.

3.2.4 Support vector regression model

Suppose we have $X_i \in \mathbb{R}^m$ as the i^{th} input patterns and y_i is the corresponding output. Let a linear function takes the form of Eq. (11), as given in the following:

$$f\left(X_i\right) = X_i^T\beta + b \quad \beta \in \mathbb{R}^m, b \in \mathbb{R} \tag{11}$$

Where β is the linear coefficients and b is the bias term. The objective function $J(\beta)$ (Vapnik 2000; Smola and Scholkopf 2004) including slack variables ξ and ξ^* to handle the feasibility of the solution is defined in Eq. (12), as shown in the following:

$$J(\beta) = \frac{1}{2}\beta^{'}\beta + C\sum_{1}^{N}\left(\xi_n + \xi_n^*\right) \tag{12}$$

Subject to,

$$\begin{aligned}
&\forall n \ : \ y_n - \left(x_n^{'}\beta + b\right) \leq \varepsilon + \xi_n \\
&\forall n \ : \ \left(x_n^{'}\beta + b\right) - y_n \leq \varepsilon + \xi_n^* \\
&\forall n \ : \ \xi_n^* \geq 0 \\
&\forall n \ : \ \xi_n \geq 0
\end{aligned}$$

The slack variables ξ and ξ^* are used to measure the deviation of the training patterns outside ϵ-insensitive zone. A positive numeric constant **C** in the objective function is to penalize the observations outside the margin and is useful in preventing the overfitting. The formulation is a convex optimization problem to minimize the objective function $J(\beta)$ (Vapnik 2000). This problem can be solved in many ways. One of the famous solver is Sequential minimal optimization (SMO) method (Smola and Scholkopf 2004) which performs a series of two-point optimizations. Further details about the model can be found in Deng et al. (2012).

3.3 Training of prediction models

In this methodology, a model is trained before its deployment. Figure 4 shows the mechanism used for the training of prediction models. The training methodology involves passing of random input data to the executing benchmark on the simulator/hardware as well as to the prediction model, to produce the actual and predicted execution times, respectively. These execution times are compared and the error is determined to fine tune the model until the final input from the pool of random inputs is used. Finally, the trained predictor is used in the evolution process of the GA.

In this work, training of the prediction model was performed through the Gem5 simulator (Binkert et al. 2011). However, the proposed approach is equally applicable to any other simulator or hardware. The Gem5 simulator has been used due to its wide adoption in the embedded systems research community while its accuracy evaluation is provided in Butko et al. (2012). It is an open source cycle-accurate architecture simulator which provides a variety of simulation options with different memory systems and CPU choices such as x86, Alpha, Sparc and ARM.

3.4 Benchmarks

In order to analyze different prediction models, we selected two benchmarks from Mälardarlen WCET suite (Gustafsson et al. 2010). These two benchmarks are *Bubble sort* and *Insertion sort*. In addition to *Bubble sort* and *Insertion sort* algorithms, two additional sorting algorithms *Gnome sort* and *Shaker sort* were selected (Dharmajee Rao and Ramesh 2012). The selection of these benchmarks is due to the fact that they can best represent the change in their execution times, influenced by the input data. Inputs to all the benchmarks (sorting algorithms) in this study are fixed-sized unsorted arrays of random integers between 0 and 1000. The algorithms used in these benchmarks sort those arrays. The number of swaps required for sorting the list depends on the algorithm used. However, the number of required swaps also depends on the initial order of the list to be sorted. Consequently, the execution time of a benchmark, which is directly related to the number of swaps, is heavily influenced by the input data. The worst-case performance for these sorting algorithms is $O(n^2)$ swaps, where n is the size of the given list of integers. The worst-case input in these benchmarks is a reverse sorted array of integers.

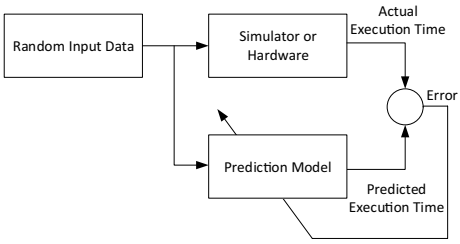


Fig. 4 Training of prediction models

4 Experimental setup

This section elaborates the experimental setup for the evaluation of prediction models, described in Sect. 3.2, with the benchmarks presented in Sect. 3.4. It first describes the particular settings for target architecture and prediction models in Sect. 4.1. Then, the required parameters for models training as well as benchmark algorithms are presented in Sect. 4.2. The parameters and the stopping criteria for GA-evolution are described in Sects. 4.3 and 4.4, respectively. While keeping the main focus on the use of various prediction models, experimentation in this work has been performed with single tasked (sole function) benchmarks. However, the same setup can be applied for multi-functions, multi-threading, and multi-tasking applications.

4.1 Target architecture and prediction models

The ARM v7-A instruction set architecture-based micro-architectural model is used as target platform, available in Gem5 simulator. The selected architecture model is a single core processor, clocked at a frequency of 2 GHz with 512 MB physical memory. Furthermore, the memory model used is simple classic memory and the mode of simulator used is system call emulation mode. All this experimentation, including the running of simulator, was performed on a Dell precision workstation having Xeon processor with twelve logical cores and 48 GB of physical memory.

The prediction models employed in this work are based on ANN, GLM, GPR and SVR. All the prediction models are realized in the Matlab. The Matlab version 9.1.0.441655 (R2016b) is used for the application of these prediction models. The implementation of ANN is provided in Neural Network Toolbox, whereas the implementations of GLM, GPR and SVR are available in Statistics and Machine Learning Toolbox in the Matlab. The architecture of ANN is 16-10-1 having 16 input neurons, 10 hidden neurons in a single hidden layer and one output neuron.

Two key parameters impact the configuration of ANN-architecture: (i) number of neurons in the hidden layer, (ii) choice of the activation function (AF). Both these parameters are explained as follows: Test experimentation with varying number of neurons in the hidden layer can be conducted to decide the most suitable number for the ANN-architecture. In this study, the experimentation is conducted with 6, 8, 10, 12, 16 and 20 neurons in the hidden layer. Best accuracy for most of the benchmarks is achieved with 10 neurons. Hence, the number of neurons in the hidden layer is set to 10. The second consideration is the choice of AF. Literature analysis suggests four closest choices: Sigmoid, Hyperbolic Tangent, Rectified Linear Unit and Softmax (Njikam and Zhao 2016; Nwankpa et al. 2018). Sigmoid is a popular choice in shallow networks because it is easy to apply and compute (Nwankpa et al. 2018). Although it can be argued that the other three AFs can be better choices than Sigmoid because of sharp deep gradients during back propagation from deep hidden layers to input layers, slow convergence and non-zero centered output (Glorot and Bengio 2010); but, this study uses neither deep hidden layers, nor a zero-centered

output is required, nor slow convergence is relevant in the shallow ANN-architecture. Hence, Sigmoid is selected as AF.

Levenberg-Marquardt training method is used for the learning of weights and bias (Hagan and Menhaj 1994). Maximum number of epochs for training is set to 1000 but learning will stop if the gradient becomes smaller than a pre-defined value. In GLM, normal distribution is used and the link function is identity function (Fahrmeir and Tutz 2013). In GPR model, squared exponential function is used as the kernel function, basis function is constant and quasi-Newton optimizer is used (Rasmussen and Williams 2006). Sequential minimal optimization (SMO) is used for optimizing the regression parameters in the SVR prediction model with linear kernel function (Vapnik 2000; Smola and Scholkopf 2004).

4.2 Parameters for models training and benchmark algorithms

For each of the four benchmarks (described in Sect. 3.4), four best-trained prediction models (described in Sect. 3.2) are used. Hence, in total, 16 best-trained prediction models are achieved. Each best-trained prediction model is used in a GA-evolution for a fixed size of array. Thus, 16 GA-evolutions are performed in an experiment. This entire experiment is repeated for array sizes of 16, 24, 32 and 40.

In order to evaluate the performance of various prediction models, Mean Absolute Percentage Error (MAPE) was used. It is a measure of prediction accuracy of a model. MAPE is calculated as:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{ET^{true}(i) - ET^{pred}(i)}{ET^{true}(i)}\right| \times 100 \tag{13}$$

In Eq. (13), $ET^{true}(i)$ is the true execution time when the sorting algorithm is applied on the i^{th} data on the simulator, $ET^{pred}(i)$ is the execution time predicted by the prediction model for i^{th} data point and N is the total number of data points. A low value of MAPE means small error and good quality of prediction model.

The training of every {Prediction model, Benchmark} set is performed five times: each time on a distinct instance of a prediction model, using 1000 randomly generated data points (each data point is a fixed-sized unsorted array of random integers between 0 and 1000). The purpose of running a training multiple times is to identify the best-trained instance of the prediction model, which will later be used in GA for fitness evaluation. A MAPE value is calculated on the results generated from testing of each of the five instances of a prediction model on the training data. The best-trained model is the one having the minimum MAPE value. The minimum MAPE values on the training data for each of the 16 {Prediction model, Benchmark} sets is shown in Table 2.

Similarly, the testing of every {Prediction model, Benchmark} set is performed five times: each time on a distinct instance of a prediction model, using 2000 randomly generated data points. As in the training, the prediction model instance having the minimum MAPE value from the 5 testing (of models) experiments is selected as the best-trained instance for later use in GA for fitness evaluation. The minimum

MAPE values on the testing data for each of the 16 {Prediction model, Benchmark} sets is shown in Table 3. Once the training is complete, the GA is evolved for each prediction model.

The worst-case input data is the reverse sorted list of integers that causes the maximum possible end-to-end execution time. Worst-case execution times for Bubble sort, Insertion sort, Gnome sort and Shaker sort are 616134, 310192, 813812, and 1234331 nanoseconds, respectively, when run on the Gem5 simulator.

4.3 Parameters for GA-evolution

The GA-parameters are kept constant throughout the experiments. Initial input to the GA is unsorted, fixed-sized arrays of randomly generated integers between 0 and 1000 which forms initial population of random input data-points. For each generation of the GA, population size is kept constant at 50. Table 1 presents a fixed set of GA-parameters, used in all of our experiments. The convergence of a GA toward the required outcome is highly dependent on the probabilities used in crossover and mutation (Sastry et al. 2014; Srinivas and Patnaik 1994). The new solutions (offsprings), produced through crossover, occur in the vicinity of old solutions (parents). The higher crossover probability means decreased exploitation and increased exploration while the lower value of this probability may results in an early convergence which is inadequate. Standard values of crossover probabilities are between 0.6 and 1.0. Mutation is just like flavoring and hence the mutation probability is relatively small as compared to the probability value used in crossover. Mutation of newly generated population through crossover is needed to search the un-revealed areas of the search space. However, a large value of mutation can change the GA-search into a pure random search. The value of mutation probability is typically considered between 0.005 and 0.05. In this work, GA-parameters are evaluated based on the methodology of Pongcharoen et al. (2002) for the determination of optimum GA-parameters. They consider the most efficient Genetic Algorithm parameters which achieve minimum total cost and minimum spread. The procedure considers various levels of the GA-parameters, population size, number of generations and the probability of crossover and mutation. The values which we determined using this methodology are presented in Table 1.

Table 1 GA-parameters and their values used during GA-evolution

GA-parameters	Methods/values
Population size	50
Parent selection	Roulette wheel selection
Crossover	Arithmetic crossover
Crossover srobability	0.8
Mutation	Single point random mutation
Mutation srobability	0.05
Population selection	Elitism, 2 best individuals
Maximum number of generations	1000

4.4 Stopping criteria for the GA

In GAs, as the generation progresses, the maximum and/or average fitness of a population increases. Fitness of a member of population or a data point corresponds to the quality of the data point according to a user defined measure which is the execution time of a sorting algorithm. Once it reaches near to a solution or trapped in some local optima, improvement in the fitness values becomes minimal with the generation. Therefore, it is not useful to run the GA for further generations and it is better to stop the GA. Different types of stopping criteria are defined in the literature (Jain et al. 2001; Bhandari et al. 2012). Therefore, we defined a stopping criteria to stop early if there is no or minimal improvement in the fitness values of the population.

We defined the following stopping criteria: (1) maximum number of generations have been reached and (2) the maximum percentage change in predicted fitness values, over a certain number of generations, is less than a user-defined constant (saturation test). In our experiments, the maximum number of generations in GA-evolution process is set to 1000 as shown in Table 1. For the maximum percentage change, the GA process is first allowed to evolve uninterruptedly (no stopping criteria) for at least 250 generations to avoid any immature halt. Since the 251st generation, the criteria of maximum percentage change is applied. The criteria of maximum percentage change is based on the following mathematical formulation:

If we represent the generation with G_i, the input data in each generation with d_i and the associated predicted fitness value of the input data with f_i, then:

$$G = \{(d_1,f_1),(d_2,f_2),(d_3,f_3),\ldots,(d_n,f_n)\} \tag{14}$$

Where n is the number of data points in each generation, set at 50 in this study. Its value can be varied depending on the experiment design. A data point is a pair of an input data and the corresponding execution time. The maximum fitness value from each generation during the evolution of GA is stored in a FIFO (First In First Out) buffer. If the FIFO buffer, filled with maximum fitness values from each generation, is represented by FP:

$$FP = \{f_{max}(G_1),f_{max}(G_2),\ldots,f_{max}(G_s)\} \tag{15}$$

Where s is the size of FIFO buffer set at 50 in this study. Its value can be varied depending on the experiment design. Therefore, the maximum percentage change in the predicted fitness values over s generations (MPC_{FP}) can be determined and compared with the user-defined constant, as follows:

$$MPC_{FP} = \frac{Max(FP) - Min(FP)}{Max(FP)} \times 100 < \gamma \tag{16}$$

Where the user defined constant, represented by γ, sets the bottom line for the percentage change in the fitness value required to determine if the saturation is reached and hence the evolution of the GA needs to be stopped. The value of γ in Eq. (16) is taken as $1.0e-05$. However, if the formulation 16 is not satisfied, the GA continues its evolution with another generation. To summarize, we defined the following values in our experimentation:

- Total number of generations = 1000
- Number of data points in each generation = 50
- Size of the FIFO buffer = 50
- The value of $\gamma = 1.0e - 05$

4.5 Performance measures

Once the prediction models are trained, GA-evolutions are run for each combination of benchmark and prediction model. As described in Sect. 4.2, one such experiment includes 16 GA-evolutions for a fixed input size. The experiment for benchmarks with input array size 16 is evaluated using five performance measures: prediction performance, isolated timing performance, evolution performance, integrated timing performance and the quality of test data. The details of each performance measure are as following:

- Prediction performance of a model describes its accuracy. It represents the closeness of the predicted value (obtained from the model) to the actual value (obtained from the hardware or simulator).
- Isolated timing performance of a model describes the time required for its training as well as prediction. It demonstrates the reduction in required time with prediction models as compared to the actual execution on the simulator.
- Evolution performance of a model describes its ability to let the GA converge in minimum possible number of generations.
- Integrated timing performance of a model demonstrates its ability to reduce the overall time required to reach the final solution.
- The data quality performance of a model shows the deviation of achieved data from the actual worst-case data.

Moreover, to analyze the impact of various complexities (different array sizes) of the problem, subsequent experiments (each experiment consisting of 16 GA-evolutions for 16 {Prediction model, Benchmark} sets) are run for input array sizes 24, 32 and 40. For these array sizes, the performance of each prediction model is evaluated in terms of time saved by using the proposed framework.

5 Results and discussion

This section presents the experimental results with the target predication models and the selected benchmarks, described in Sects. 3.2 and 3.4, respectively. Sections 5.1 to 5.5 present results for benchmarks with input array size 16 using five performance measures, defined in Sect. 4.5. Section 5.6 analyzes the impact of using array sizes 24, 32 and 40.

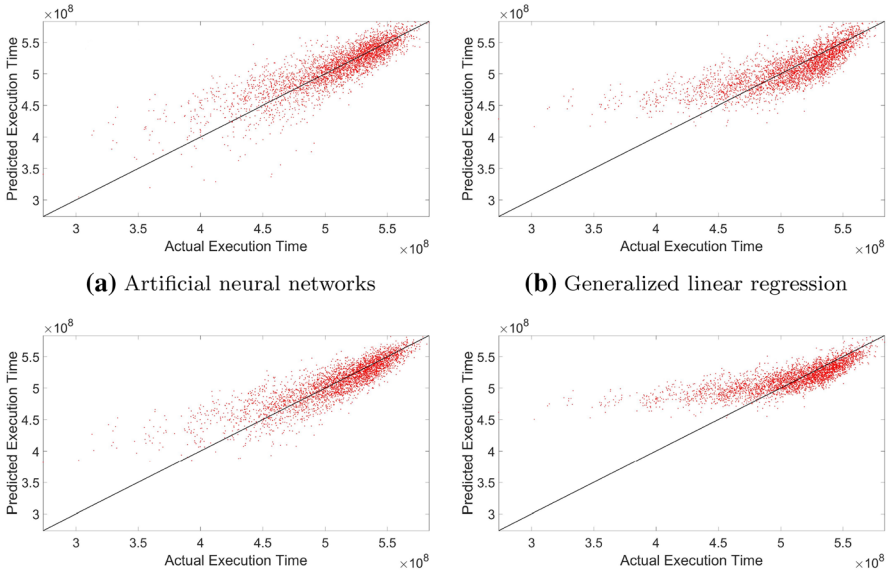
5.1 Prediction performance

This performance measure (prediction performance) evaluates the accuracy of a trained prediction model before its integration with the GA. The predicted values from a trained prediction model are compared with the actual fitness values (computed from simulator) and the results are shown in the form of scatter plots. Figures 5, 6, 7 and 8 provide the comparison between actual execution time and predicted execution time for Bubble sort, Insertion sort, Gnome sort and Shaker sort, respectively.

In each figure, the x-axis represents the actual execution time obtained from the simulator against a given input, whereas the y-axis shows the execution time predicted by the model against the same input. Furthermore, each figure (Figs. 5, 6, 7, 8) consists of four graphs, showing the results for ANN, GLM, GPR and SVR prediction models.

Figure 5 shows a worse prediction of the real execution times for Bubble sort, than the other sorting algorithms in Figs. 6 and 7. After GA-evolution using the same prediction models, Table 7 shows the best results for Bubble sort, except for GPR. The reason for this could be the better accuracies of the prediction models for bubble sort at higher execution times in Fig. 5 as compared to the uniform prediction accuracies depicted in Figs. 6 and 7.

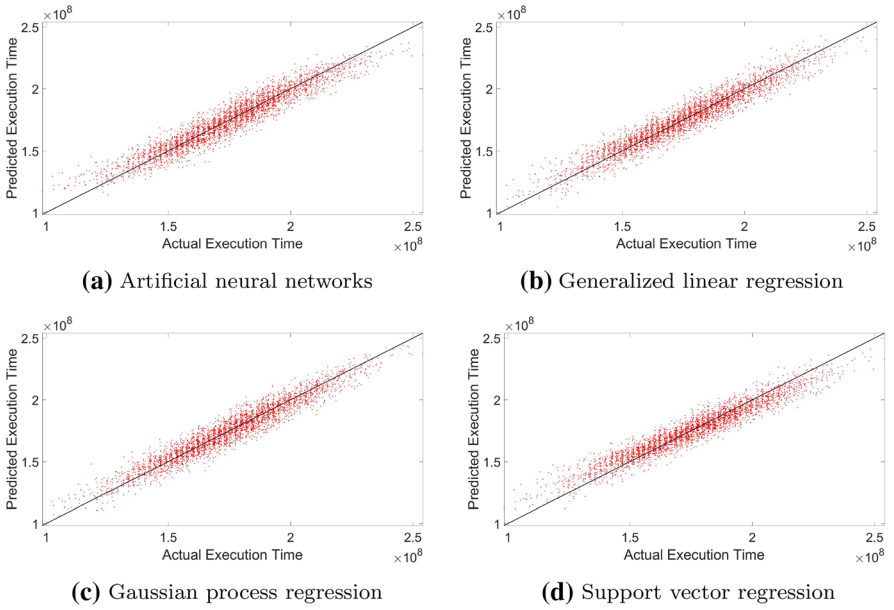
Tables 2 and 3 present MAPE values (defined in Sect. 4.2) from training and testing data, respectively, using best-trained prediction models. Lower MAPE values on both training and testing data indicate better quality of prediction models.



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression

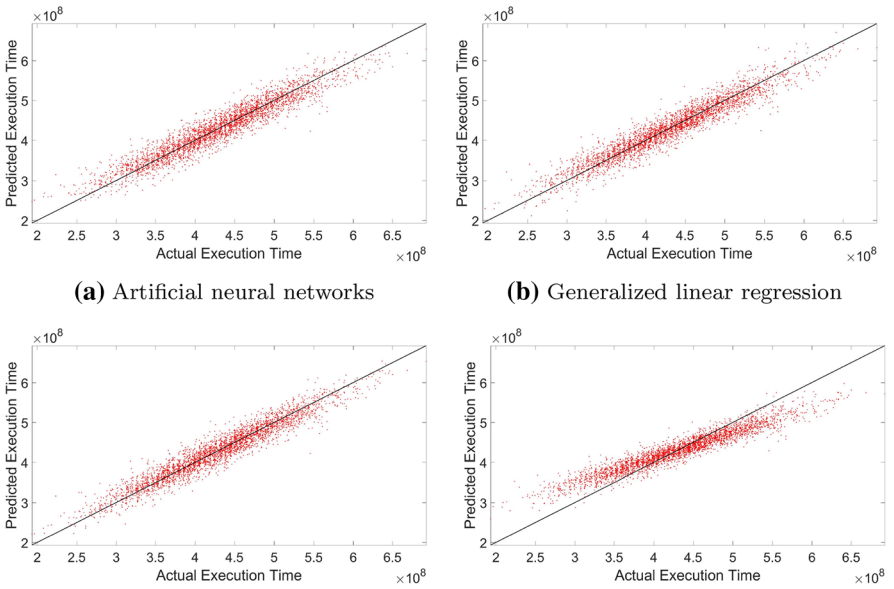
Fig. 5 A scatter plot of measured vs predicted execution times for bubble sort using different prediction models



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression

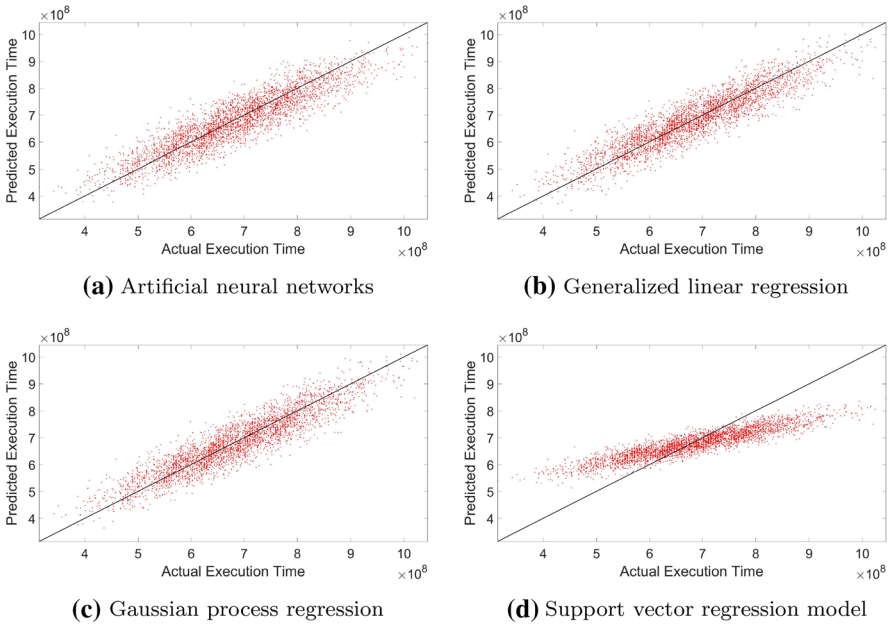
Fig. 6 A scatter plot of measured vs predicted execution times for Insertion sort using different prediction models



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression

Fig. 7 A scatter plot of measured vs predicted execution times for Gnome sort using different prediction models



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression model

Fig. 8 A scatter plot of measured vs predicted execution times for Shaker sort using different prediction models

Comparison of Tables 2 and 3 reveals that the MAPE values obtained on the training set are comparable with those obtained on the test data. Hence, the models are not suffering from overfittings and good generalizations are achieved. It means that the prediction models can be used interchangeably instead of actual execution of the benchmarks.

From Table 3, the MAPE values on testing data indicate that for all the sorting algorithms the GPR model gives the best quality. The SVR model has shown the worst quality except for Insertion sort. The MAPE value of SVR for Shaker sort is the highest among all the MAPE values. Similarly, GPR gives the best, and SVR, except for Insertion sort, gives the worst quality on the training data in Table 3. The exception of Insertion sort in the worst quality of SVR remains the same in Tables 2 and 3.

Table 2 MAPE results on training data using best-trained prediction models

Models	Benchmarks (sorting slgorithms)			
	Bubble	Insertion	Gnome	Shaker
ANN	3.47	2.71	2.96	4.53
GLM	3.28	2.23	2.87	4.06
GPR	1.39	1.69	2.08	3.86
SVR	3.68	2.59	4.38	7.06

Table 3 MAPE results on testing data using best-trained prediction models

Models	Benchmarks (sorting algorithms)			
	Bubble	Insertion	Gnome	Shaker
ANN	3.75	3.14	3.85	5.09
GLM	3.57	2.30	2.91	4.23
GPR	2.27	2.36	2.66	4.11
SVR	4.06	2.65	4.41	7.17

From these findings, although GPR seems to be the best choice of a prediction model, later results suggest that it is hard to guarantee a single choice of prediction model that retains the best-quality for the entire set of data points or benchmarks. For example, (i) unlike in Table 3, GPR is not the best-quality prediction model in Table 7, Bubble sort being the exception. (ii) Unlike in Table 3, SVR in Table 7 is the worst-quality prediction model only for Gnome and Shaker sort, but not for Insertion or Bubble sort. This inconsistent quality of prediction models may be attributed to multiple possible reasons. Firstly, the prediction models trained on random data may not be accurately predicting during the GA-evolution, as the population starts getting closer to the worst-case data. Secondly, the intrinsic randomness in GA-evolution can be a factor that changes the accuracy patterns of the trained prediction models. These arguments need further investigation.

The GA must use the best-quality prediction model for fitness evaluation. Finding the best-quality prediction model may involve training multiple prediction models, which is the approach followed in this study. The prediction model with minimum MAPE value can be selected as the best-quality prediction model for use in the GA.

5.2 Isolated timing performance

In addition to the prediction performance of a trained prediction model, the evaluation of its timing performance is equally important. The isolated timing performance evaluates: (1) the time consumed by a prediction model for its training and (2) the time taken by a trained prediction model for predicting the fitness values. Therefore, training times as well as prediction time for each model should be measured for the selected benchmarks. Table 4 shows training as well as prediction times for all the target prediction models.

Column 1 of Table 4 lists the selected benchmarks while columns 2 and 3 represent the training and prediction times for various prediction models, respectively. The results in this table reveal that the prediction models require almost negligible amount of time for training as well as prediction. Moreover, the ANN prediction model takes comparatively larger amount of training and prediction times as compared to other prediction models. Similarly, the Bubble sort algorithm consumes more time as compare to other algorithms, as shown in Table 4. Furthermore, it can be observed that the training times are large as compared to the prediction times. However, the models are trained only once during the entire experiment.

Table 4 Training and prediction times in seconds

Benchmarks	Training times (s)				Prediction times (s)			
	ANN	GLM	GPR	SVR	ANN	GLM	GPR	SVR
Bubble sort	6.2639	4.5841	3.4283	1.9239	0.2852	0.0125	0.0806	0.1076
Insertion sort	0.6347	0.2742	3.2103	0.0684	0.0469	0.0086	0.0759	0.0077
Gnome sort	0.4883	0.1395	4.4501	0.0950	0.0456	0.0023	0.0753	0.0186
Shaker sort	0.2912	0.0903	1.8007	0.0607	0.0221	0.0022	0.0625	0.0189

5.3 Evolution performance

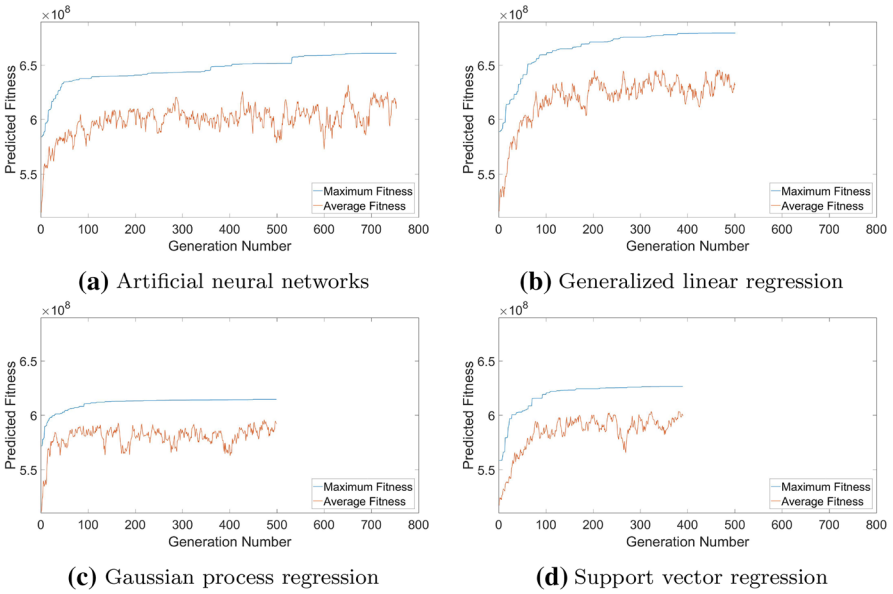
The previous two performance measures (i.e., prediction performance and isolated timing performance) evaluate the performance of a trained prediction models without any interaction with the GA-evolution process. However, the trained prediction models are required to be integrated with the GA and their integrated performance needs to be measured. Therefore, "evolution performance" is a measure of a predication model's integrated performance during the GA-evolution.

The prediction model predicts the fitness values during multiple generations of GA-evolution process. In other words, the prediction accuracy of the prediction model directly influences the number of the GA generations. Generally speaking, lesser the number of generations required in a GA-evolution process, better is the prediction model and vice versa.

The GA-evolution performance for Bubble sort, Insertion sort, Gnome sort and Shaker sort are shown in Figs. 9, 10, 11 and 12, respectively. Furthermore, each figure (Figs. 9, 10, 11, 12) consists of four graphs, showing the results for target prediction models. In each graph, the x-axis represents the number of generations while the y-axis provides the predicted fitness values during various GA generations. For the comparison of different approaches, the horizontal axis has been standardized for all the plots with 0 to 800 generations and the vertical axis have been standardized in quadruplets (for every benchmark) with minimum and maximum predicted fitness values. Two curves are used in each graph to represent the maximum and the average predicted fitness values during the GA-evolution process.

A comparison of quadruplets reveals variations in the maximum fitness values of the same benchmark for different prediction models. For example, GLM shows highest maximum predicted fitness value. Since these values are higher than the WCET, it clearly shows that this model has an added bias in its predicted value. Similarly, with negative biases in prediction values GPR, ANN, and SVR have the lowest maximum fitness values for Bubble, Insertion and Gnome, and Shaker sort, respectively.

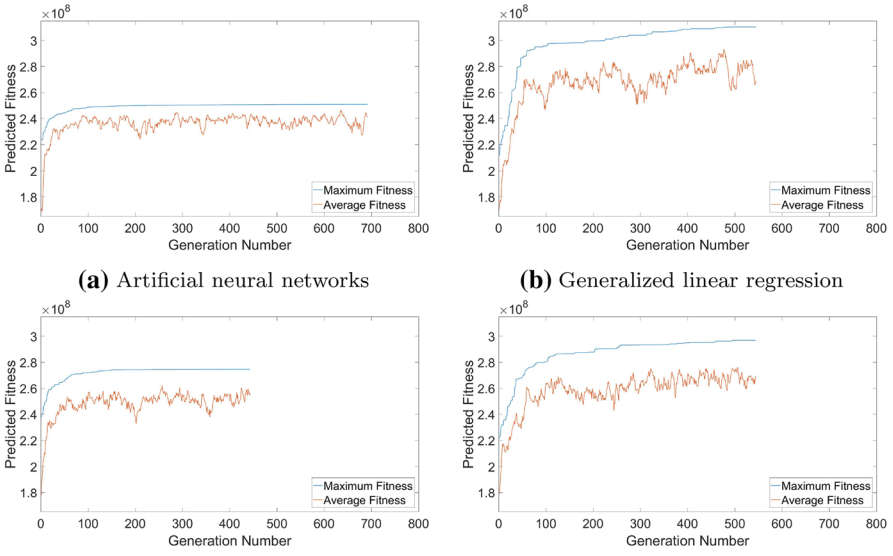
It can be observed from Figs. 9, 10, 11 and 12 that the gamma-based stopping criteria causes the simulation to stop at different generations in each experiment. As explained in Sect. 4.4, gamma-based stopping criteria stops the GA once there is a very small improvement in the maximum fitness value. Figure 9 shows that for Bubble sort algorithm, SVR converges faster as compared to all other three



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression

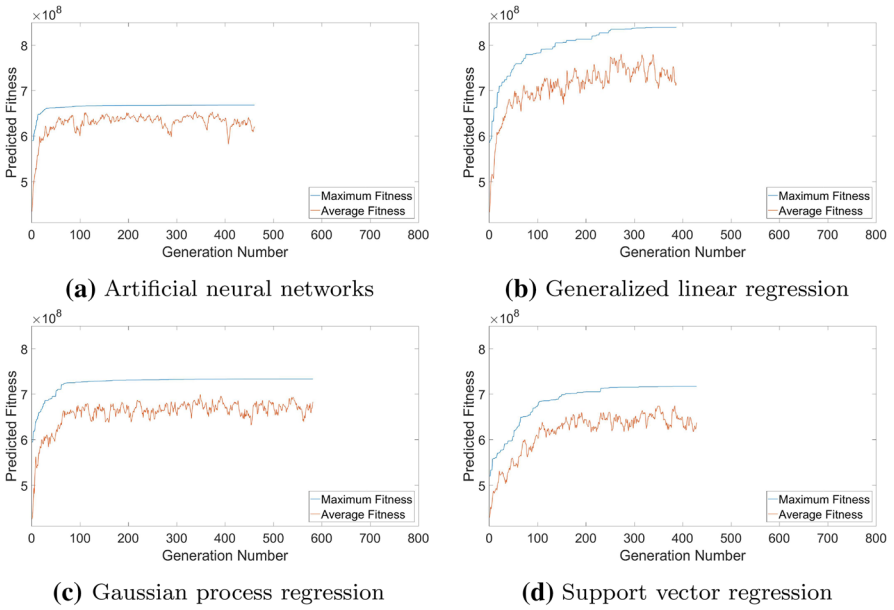
Fig. 9 Predicted maximum and average fitness values in GA-evolution for bubble sort using different prediction models



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression

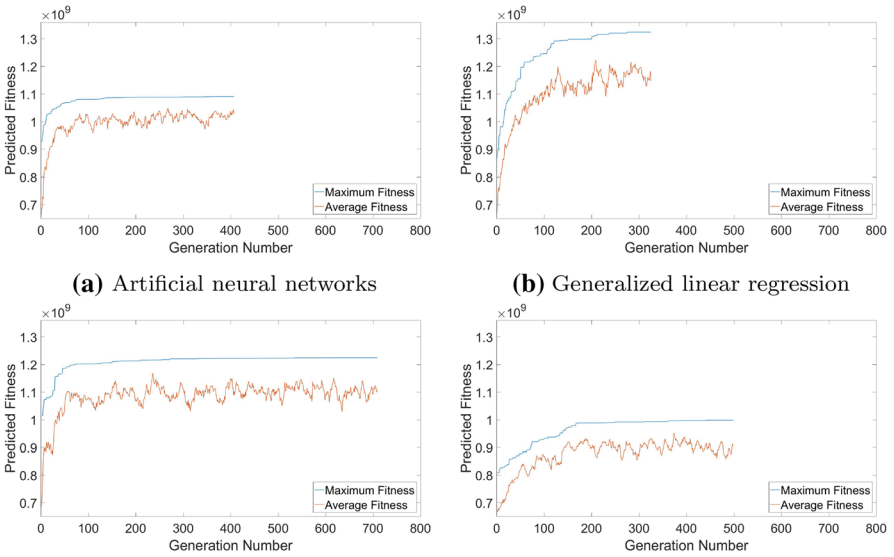
Fig. 10 Predicted maximum and average fitness values in GA-evolution for Insertion sort using different prediction models



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression

Fig. 11 Predicted maximum and average fitness values in GA-evolution for Gnome sort using different prediction models



(a) Artificial neural networks **(b)** Generalized linear regression

(c) Gaussian process regression **(d)** Support vector regression

Fig. 12 Predicted maximum and average fitness values in GA-evolution for Shaker sort using different prediction models

prediction models, whereas GPR converges faster for Insertion sort and Shaker sort algorithms. However, GLM performs better for Gnome sort algorithm. An early stoppage of a GA means that its evolution has matured and the fitness value has reached to its highest level. Hence further generations are not needed. The best time gain achieved is 17.7 times and the best accuracy achieved is 98.5% across the used prediction models, benchmarks and the investigated complexities of the benchmarks. The best overall results are achieved for bubble sort with array size 40 using the GPR model. The time gain is 17.6 times and the accuracy achieved is 98.35%.

While the experiments show that the time gain in WCET analysis is considerable, the accuracy in different cases still needs to be improved. Hence, for the best accuracy for a particular task, a focused research is need on the choice of a best perdiction model and its parameters according to the nature and the complexity of a task.

5.4 Integrated timing performance

This performance measure evaluates the integrated timing performance of a prediction model. In order to do this, the total time taken by a prediction model-based solution is compared with the simulator-based solution. The total time in a prediction model-based solution consists of two components: (1) the time required to generate the training data using simulator and (2) the time required to train the prediction model as well as to evolve the GA successfully.

Table 5 shows the timing analysis of a prediction model-based solution. In Table 5, column 1 lists the benchmarks whereas column 2 represents the training data generation time on the simulator, required to train the prediction models. Remaining columns show the training and evolution time of different models for the benchmarks. It can be observed from Table 5 that the GLM model have taken the least time for training and evolution by the GA as compared to other models whereas the GPR model is slowest as compared to other models.

Table 6 compares the total time taken by a model-based solution (computed in Table 5) with the pure simulator-based solution. Column 1 lists the selected benchmarks whereas column 2 and column 3 represent the total time required in prediction and simulator-based solutions, respectively. It can be observed from Table 6 that the total required time to reach the final solution has been reduced drastically in the prediction model-based solutions as compared to the simulator-based solution.

Table 5 Timing analysis of a prediction model-based GA simulation (all values are in seconds, bold signifies the best integrated timing performance value for each benchmark)

Benchmarks	Total time using prediction models				
	Training data generation time	Training and evolution time			
		ANN	GLM	GPR	SVR
Bubble sort	449.892	221.05	**10.888**	43.665	43.892
Insertion sort	447.321	33.09	5.012	36.837	**4.292**
Gnome sort	438.951	21.516	**1.065**	48.297	8.089
Shaker sort	439.697	9.299	**0.811**	46.173	9.49

Table 6 Comparison of time spent by the prediction model-based GA simulation and pure GA-based simulation (all values are in seconds, bold signifies the best integrated timing performance value for each benchmark)

Benchmarks	Time for training data generation, training and GA-evolution				Time for simulator-based simulation
	ANN	GLM	GPR	SVR	
Bubble sort	670.942	**460.78**	493.557	493.784	8007.751
Insertion sort	480.411	452.333	484.158	**451.613**	9331.768
Gnome sort	460.467	**440.016**	487.248	447.04	7783.08
Shaker sort	448.996	**440.508**	485.87	449.187	6847.275

The advantage of using a prediction model in GA-evolution process is twofold: When a pure GA is used to evolve the population, the data points in the population proceed towards good solutions corresponding to the higher execution time and ultimately approach near to the worst-case execution time. Hence, as the GA evolves, achievement of the desired solutions or data points takes a longer time. On the other hand, prediction model-based GA-evolution process takes constant time per generation. Prediction of execution time by the prediction model is faster than running the benchmark on the simulator for any data point.

5.5 Data quality performance

As described in Sect. 4.4, the process of GA-evolution halts on either the maximum number of generations are reached or the stopping criteria in Eq. (16) is satisfied. In either case, the last generation of GA-evolution process is considered as the final test data. The last generation of every benchmark is then run on the simulator to get the actual execution time ($ET_i^{Obtained}$). The salient feature of the generated test data is that the execution time of the corresponding benchmark for this test data is near to the maximum possible execution time for the benchmark. The description of maximum possible execution time for each sorting algorithm is given in Section 4.3. Consequently, the data quality performance shows the ability of trained model to generate the test data using a GA for which the execution time is near to the maximum.

Table 7 presents 20 best data points for each experiment, determined by the employed methodology, where the execution time is near to the maximum possible execution time. Percentage deviation of the actual execution time ($ET_i^{Obtained}$) in the last generation of the GA from the true worst-case execution time $WCET^{True}$ is calculated as follows:

$$PD_i = \left(\frac{|WCET^{True} - ET_i^{Obtained}|}{WCET^{True}} \right) \times 100 \tag{17}$$

Consequently, the following observations can be recorded from Table 7:

Table 7 Percent deviations (PD_i) of GA's last generation test-data execution-times from the WCET

Benchmarks															
Bubble sort				Insertion sort				Gnome sort				Shaker Sort			
ANN	GLM	GPR	SVR	ANN	GLM	GPR	SVR	ANN	GLM	GPR	SVR	ANN	GLM	GPR	SVR
3.95	4.03	4.51	4.18	9.38	12.36	2.84	9.16	9.17	8.98	1.50	12.37	7.94	8.52	6.77	9.23
5.26	4.07	5.09	4.86	9.50	13.47	2.84	10.15	9.17	8.98	1.50	12.37	11.20	8.60	6.77	10.34
6.05	4.23	5.25	4.91	13.27	14.11	2.84	10.67	9.17	8.98	1.98	13.01	11.29	9.19	6.77	10.34
6.16	4.56	5.43	4.96	13.79	14.18	2.84	10.67	9.60	9.62	1.98	13.11	11.34	9.62	7.27	10.34
6.16	4.71	5.43	4.96	14.61	14.54	4.56	10.67	10.06	10.44	1.98	13.82	11.94	9.64	8.40	10.88
6.16	4.93	5.43	4.97	15.37	14.56	6.74	10.67	10.12	10.54	1.98	13.82	12.09	9.72	8.40	11.24
6.31	5.19	5.45	5.23	15.98	14.67	7.07	11.12	10.83	12.00	1.98	14.27	12.09	10.01	8.55	11.35
6.36	5.43	5.47	5.34	16.04	14.72	9.76	11.13	10.83	12.13	2.57	14.53	12.09	10.06	8.88	11.59
6.84	5.59	5.49	5.34	16.04	14.91	11.29	12.40	10.83	12.51	2.67	14.84	12.09	10.06	8.88	11.64
7.10	5.75	5.59	5.34	16.04	15.22	11.59	12.66	10.90	12.90	4.07	15.41	12.09	10.06	8.92	11.88
7.26	5.96	5.63	5.36	16.65	15.72	12.01	12.71	10.97	13.07	4.88	15.47	12.20	10.06	8.94	12.10
7.42	6.07	5.76	5.74	16.96	15.91	12.09	12.71	11.85	13.07	5.37	15.51	12.36	10.13	9.17	12.21
7.56	6.15	5.82	5.74	17.40	15.91	12.14	13.06	12.37	14.48	5.57	16.98	12.59	11.20	9.27	12.73
7.81	6.22	6.10	5.74	17.44	15.96	12.14	14.26	12.37	14.67	6.09	16.98	12.68	11.46	9.59	13.00
7.94	6.33	6.10	5.74	17.47	16.15	12.17	14.79	12.91	14.67	6.28	16.98	13.01	11.54	9.79	13.08
7.98	6.37	6.36	5.99	17.47	16.15	12.99	14.91	12.99	15.92	6.59	16.98	13.37	11.95	9.97	13.31
8.01	6.39	6.59	6.00	17.47	16.44	13.42	14.91	13.72	16.20	6.70	16.98	13.50	12.22	10.04	13.31
8.17	6.46	6.73	6.10	17.68	16.96	13.50	14.91	13.88	16.22	6.76	16.98	13.72	12.24	10.83	13.31
8.49	6.51	6.74	6.10	18.03	18.48	14.17	14.91	13.93	16.22	7.28	16.98	14.55	12.25	11.24	13.41
8.49	6.51	6.74	6.10	18.18	18.86	14.17	14.91	14.52	16.41	7.32	16.98	14.58	12.67	12.61	14.12

– For bubble sort, both the highest (8.49%) and the lowest (3.95%) deviations have been generated by ANN model.
– For the rest of sorting algorithms, the GPR model provides comparatively better results in terms of percent deviations (PD_i) of measured times from maximum possible execution times.
– The least deviation has been observed in the data generated for bubble sort algorithm
– The maximum deviation has been observed in the data generated for insertion sort algorithm

To summarize, the employed methodology in this article can generate the test data with the minimum deviation of 1.5% (GPR model for Gnome experiment) and the maximum deviation of 18.86% (GLM for Insertion sort).

5.6 Changing array size for sorting algorithms

This section analyzes the effects of various complexities of the problem (different array sizes) on the performance of proposed framework. The performance of the four prediction models as well as the time saved by using the proposed framework are analyzed for array sizes 24, 32 and 40.

For each benchmark, "$Gen_{ANN}, Gen_{GLM}, Gen_{GPR}$" and "$Gen_{SVR}$" are the number of generations after the gamma stopping criteria. Pure simulator-based GA is run for the number of generations "Gen_{Sim}", which is the minimum value in all the four models and calculated as follows:

$$Gen_{Sim} = min\left(Gen_{ANN}, Gen_{GLM}, Gen_{GPR}, Gen_{SVR}\right) \tag{18}$$

Consequently, the time spent in seconds is plotted in Fig. 13 for a pure simulator-based GA-evolution. In Fig. 13, it can be noted that as the array size is increasing, the time spent by a pure simulator-based GA is also increasing for all sorting benchmarks.

Gain in time spent by the model is defined as follows:

$$Ratio_{time} = \frac{Time_{Sim}}{Time_{model}} \tag{19}$$

Where "$Time_{Sim}$" is the time spent by a pure simulator-based GA for the minimum number of generations "Gen_{Sim}". On the other hand, "$Time_{model}$" is the time spent by the model-based GA-evolution with gamma stopping criteria. Hence, ($Ratio_{time}$) gives the ratio of the time consumed by the simulator in actual execution of a benchmark to the time consumed in the estimation of execution time using the prediction models. Gain in time ($Ratio_{time}$) for each model is shown in Table 8, where each value is the mean of $Ratio_{time}$ achieved by all the four benchmarks. The values are aggregated by using a representative value (mean and the standard deviation) because the time gain is almost the same across the benchmarks, evidenced by the small standard deviation values. It can be noted from Table 8 that the time

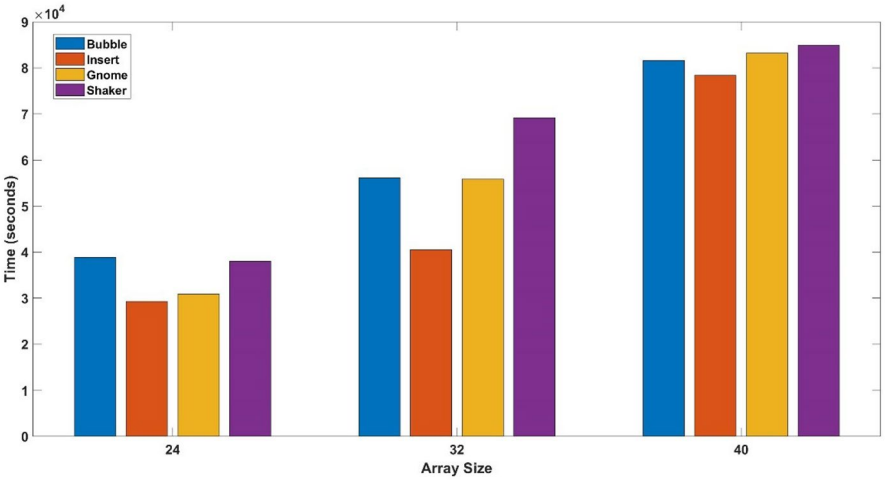


Fig. 13 Time spent by pure simulator-based GA evolution for different array sizes

gain ($Ratio_{time}$) for model-based GA-evolution increases as the array size increases, which supports our claim of time efficiency by using the prediction model-based GA-evolution.

In the following, Tables 9, 10, 11 and 12 show the performance of all four models for different array sizes. For Bubble sort algorithm, GPR prediction model has outperformed other prediction models and predicted the data points which are near to the true worst-case execution time. For the benchmark of Insertion sort algorithm, GPR performance is better than other prediction models whereas GLM has performed better than GPR in case of array size of 24. All prediction models could not perform better for Insertion sort and percentage deviation from the true WCET is higher in all cases. May be, some other prediction model (not targeted in this article) can perform better for Insertion sort. Table 11 shows that the ANN prediction model has provided better results than other prediction models for Gnome sort algorithm for array sizes of 24 and 32. However, for array size of 40, SVR has performed better than others. Further tuning of different parameters of the prediction models may reduce the percentage deviation. Finally, it is interesting to observe in Table 12 that GLM has performed better than other prediction models in case of Shaker sort algorithm.

Table 8 Gain in time ($Ratio_{time}$) for different models against different array sizes

Prediction Model	Array sze 24	Array size 32	Array size 40
ANN	7.8 ± 0.2	14.2 ± 1.6	17.4 ± 2.6
GLM	7.9 ± 0.3	14.8 ± 1.6	17.8 ± 2.6
GPR	7.7 ± 0.3	13.5 ± 1.6	17.6 ± 2.5
SVR	7.8 ± 0.3	13.3 ± 1.6	17.7 ± 2.5

Table 9 Percentage deviation of the best obtained data point from True worst-case execution time (benchmark: bubble sort)

Array size	Prediction models			
	ANN	GLM	GPR	SVR
24	7.84	4.62	**2.16**	5.14
32	6.98	5.51	**4.76**	5.02
40	3.18	1.98	**1.65**	1.91

Table 10 Percentage deviation of the best obtained data point from True worst-case execution time (benchmark: insertion sort)

Array size	Prediction models			
	ANN	GLM	GPR	SVR
24	14.93	**13.92**	15.52	14.54
32	19.00	18.86	**16.95**	18.73
40	20.17	19.64	**18.90**	20.12

Table 11 Percentage deviation of the best obtained data point from True worst-case execution time (benchmark: gnome sort)

Array size	Prediction models			
	ANN	GLM	GPR	SVR
24	**14.38**	15.36	15.95	14.82
32	**13.81**	17.50	18.19	15.28
40	10.69	10.05	9.62	**8.32**

Table 12 Percentage deviation of the best obtained data point from True worst-case execution time (benchmark: shaker sort)

Array size	Prediction models			
	ANN	GLM	GPR	SVR
24	8.86	**8.22**	9.08	8.37
32	15.61	**9.65**	10.02	10.26
40	9.00	5.75	**5.24**	5.26

As evident from Tables 9, 10, 11 and 12, it can be concluded that different prediction models can perform better than others for different benchmarks. Hence, the selection of a prediction model for a benchmark or real-life problem is very important to obtain good performance for generating good solutions (solutions showing highest execution time). Furthermore, prediction models are not limited to just four models, presented in this article, and there are variety of prediction models available in the literature (Agresti 2013; Vapnik 2000). Consequently, a better understanding of the problem (mapping of input data to execution time) can lead to an appropriate selection of the prediction model.

5.7 Limitations and future directions

This study evaluates the performance from the perspectives of prediction accuracy, isolated timing, evolution performance, integrated time and data quality. However, this section indicates the limitations of the current research and provides future directions for more focused investigation.

The choice of a prediction model is a factor that needs to be considered here. Researchers have previously worked on finding the best choice of prediction models using Information Theory to optimise the accuracy and complexity (Akaike 1998; Burnham and Anderson 2004). A best choice prediction model can avoid overfitting and generalise beyond the sample data by balancing the model fit versus complexity (Myung 2000). Hence, the selection of model for a particular problem has as an impact on the prediction performance. It is a challenge to guarantee a single choice of prediction model which retains the best-quality for the entire set of data points or benchmarks. Multiple models were used in previous works in an effort to improve accuracy and reduce the complexity (Kim et al. 2019; Zhang and Shen 2019). However, developing the criteria for selection of the best-quality prediction model was beyond the scope of this study, and can be a worth-investigating area in future.

Although the experimentation in this study is limited to the sorting tasks, the proposed approach is general. However, its application on other benchmarks needs further investigations. Another future research direction could be investigation of the impact of different hardware platforms and prediction models on the performance of the proposed methodology. Similarly, to achieve the best prediction accuracy for a given task, the impact of different parametric choices for a prediction model needs to be investigated further. For example, tailored training of ANN by changing its architecture parameters for a specific task. Moreover, our results show that the time gain increases with the increase in the complexity of a task. However, more research is needed to make sure that the accuracy is still acceptable.

This article has explored the performance of various prediction models, in a GA-based temporal verification process, for real-time systems. The employed prediction models are Artificial Neural Network, Generalized Linear Regression Model, Gaussian Process Regression Model and Support Vector Regression Model. A cycle-accurate simulator is used to train the model so that the trained model can predict the execution time without executing the software. The performance has been explored, for various complexities of the problem, in terms of their prediction performance, timing performance, evolution performance, integrated timing performance and the quality of generated test data. It has been revealed that the time required to generate the worst-case execution-time test data has been reduced up to 17.7 times, and the accuracy achieved ranges from 79.83% to 98.5% across the used prediction models, benchmarks and the investigated complexities of the benchmarks.

Acknowledgements This research was primarily conducted at Science and Technology Unit, Umm Al-Qura University, Makkah. The first author worked on the revisions while being affiliated with the University of South Australia. We acknowledge the funding support of KACST (King Abdul Aziz City for Science and Technology) and NSTIP (National Science Technology, Innovative Plan), Kingdom of Saudi Arabia for this project (12-INF2281-10).

References

Abella J, Hernandez C, Quinones E, Cazorla FJ, Conmy PR, Azkarate-askasua M, Jon P, Enrico M, Tullio V (2015) WCET analysis methods: pitfalls and challenges on their trustworthiness. In: 10th IEEE International Symposium on Industrial Embedded Systems, Siegen, Germany, pp 1–5

Agresti A (2013) Categorical Data Analysis, 3rd edn. Wiley, New York, p 744

Akaike H (1998) Information theory and an extension of the maximum likelihood principle. In: Selected papers of hirotugu akaike. Springer, pp 199–213

Alghamdi MI, Jiang X, Zhang J, Zhang J, Jiang M, Qin X (2017) Towards two-phase scheduling of real-time applications in distributed systems. J Netw Comput Appl 84:109–117

Armaghani DJ, Hasanipanah M, Mahdiyar A, Majid MZ, Anieh HB, Tahir MM (2018) Airblast prediction through a hybrid genetic algorithm-ANN model. Neural Comput Appl 29(9):619–629

Aziz MW, Shah SAB (2015) Test-data generation for testing parallel real-time systems. In: IFIP International conference on testing software and systems. Springer, pp 211–223

Bambagini M, Marinoni M, Aydin H, Buttazzo G (2016) Energy-aware scheduling for real-time systems: a survey. ACM Trans Embed Comput Syst 15(1):1–34

Baruah S, Bertogna M, Buttazzo G (2015) Multiprocessor scheduling for real-time systems. Springer, New York

Bate I, Khan U (2011) WCET analysis of modern processors using multi-criteria optimisation. Empir Softw Eng 16(1):5–28

Bhandari D, Murthy CA, Pal SK (2012) Variance as a stopping criterion for genetic algorithms with Elitist model. Fundam Inform 120(2):145–164

Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S, Sen R, Sewell K, Shoaib M, Vaish N, Hill MD, Wood DA (2011) The gem5 simulator. ACM SIGARCH Comput Archit News 39(2):1–7

Bunte S, Zolda M, Kirner R (2011) Let's get less optimistic in measurement-based timing analysis. In: 6th IEEE international symposium on industrial and embedded systems, Vasteras, pp 204–212

Bunte S, Zolda M, Tautschnig M, Kirner R (2011) Improving the confidence in measurement-based timing analysis. In: 14th International symposium on object/component/service-oriented real-time distributed computing. IEEE, pp 144–151

Buret P, Iguchi-Cartigny J, Grimaud G (2014) Genetic algorithm for DWCET evaluation on complex platform. In: Proceedings of the 9th IEEE international symposium on industrial embedded systems, Pisa, pp 1–4

Burnham KP, Anderson DR (2004) Model selection and multi-model inference, vol 63, 2nd edn. Springer-Verlag, New York

Butko A, Garibotti R, Ost L, Sassatelli G (2012) Accuracy evaluation of GEM5 simulator system. In: 7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC), pp 1–7

Cazorla FJ, Kosmidis L, Mezzetti E, Hernandez C, Abella J, Vardanega T (2019) Probabilistic worst-case timing analysis: taxonomy and comprehensive survey. ACM Comput Surv 52(1):1–35

Deng N, Tian Y, Zhang C (2012) Support vector machines: optimization based theory, algorithms, and extensions. Data Mining and Knowledge Discovery Series. Chapman & Hall/CRC, Taylor & Francis, London, p 363

Dharmajee Rao DTV, Ramesh B (2012) Experimental based selection of best sorting algorithm. Int J Mod Eng Res 2(4):2908–2912

Fahrmeir L, Tutz G (2013) Multivariate statistical modelling based on generalized linear models. Springer Science & Business Media, Berlin, p 426

Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp 249–256
Gustafsson J, Betts A, Ermedahl A, Lisper B (2010) The Malardalen WCET benchmarks: past, present and future. In: International Workshop on Worst-case Execution time Analysis. Brussels, Belgium, pp 137–147
Haftka RT, Villanueva D, Chaudhuri A (2016) Parallel surrogate-assisted global optimization with expensive functions: a survey. Struct Multidiscip Optim 54(1):3–13
Hagan MT, Menhaj MB (1994) Training feedforward networks with the Marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993
Hagan Martin T, Demuth Howard B, Beale Mark H, Jesus Orlando D (2014) Neural Network Design, 2nd edn. Martin Hagan, Dame
Jain BJ, Pohlheim H, Wegener J (2001) On termination criteria of evolutionary algorithms. In: Proceedings of the 3rd annual conference on genetic and evolutionary computation. Morgan Kaufmann Publishers San Francisco, California, USA, p 768
Kelter T, Falk H, Marwedel P, Chattopadhyay S, choudhury AR (2014) Static analysis of multi-core TDMA resource arbitration delays. Real-Time Syst 50(2):185–229
Kim W, Cho W, Choi J, Kim J, Park C, Choo J (2019) A comparison of the effects of data imputation methods on model performance. In: 21st international conference on advanced communication technology (ICACT). IEEE, pp 592–599
Koopialipoor M, Armaghani DJ, Haghighi M, Ghaleini EN (2019) A neuro-genetic predictive model to approximate overbreak induced by drilling and blasting operation in tunnels. Bull Eng Geol Environ 78(2):981–990
Kozyrev VP (2016) Estimation of the execution time in real-time systems. Program. Comput. Softw. 42(1):41–48
Kudjo PK, Ocquaye E, Ametepe W (2017) Review of genetic algorithm and application in software testing. Int J Comput Appl 160(2):1–6
Law S, Bate I (2016) Achieving Appropriate test coverage for reliable measurement-based timing analysis. In: 28th Euromicro conference on real-time systems (ECRTS), Toulouse, pp 189–199
Moayedi H, Moatamediyan A, Nguyen H, Bui X-N, Bui DT, Rashid AS (2019) Prediction of ultimate bearing capacity through various novel evolutionary and neural network models. In: Engineering with computers. Springer, pp -17
Myung IJ (2000) The importance of complexity in model selection. J Math Psychol 44(1):190–204
Nélis V, Yomsi PM, Pinho LM (2015) Methodologies for the WCET analysis of parallel applications on many-core architectures. In: Proceedings of the 2015 Euromicro conference on digital system design, Washington, DC, USA, pp 748–755
Njikam ANS, Zhao H (2016) A novel activation function for multilayer feed-forward neural networks. Appl Intell 45(1):75–82
Nwankpa C, Ijomah W, Gachagan A, Marshall S (2018) Activation functions: comparison of trends in practice and research for deep learning. In: arXiv preprint arXiv:1811.03378
Pongcharoen P, Hicks C, Braiden PM, Stewardson DJ (2002) Determining optimum genetic algorithm parameters for scheduling the manufacturing and assembly of complex products. Int J Prod Econ 78(3):311–322
Puschner PP (1999) Real-time performance of sorting algorithms. Real-Time Syst 16(1):63–79
Rapita Systems Ltd. (2017) Rapita verification suite, http://www.rapitasystems.com/products/rvs. (2017)
Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning, vol 1. MIT Press, Cambridge
Reineke J, Wilhelm R (2016) Static timing analysis: what is special? In: Semantics, Logics, and Calculi, Volume 9560 of the series Lecture Notes in Computer Science, pp 74–87
Rodriguez-Roman D (2018) A surrogate-assisted genetic algorithm for the selection and design of highway safety and travel time improvement projects. Saf Sci 103:305–315
Sastry K, Goldberg DE, Kendall G (2014) Genetic algorithms. Springer, Boston, MA, pp 93–117
Sha L, Caccamo M, Mancuso R, Kim J-E, Yoon M-K, Pellizzoni R, Yun H, Kegley RB, Perlman DR, Arundale G, Bradford R (2016) Real-Time computing on multicore processors. IEEE Comput 49(9):69–77

Shah SAB, Rashid M, Arif M (2017) A prediction model for measurement-based timing analysis. In: Proceedings of 6th ACM International Conference on Software and Computer Applications (ICSCA), Thailand, pp 9–14
Smola AJ, Scholkopf B (2004) A tutorial on support vector regression. Stat Comput 14(3):199–222
Srinivas M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern 24(4):656–667
Surendran A, Samuel P (2016) Evolution or revolution: the critical need in genetic algorithm based testing. Artif Intell Rev, pp 1–47
Tracey N, Clark J, McDermid J, Mander K (2002) A search-based automated test-data generation framework for safety-critical systems. In: Systems Engineering for Business Process Change, pp 174–213
Vapnik V (2000) The nature of statistical learning theory, vol 2. Springer-Verlag, New York, p 314
Wegener J, Grimm K, Grochtmann M, Sthamer H, Jones B (1996) Systematic testing of real-time systems. In: 4th International conference on software testing analysis and review
Wenzel I, Kirner R, Rieder B, Puschner P (2008) Measurement-based timing analysis. In: International symposium on leveraging applications of formal methods, verification and validation. Springer, pp 430–444
Zhang P, Shen C (2019) Choice of the number of hidden layers for back propagation neural network driven by stock price data and application to price prediction. J Phys 1302(2):022017
Zhang Q, Lin M, Yang LT, Chen Z, Li P (2019) Energy-efficient scheduling for real-time systems based on deep Q-learning model. IEEE Trans Sustain Comput 4(1):132–141

Syed Abdul Baqi Shah lives in the beautiful city of Adelaide and likes hiking with kangaroos. He was Lecturer at the Science and Technology Unit, Umm Al Qura University, Makkah, Kingdom of Saudi Arabia. He received a Bachelor of Science in Electronic Engineering from International Islamic University, Pakistan in 2007. He completed his Master of Science in Information and Mechatronics from Gwnangju Institute of Science and Technology, Republic of Korea in 2010. He is pursuing his Ph.D. at the University of South Australia as a Commonwealth funded Australian Government Research Training Program scholar. His research interests include artificial intelligence, embedded systems and real-time systems.

Muhammad Rashid received his Bachelor's degree in electrical engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2000, his Master's degree in embedded systems design from the University of Nice, Sophia-Antipolis, France, in 2006, and his Ph.D. degree in embedded systems design from the University of Bretagne Occidentale, Brest, France, in 2009. He is an Assistant Professor with the Computer Engineering Department, Umm Al-Qura University, Mecca, Kingdom of Saudi Arabia. Prior to joining Umm Al-Qura University, he worked with Thomson Research and Development, Paris, France, and the Advanced Engineering Research Organization, Wah Cantt, Pakistan. His research interests mainly include electronic design automation for embedded systems and engineering education.

Muhammad Arif has done Bachelor of Engineering in 1990 from Ned University, Karachi, Pakistan, MS from Quid-e-Azam University, Islamabad, Pakistan in 1993 and Ph.D. in System Information Sciences from Tohoku University, Japan in 1999. Currently, he is working as Professor in the Department of Computer Science, College of Computer and Information systems, Umm Al-Qura University, Kingdom of Saudi Arabia. He has published more than 100 papers in various journals and conference proceedings. His research interests are Intelligent Pattern Recognition, Biometrics, and biomedical signal processing.