Software Reliability Prediction Using Multi-Objective Genetic Algorithm

Sultan H. Aljahdali
Computer Sciences Department, Al-Taif University
Al-Taif -SAUDI ARABIA
aljahdali@tu.edu.sa

Mohammed E. El-Telbany
Computers Engineering Department, Al-Taif University
Al-Taif -SAUDI ARABIA
telbany@eri.sci.eg

***Abstract*—Software reliability models are very useful to estimate the probability of the software fail along the time. Several different models have been proposed to predict the software reliability growth (SRGM); however, none of them has proven to perform well considering different project characteristics. The ability to predict the number of faults in the software during development and testing processes. In this paper, we explore Genetic Algorithms (GA) as an alternative approach to derive these models. GA is a powerful machine learning technique and optimization techniques to estimate the parameters of well known reliably growth models. Moreover, machine learning algorithms, proposed the solution overcome the uncertainties in the modeling by combining multiple models using multiple objective function to achieve the best generalization performance where. The objectives are conflicting and no design exists which can be considered best with respect to all objectives. In this paper, experiments were conducted to confirm these hypotheses. Then evaluating the predictive capability of the ensemble of models optimized using multi-objective GA has been calculated. Finally, the results were compared with traditional models.**

I. INTRODUCTION

Reliability in the general engineering sense is the probability. It gives component or system in a define environment will operate correctly for a specified period of time. Since the software systems permeate every corner of modern life, and any failure of those systems impacts us. An important issue in developing such software systems is to produce high quality software system that satisfies user requirements. As part of the software engineering process, developers attempt to gauge the reliability of their software, and compare the current level of reliability with the past history of that software. If a software system is experiencing fewer failures as time goes on. The reliability of that system is said to be growing. Answering two questions of when the software should be shipped, and what its reliability will be at that time are based on the use of software reliability models. The basic assumption in software reliability modeling is that software failures are the result of a stochastic process, having an unknown probability distribution. Software reliability models specify some reasonable form for this distribution, and are fitted to data from a software project. Once a model demonstrates a good fit to the available data, it can be used to determine the current reliability of the software, and predict the reliability of the software at future times. The problem is that software systems are so complex such that software engineers are not currently able to test software well enough to insure its correct operation. This may be due to the assumptions made by various software reliability models, or due to there is dependence among successive software runs. The stochastic dependence of successive software runs also depends on the extent to which internal state of software has been affected and on the nature of operations undertaken for execution resumption. Addressing these problems is:

1. By finding mechanisms or relationships to more accurately determine the quality of software systems, without visiting a large fraction of their possible states.
2. Taking in consideration the failure correlation and;
3. Considering there is no single model sufficiently trustworthy in most or all applications

Recently many ways of using *parametric models*, *nonlinear time series* analysis and *data mining* to model software reliability and quality have been investigated. These investigations point the way towards using *computational intelligence* technologies to support human developers in creating software systems by exploiting the different forms of *uncertainty* present in a software system results from infrequent and unpredictable occurrence of human errors and incomplete or imprecise data, in order to model complex systems and support decision making in uncertain environments [22]. These computational intelligence methods are evolving collections of methodologies, which adopt tolerance for imprecision, uncertainty, and partial truth to obtain robustness, tractability, and low cost. Fuzzy logic, neural networks, genetic algorithm, genetic programming and evolutionary computation are the most important key methodologies.

In this paper, genetic-based approach as one of the computational intelligence techniques is followed in predicting software reliability by predicting the faults during the software testing process using software faults historical data. Moreover, a multi-objective genetic algorithm is applied to solve the three problems listed previously by incorporating the possible dependence among successive software run and use ensemble of forecasting models by developing methods for estimating the model(s) parameters with multiple and competing objectives, through the framework of GA optimizing.

978-1-4244-3806-8/09/$25.00 © 2009 IEEE

Detailed results are provided to explore the advantages of using GA in solving this problem. The rest of the paper is organized in the following manner. In Section 2, a brief review of the works carried out in the area of software reliability prediction in research is presented. In Section 3, the genetic algorithms that will be applied in this paper are described briefly. In section 4 and 5, we provide an overview of various SRGM and the data set which we will be used in this paper. Detailed experiments results are provided in section 6. Finally, Section 7 concludes the paper.

II. RELATED WORK

Computationally intelligent technologies find its use software engineering because its focus on system modeling and decision making in the presence of uncertainty. In the last years many research studies has been carried out in this area of software reliability modeling and forecasting. They included the application of neural networks, fuzzy logic models; Genetic algorithms (GA) based neural networks, recurrent neural networks, particle swarm optimization (PSO), Bayesian neural networks, and support vector machine (SVM) based techniques [12]. Cai *et al*. [11] advocated the development of fuzzy software reliability models in place of probabilistic software reliability models (PSRMs). Their argument was based on the proof that software reliability is fuzzy in nature. A demonstration of how to develop a fuzzy model to characterize software reliability was also presented. Karunanithi *et al*. [18] carried out a detailed study to explain the use of connectionist models in software reliability growth prediction. It was shown through empirical results that the connectionist models adapt well across different datasets and exhibit better predictive accuracy than the well-known analytical software reliability growth models. Aljahdali *et al*. [20, 21], made contributions to software reliability growth prediction using neural networks by predicting accumulated faults in a determined time interval. They use a feed forward neural network in which the number of neurons in the input layer represents the number of delay in the input data. For the experiment, they used 4 delays: β_{i-1}, β_{i-2}, β_{i-3} and β_{i-4}, representing the number of failures observed in the previous days before β_i. Ho *et al*. [25] performed a comprehensive study of connectionist models and their applicability to software reliability prediction and found them to be better and more flexible than the traditional models. A comparative study was performed between their proposed modified Elman recurrent neural network, with the more popular feed forward neural network, the Jordan recurrent model, and some traditional software reliability growth models. Numerical results show that the proposed network architecture performed better than the other models in terms of predictions. Despite of the recent advancements in the software reliability growth models, it was observed that different models have different predictive capabilities and also no single model is suitable under all circumstances. Tian and Noore [14] proposed an *on-line* adaptive software reliability prediction model using evolutionary connectionist approach based on multiple-delayed-input single-output architecture. The proposed approach, as shown by their results, had a better performance with respect to next-step predictability compared to existing neural network model for failure time prediction. Tian and Noore [13] proposed an evolutionary neural network modeling approach for software cumulative failure time prediction. Their results were found to be better than the existing neural network models. It was also shown that the neural network architecture has a great impact on the performance of the network. Pai and Hong [19] have applied support vector machines (SVMs) for forecasting software reliability where simulated annealing (SA) algorithm was used to select the parameters of the SVM model. The experimental results show that the proposed model gave better predictions than the other compared methods. Su and Huang [27] showed how to apply neural networks to predict software reliability. Further they made use of the neural network approach to build a dynamic weighted combinational model (DWCM) and experimental results show that the proposed model gave significantly better predictions. Oliveira *et al*. [5, 6] proposed the using of genetic programming (GP) to obtain software reliability model for forecasting the reliability and extended this work by boosting the GP algorithm using re-weighting. The re-weighting algorithm calls many times the learning algorithm with assigned weights to each example. Each time, the weights are computed according to the error (or loss) on each example in the learning algorithm. In this way, the learning algorithm is manipulated to look closer at examples with bad prediction functions. Sheta [1] uses genetic algorithms to estimate the COCOMO model parameters for NASA Software Projects. The same idea is implemented for estimating the parameters of different SRGM models using PSO [2]. In this paper, we explore the use of GA to predict the faults during the software testing process using software faults historical data. Detailed results are provided to explore the advantages of using GA in solving this problem.

III. GENETIC ALGORITHMS

Genetic algorithms are machine learning and optimization schemes, much like neural networks. However, genetic algorithms do not appear to suffer from local minima as badly as neural networks do. Genetic algorithms are based on the model of evolution, in which a population evolves towards overall fitness, even though individuals perish. Evolution dictates that superior individuals have a better chance of reproducing than inferior individuals, and thus are more likely to pass their superior traits on to the next generation. This "survival of the fittest" criterion was first converted to an optimization algorithm by Holland in 1975 [7], and is today a major optimization technique for complex, nonlinear problems. In a genetic algorithm, each individual of a population is one possible solution to an optimization problem, encoded as a binary string called a chromosome. A group of these individuals

will be generated, and will compete for the right to reproduce or even be carried over into the next generation of the population. Competition consists of applying a fitness function to every individual in the population; the individuals with the best result are the fittest. The next generation will then be constructed by carrying over a few of the best individuals, reproduction, and mutation. Reproduction is carried out by a "crossover" operation, similar to what happens in an animal embryo. Two chromosomes exchange portions of their code, thus forming a pair of new individuals. In the simplest form of crossover, a crossover point on the two chromosomes is selected at random, and the chromosomes exchange all data after that point, while keeping their own data up to that point. In order to introduce additional variation in the population, a mutation operator will randomly change a bit or bits in some chromosome(s). Usually, the mutation rate is kept low to permit good solutions to remain stable. The two most critical elements of a genetic algorithm are the way solutions are represented, and the fitness function, both of which are problem-dependent. The coding for a solution must be designed to represent a possibly complicated idea or sequence of steps. The fitness function must not only interpret the encoding of solutions, but also must establish a ranking of different solutions. The fitness function is what will drive the entire population of solutions towards a globally best [4].

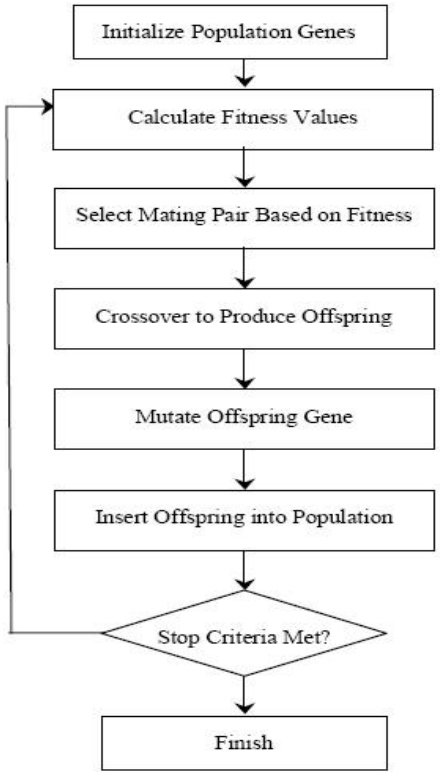

FIGURE I.CANONICAL GENETIC ALGORITHM

Figure 1 illustrates the basic steps in the canonical genetic algorithms. Most GAs has been used for single objective problems, although several multi-objective approaches have been proposed. There are three different approaches to cope with multi-objective problems, namely: 1) transforming the original multi-objective problem into a single objective problem by using a weighted function, 2) the lexicographical approach, where the objectives are ranked in order of priority, and 3) the Pareto approach which consists of as many non-dominated solutions as possible and returning the set of Pareto front to the user. The main conclusions are that the weighted formula approach, which is by far the most used in the data mining literature, is an ad-hoc approach for multi-objective optimization, whereas the lexicographic and the Pareto approaches are more principled approaches, and therefore deserved more attention from the computer science community [16].

IV. PREDICTING MODELS

In the past three decades, hundreds of models were introduced to estimate the reliability of software systems [15, 24]. The issue of building growth models was the subject of many research works [8] which helps in estimating the reliability of a software system before its release to the market. There appear to be three major trends in software reliability research: the use of Non-Homogeneous Poisson Process (NHPP) models, Bayesian inference, and time series analysis. An NHPP is a Poisson process with a time-varying mean value function. Bayesian inference in software reliability models essentially consists of treating the parameters of a reliability model as random variables instead of constants to be estimated. Some reasonable prior distributions are assumed for these parameters, and Bayes' theorem is then invoked to determine the posterior distributions using reliability data. Finally, time series analysis uses an auto-regressive process and an auto-regressive integrated moving average (ARIMA) model. In addition to these three large-scale trends, there are many other proposing software reliability models that are somewhat unique. In this paper, the auto-regression models are adopted.

A. Regression Model

A time series is a time-ordered sequence of observation values of a physical or financial variable made at equally spaced time intervals Δt, represented as a set of discrete values x_1, $x_2, x_3, \ldots$, *etc*. Time series analysis deals with the problems of identification of basic characteristic features of time series, as well as with discovering - from the observation data on which the time series is built - the internal time series structure to predict time series data values which help in deciding about the subsequent actions to be taken. One of most used times series models is the auto regression model. Much of the appeal of this technique lies with its simplicity and also its easy accessibility from many of the popular statistical packages. The AR model can be described by the following equation:

$$y_j = \omega_0 + \sum_{i=1}^{n} \omega_i y_{j-i} \quad (1)$$

where y_{j-i} is the previous observed number of faults and $(i=1,2,\ldots,n)$. The value of n is referred to as the "order" of the model, ω_0 and $\omega_i,(i=1,2,\ldots,n)$ are the model parameter.

B. Multiple Regression Model

Stochastic uncertainty that arises because faults occur during the software testing process can behave in many different unpredictable ways and is thus a property of reality. Reducing reality into a model inevitably results in an error, reflecting the discrepancies between the reality portion of interest and its model representation. These errors can be associated with the structure of the model stemming from simplifications, assumption and approximations or due to uncertainties in the values assumed by the model parameters or due to errors in the measurement process itself. This error can be viewed as a measure of how good a model is in representing reality. Machine learning algorithms, proposed the solution by combining multiple models, we are aiming at a more accurate prediction at the expense of increased uncertainty [26]. The fusion approach, that will be applied combine such as the average predictions of multiple models. Mathematically, the ensemble models can be described by the following equation:

$$y_j = M_j(\Omega_j, S_j) \tag{2}$$

where y_j is the prediction of the model about a reality aspect of interest, S_j represents the model's structure reflecting a set of assumptions and simplifications encoded into the mathematical model M_j, and $\Omega_j = (\omega_0, \omega_1, \omega_3, \ldots)$ is a finite set of model parameters. In a general case of a discrete set of n models Ψ, each model $M_j(\Omega_j, S_j), j=1,2,\ldots,n$ represents an alternate form of S_j with given set of parameters Ω_j. Each model in the set Ψ provides an estimate about the quantity of interest y_j in the form of a predictive probability distribution $P(y \mid Mj) = P(y \mid \Omega_j, S_j)$. The literature on combining methods is very reach and diverse, among the methods: the simple averaging (equal weights) and the weighted average [23]. In this study, the combination function υ is implemented both the schemes, equation 3, represent the average predictions of multiple models and equation 4, represent the weighted average predictions of multiple models.

$$y = \frac{1}{n}\sum_{j=1}^{n} y_j \tag{3}$$

$$y = \frac{1}{n}\left(\sum_{j=1}^{n} w_j y_j\right) \tag{4}$$

V. Problem Formulation

The standard method of performing time series prediction problem can be formulated within the supervised machine learning frameworks as the following two cases:

Case 1: *Given a set of m examples,* $\{(u_i, t_i),\ i=1,\ldots,m\}$ *where* $f(u_i)=t_i, \forall i$, *return a function g that approximates f in the sense that the norm of the error vector* $E=(e_1,..,e_m)$ *is minimized, where each* e_i *is defined as* $e_i = q(g(u_i), t_i)$ *and* $Q=\sum_{i=1}^{m} q_i$ *is an arbitrary error function.*

Case 2: *Given a set of m examples,* $\{(u_i, t_i),\ i=1,\ldots,m\}$ *where* $f(u_i)=t_i, \forall i$, *return functions* $g_j(u_i)=t_{ji}, \forall i$ *that its combination function* $t_i = \upsilon(g_1(u_i), g_2(u_i),\ldots)$ *approximates f in the sense that the norm of the error vector* $E=(e_1,..,e_m)$ *is minimized, where each* e_i *is defined as* $e_i = q(\upsilon(g_1(u_i), g_2(u_i),\ldots), t_i)$ *and* $Q=\sum_{i=1}^{m} q_i$ *is an arbitrary error function.*

The parameters of any model can be thought as the genes vector or sub-vector of the chromosome in the GA. The parameters of each chromosome vector are initialized randomly and are evolved using GA algorithm. The fitness function Q that determines the quality of population members is a multi-objective function that optimizing several performance index: The value of normalized root mean square error (NRMSE) and Correlation Coefficient (R^2) between the observed and forecasted failures. The correlation coefficient, R^2 measures the percentage of variation in the dependent variable that is explained by the regression or trend line. It has a value between zero and one, with a high value indicating a good fit.

$$R^2 = \frac{\sum_{j}^{m} \left(\upsilon(g_1(u_j), g_2(u_j),\ldots) - \bar{y}\right)^2}{\sum_{j}^{m} (y_j - \bar{y})^2} \tag{6}$$

The objective is to carefully ensemble the different forecasting models to achieve the best generalization performance. This task is to have minimal values of NRMSE and a maximum value for R^2. This problem is equivalent to finding the Pareto solutions of a multi-objective optimization problem. A Pareto-optimal solution has the property that it is not possible to reduce any of the objective functions without increasing at least one of the other objective functions. The most straightforward approach to multi-objective optimization is combine all the objectives into a single one using either an addition, multiplication or any other combination of arithmetical operations that we could devise. Where

$$Q = R^2 / RMSE \quad (7)$$

This approach is not computationally intensive and results in a single best solution based on the assigned weights.

VI. Experiments Results

This section describes the data used and the measurements adopted to evaluate the obtained GA model. We also present the main steps followed to configure the GA algorithm. This experiment explored GA models based on time. This is easily achieved with an appropriate terminal set. This terminal set is compound by past accumulated failures.

A. Software Reliability Data Set

John Musa of Bell Telephone Laboratories compiled a software reliability database [9]. His objective was to collect failure interval data to assist software managers in monitoring test status, predicting schedules and to assist software researchers in validating software reliability models. These models are applied in the discipline of software reliability engineering. The dataset consists of software failure data on 16 projects. Careful controls were employed during data collection to ensure that the data would be of high quality. The data was collected throughout the mid 1970s. It represents projects from a variety of applications including real time command and control, word processing, commercial, and military applications. The failure data consists of: project identification, failure number; time between failures (TBF), and day of occurrence. In our case, we used data from three projects. They are Military, Real Time Control and Operating System.

B. Regression Models Structures and Training

The architecture of the regression model used for prediction the software reliability is modeled as in Equation 1; with $n = 4$. For multiple models we combine the three models with $n = 1,2,3$. The chosen orders of AR models are simples to implement the principles of parsimony. The genetic algorithms are learned to estimate the models parameters and their combining weights. The trainings accomplish by dividing the data set into two sections, training and test sets, comprising of 70% and 30% of the total data set respectively. So, we took the first 70, 96 and 194 data points for training in each project respectively, the next 30, 40 and 83 points for validation and test. The GA training algorithms are conducted several pre-experiments to determine the parameters setting per algorithm that yields the best performance with respect to the dataset. These parameters are values are shown in Table 1.

C. Experimental Evaluation

The training data from real time control and their predicted results from different model are shown in Figures 2 and the predicted squared error in Figure 3. The forecasted and actually measured values where compared to verify the generated models by GA learning algorithm. From this figure it can be observed that the weighted average ensemble of models forecast more closely to the actual values than other modeling methodologies in most of the testing time period. The results of runs on this case study training data set summarized in Table 2 in terms of multi-objective function. According to results shown in Table 2; the ensemble of models are better than the single model and the variation between the weighted average and average combination of ensemble is minor. The above results show that AR ensemble models performance can be very dependent on the ability of optimization algorithms to find a good set of parameters. The better performance can be illustrated by showing the learning curves of the parameters of the proposed methodology as shown in Figures 4.

TABLE I. The GA Parameters used in This Study

Parameter	Value
Population Size	25
Number of generations	2000
Crossover rate	0.6
Mutation rate	0.05
Selection method	tournament selection

TABLE II. The comparison among single and ensemble of models learned using GA for training data set

	Training Data Set	
	NRMSE	R^2
single model	3.87e-6	0.93
average ensemble	3.46e-6	0.99
weighted average ensemble	3.44e-6	0.99

TABLE III. The comparison among single and ensemble of models learned using GA for testing data set

	Testing Data Set	
	NRMSE	R^2
single model	4.11e-6	0.97
average ensemble	2.79e-6	0.98
weighted average ensemble	2.66e-6	1.00

The test data from real time control and their predicted results from different model are shown in Figures 5 and the predicted squared error in Figure 6. The results of runs on this case study test data set summarized in Table 3, according to results shown in Table 3; the productivity of ensemble of models are better than the single model and the variation between the weighted average and average combination of ensemble is minor. Comparing these results with a pervious work, that implemented the genetic algorithms with single objective function [2], we found that the performance is enhanced as summarized in Table 4.

TABLE IV. The comparison among single and multiple objective functions for training data set using NRMSE value

	Training Data Set	
	multiple objective	single objective
single model	4.11e-6	7.61E-06
average ensemble	2.79e-6	5.57E-06
weighted average ensemble	2.66e-6	5.57E-06

VII. Conclusions and Future Works

In this work, we have measured the predictability of software reliability using ensemble of models trained using GA. The study is applied on three study sets; Military, Real Time Control and Operating System. As far as the predictability of the single AR model and ensemble of AR models trained by GA algorithm over the trained and test data is concerned, the ensemble of models performed better the single model. Also, we find that the weighted average combining method for ensemble has a better performance in a comparison with average method. This due to the GA learned weights which decide the contribution of each model in the final results. However these models are linear in the future, we plan to use non-liner models like neural networks and other form of ensemble combinations.

VIII. Acknowledgement

The authors would like to thank Dr. M. Maged for reviewing this paper and her valuable comments.

IX. References

[1]. A., Sheta, Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects, Journal of Computer Science, USA, 2(2):118–123, 2006.

[2]. A., Sheta, Reliability growth modeling for software fault detection using particle swarm optimization. In 2006 IEEE Congress on Evolutionary Computation, Sheraton, Vancouver Wall Centre, Vancouver, BC, Canada, July 16-21, pp. 10428–10435, 2006.

[3]. C., Houck, J., Joines, and M., Kay, A Genetic Algorithm for Function Optimization: A MATLAB Implementation, ACM Transactions on Mathematical Software, 1996

[4]. D., Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Boston, Massachusetts, 1989.

[5]. E., Oliveira, A., Pozo, and S., Vergilio, Using Boosting Techniques to Improve Software Reliability Models Based on Genetic Programming, *in* Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence

[6]. E., Oliveira, C., Silia, A., Pozo, and G., Souza, Modeling Software Reliability Growth with Genetic Programming, In Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, 2005.

[7]. J., Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

[8]. J., Musa, Software Reliability Engineering: More Reliable Software, Faster and Cheaper. Published Author House, 2004.

[9]. J., Musa. A theory of software reliability and its application. *IEEE Transactions on Software Engineering*, pages 312–327, 1975.

[10]. JY., Park, SU., Lee and JH., Park, Neural network modeling for software reliability prediction from failure time data. Journal of Electrical Engineering and Information Science 4:533–538, 1999.

[11]. K.Y., Cai, C.Y., Wen, and M.L., Zhang, A critical review on software reliability modeling. Reliability Engineering and System Safety 32 (3), 357–371, 1991

[12]. L. Tian and A. Noore: *Computational Intelligence Methods in Software Reliability Prediction*, in Computational Intelligence in Reliability Engineering (SCI) **39**, 375–398, 2007.

[13]. L., Tian, and A., Noore, Evolutionary neural network modeling for software cumulative failure time prediction. Reliability Engineering and System Safety 87, 45–51, 2005.

[14]. L., Tian, and A., Noore, On-line prediction of software reliability using an evolutionary connectionist model. The Journal of Systems and Software 77, 173–180, 2005.

[15]. M. Xie. Software Reliability Models - Past, Present and Future. *In N. Limnios and M. Nikulin (Eds). Recent Advances in Reliability Theory: Methodology, Practice and Inference*, pages 323–340, 2002.

[16]. M., Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, Massachusetts, 1996.

[17]. N. Raj Kiran, V. Ravi, Software Reliability Prediction by Soft Computing Techniques, J. Syst. Software, 2007.

[18]. N., Karunanithi, D., Whitley, and Y., Maliya, Prediction of software reliability using connectionist models. IEEE Transactions on Software Engineering 18, 563–574, 1992

[19]. P. F., Pai, and W.C., Hong, Software reliability forecasting by support vector machines with simulated vector machines with simulated annealing algorithms. The Journal of Systems and Software 79, 747-755, 2006

[20]. S., Aljahdali A., Sheta and R., Rine, Predicting accumulated faults in software testing process using radial basis function network models. In 17th International Conference on Computers and Their Applications (CATA), Special Session on Intelligent Software Reliability, San Francisco, California, USA, 2002.

[21]. S., Aljahdali D., Rine and A., Sheta, Prediction of software reliability: A comparison between regression and neural network nonparametric models. In ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2001), Beirut, Lebanon, pp.470–473, 2001.

[22]. S., Dick and A., Kandel, *Computational Intelligence in Software Quality Assurance*, World Scientific Publishing Co. 2005.

[23]. S., Hashem, B., Schmeiser, and Y. Yih, Optimal Linear Combinations of Neural Networks: An Overview. *Tech. Rep. SMS93-19,* School of Industrial Engineering, Purdue University. *(Proceedings of the 1994 IEEE International Conference in Neural Networks,* 1993.

[24]. S., Yamada, Software reliability models and their applications: A survey. In *International Seminar on Software Reliability of Man-Machine Systems - Theories Methods and Information Systems Applications - August 17-18, Kyoto University, Kyoto, Japan*, 2000.

[25]. S.L., Ho, M., Xie, and T.N., Goh, A study of connectionist models for software reliability prediction. Computers and Mathematics with Applications 46 (7), 1037–1045, 2003.

[26]. T. G. Dietterich. Ensemble Methods in Machine Learning. In J. Kittler and F. Roli, editors, Multiple Classifier Systems, volume 1857 of Lecture Notes in Computer Science, Cagliari, Italy, Springer, pp. 1–15, 2000.

[27]. Y., Su, and C.,Huang, Neural Network-Based Approaches for Software Reliability Estimation using Dynamic Weighted Combinational Models. Journal of Systems and Software 80 (4), 606–615, 2006.

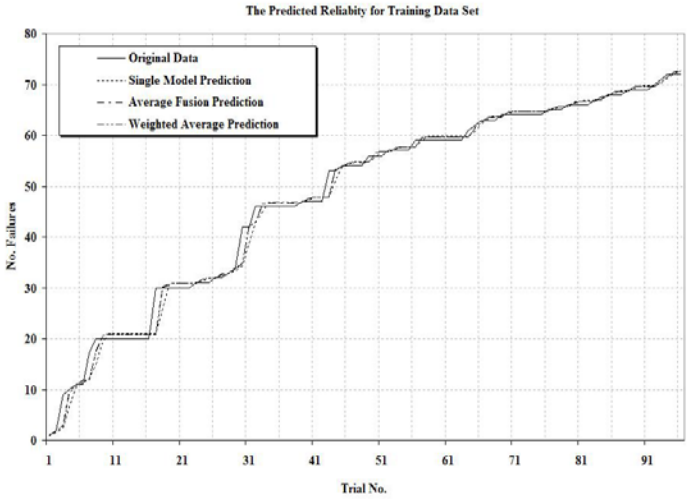

FIGURE II. ACTUAL AND ESTIMATED FAULTS FOR REAL TIME AND CONTROL APPLICATION

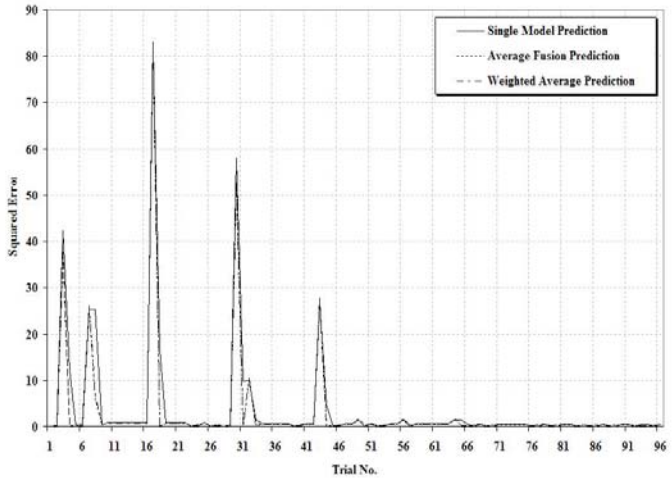

FIGURE III. PREDICTION ERROR FOR REAL TIME AND CONTROL APPLICATION TRAINING SET

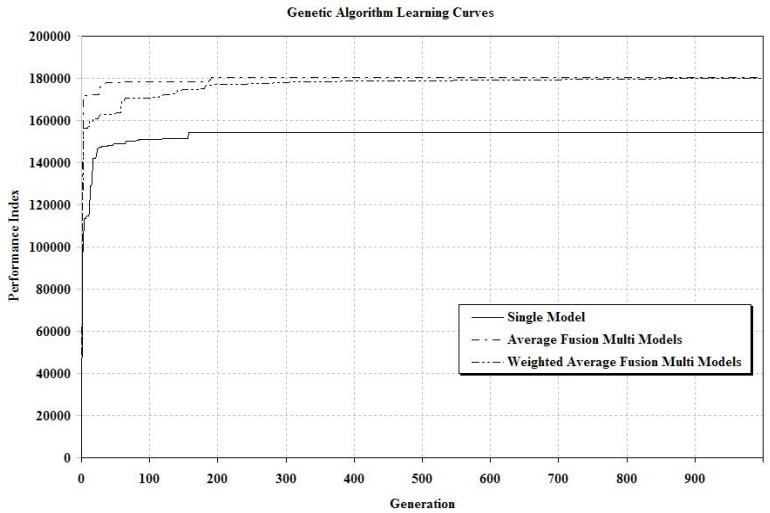

FIGURE IV. LEARNING RATE OF GA FOR REAL TIME AND CONTROL APPLICATION

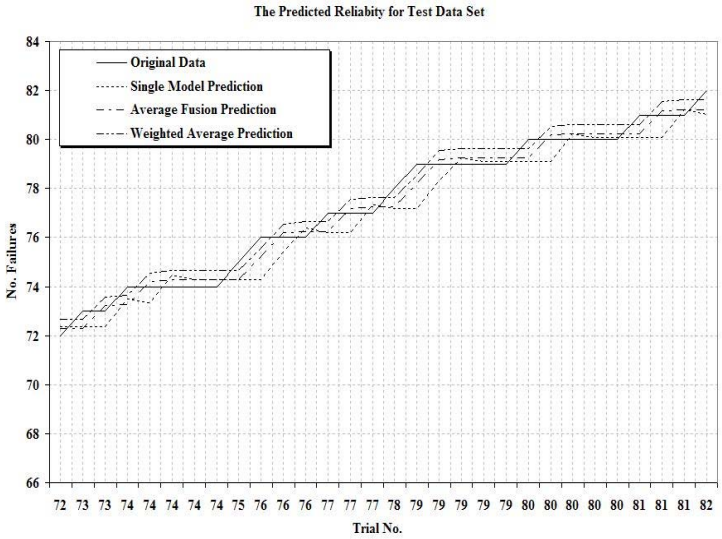

FIGURE V. ACTUAL AND ESTIMATED FAULTS FOR REAL TIME AND CONTROL APPLICATION TEST SET

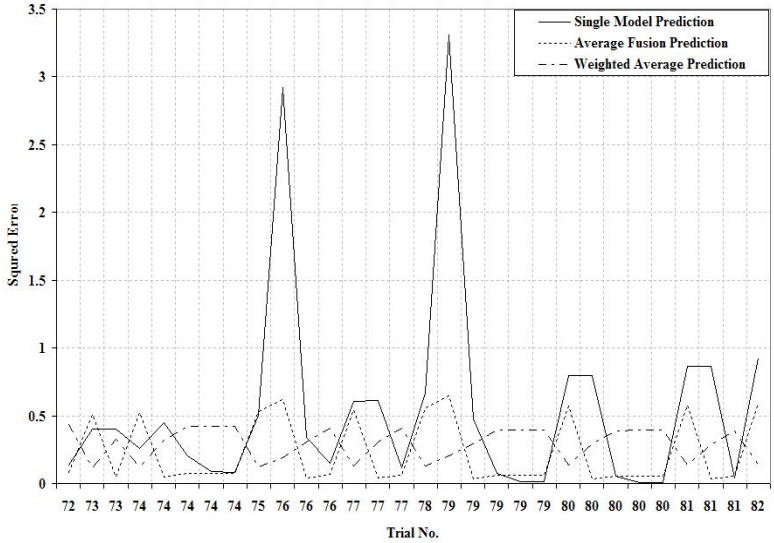

FIGURE VI. PREDICTION ERROR FOR REAL TIME AND CONTROL APPLICATION TEST SET