www.icgst.com

Genetic Algorithms for Optimizing Ensemble of Models in Software Reliability Prediction

Sultan H. Aljahdali[1] and Mohammed E. El-Telbany[2]

[1]College of Computers and Information Systems, Taif University
Taif , Saudi Arabia
[2]Computers and Systems Department
Electronics Research Institute, Cairo, Egypt
Email: aljahdali@tu.edu.sa and 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 aiming at a more accurate prediction at the expense of increased uncertainty. The main motivation to choose GA for this task is its capability of estimating optimal parameters through learning from historical data. In this paper, experiments were conducted to confirm these hypotheses by evaluating the predictive capability of the developed ensemble of models and the results were compared with traditional models.

Keywords: *genetic algorithms, software reliability, prediction, ensemble*

1. Introduction

Reliability, in the general engineering sense, is the probability that a given component or system in a given 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. Addressing this problem is by finding mechanisms or relationships to more accurately determine the quality of software systems, without visiting a large fraction of their possible states. 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 [4]. 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. 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, the genetic algorithms that will be applied in this paper are described briefly. In section 3 and 4, 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 5. Section 6, a brief review of the works carried out in the area of software reliability prediction in research is presented. Finally, Section 7 concludes the paper.

2. 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 [9], 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 [6]. Figure 1 illustrates the basic steps in the canonical genetic algorithms.

3. Predicting Models

In the past three decades, hundreds of models were introduced to estimate the reliability of software systems [26, 27]. The issue of building growth models was the subject of many research works [14] which helps in estimating the reliability of a software system before its release to the market. There appears 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.

3.1 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,…, *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:

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

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.

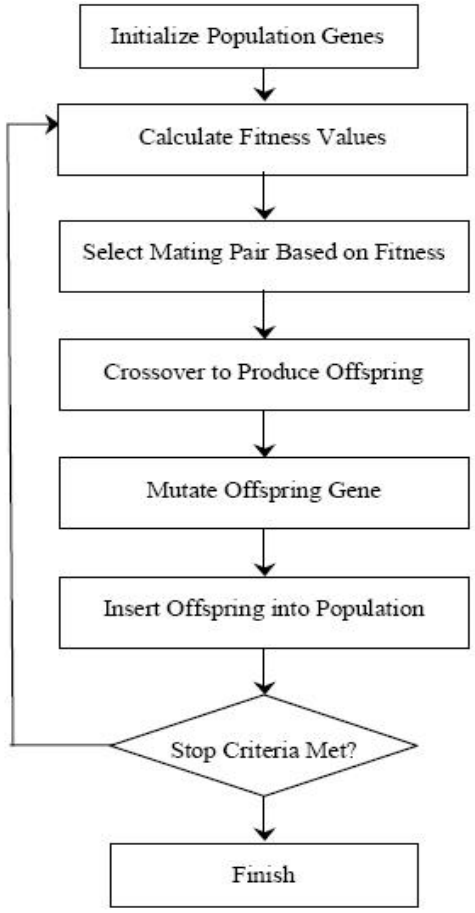



Figure 1, the canonical GA algorithm.

3.2 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 [5]. 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:

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

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 [7]. 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.

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

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

4. 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 the normalized root mean of the sum of the squares of the errors between the actual faults and the value predicted by the model(s) (NRMSE):

$$Q = \frac{1}{m-1}\sqrt{\frac{\sum_{j}^{m}\left(y_j - \upsilon\left(g_1\left(u_j\right), g_2\left(u_j\right), \ldots\right)\right)^2}{\sum_{j=1}^{m}\left(y_j\right)^2}} \qquad (5)$$

5. 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.

5.1 Software Reliability Data Set

John Musa of Bell Telephone Laboratories compiled a software reliability database [13]. 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.

5.2 Regression Models Structures and Training

We implemented the standard genetic algorithms using MATLAB with binary representations where the population contains a set of represented model(s) parameters. Hence, a chromosome is a set of parameters, where each parameters represented by 10 bits. The parameters take the values between [-1, 1]. Initial chromosomes are generated by random bit strings. 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 goal of the genetic algorithm is to naturally select using roulette wheel selection technique, the model(s) that better solves a predication problem using fitness function as in Equation 5. We use the conventional genetic operators to produce new generation of population of chromosomes from current generation. 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.

Table 1, 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

5.3 Experimental Evaluation

The data from Military, Real Time Control and Operating System projects and their predicted results from different model are shown in Figures 3, 4 and 5. The forecasted and actually measured values where compared to verify the generated models by GA learning algorithm. From these figures 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 the three case studies data sets summarized in Table 2 in terms of normalized root mean squares errors (NRMSE). According to results shown in Table 2; the ensemble of models are better than the single model. Moreover, the weighted average and average combination of ensemble more similar, but weighted average more better especially for military and operating system data sets. This better performance can be illustrated by showing the learning curves of the parameters of the proposed methodology as shown in Figures 6, 7, and 8.

Table 2, the comparison among software reliability prediction using single and ensemble of models learned using GA.

	Training Data Set		
	NRMSE		
	Military	OS	RT
single model	3.55E-06	5.75E-07	7.61E-06
average ensemble	3.40E-06	4.72E-07	5.57E-06
weighted average ensemble	3.38E-06	4.71E-07	5.57E-06

6. 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 [23]. Cai *et al*. [3] 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*. [11] 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*. [1, 2], 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*. [8] 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 [25] 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 [24] 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 [17] 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 [22] 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*. [15, 16] 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 [20] 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 [21]. 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.

7. Conclusion and Future Work

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. Moreover, we plan to use multiple objective function functions to achieve a best generalization performance.

Acknowledgement

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

References

[1]. Aljahdali S., Rine D., and Sheta A., 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.

[2]. Aljahdali S., Sheta A., and Rine D., 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.

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

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

[5]. Dietterich T., 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.

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

[7]. Hashem S., Schmeiser B., and Yih Y., 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.

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

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

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

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

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

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

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

[15]. Oliveira E., Pozo A., and Vergilio S., 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

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

[17]. Pai P., and Hong W., 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

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

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

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

[21]. Sheta A., 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.

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

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

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

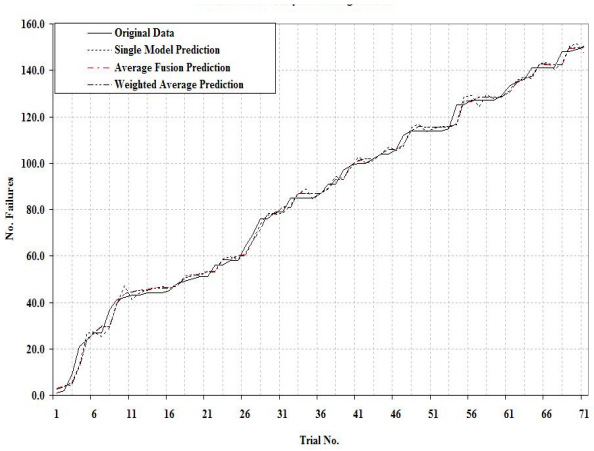
[25]. Tian L., and Noore A., On-line prediction of software reliability using an evolutionary connectionist model. The Journal of Systems and Software 77, 173–180, 2005.

[26]. Xie M., 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.

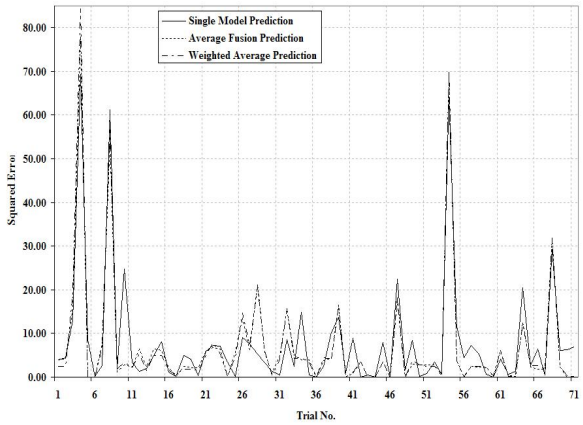
[27]. Yamada S., Software reliability models and their applications: A survey. In *International Seminar on Software Reliability of Man-Machine Systems - Theories Methods and Information Systems Applications - August 17-18, Kyoto University, Kyoto, Japan*, 2000.

Sultan Hamadi Aljahdali, Ph.D. received the B.S from Winona State University, Winona, Minnesota in 1992, and M.S. with honor from Minnesota State University, Mankato, Minnesota, 1996, and Ph.D. Information Technology from George Mason University, Fairfax, Virginia, U.S.A, 2003. He is an associate dean of the college of computers and information systems at Taif University. His research interest includes software testing, developing software reliability models, soft computing for software engineering, computer security, reverse engineering, and medical imaging, also he is a member of ACM, IEEE, and ISCA.

Mohammed El-Telbany, Ph.D. was born in Dammitta, Egypt, in 1968. He received the B.S. degree in computer engineering and science from the University of Minufia in 1991 and the M.Sc. and Ph.D degree in Computer Engineering, from Electronics and Communication Department, Cairo University, Faculty of Engineering, in 1997 and 2003 respectively. He has been an associative professor at the Electronics Research Institute. He has also worked at the ESA at European Space Research Institute (ESRIN), 1998-1999, Frascati, Italy, at the Faculty of Engineering, Al-Ahliyya Amman University, Jordan, 2004-2005, College of Computer Sciences, King Khalid University, KSA, 2005-2008 and College of Computer Sciences, Taif University, KSA. He has been involved in the field of autonomous mobile robots and machine leaning. His pervious research includes work on Evolutionary Computation and Forecasting. Current research includes work in robotics and reinforcement learning, and swarm intelligence.

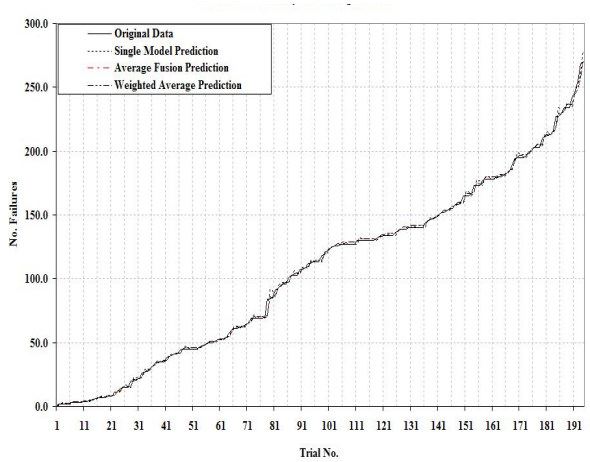


(a)

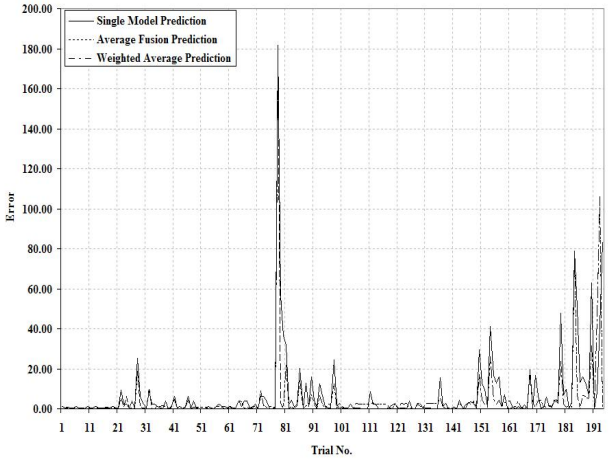


(b)

Figure 3, (a) Actual and Estimated Faults(b) Prediction error: Military Application

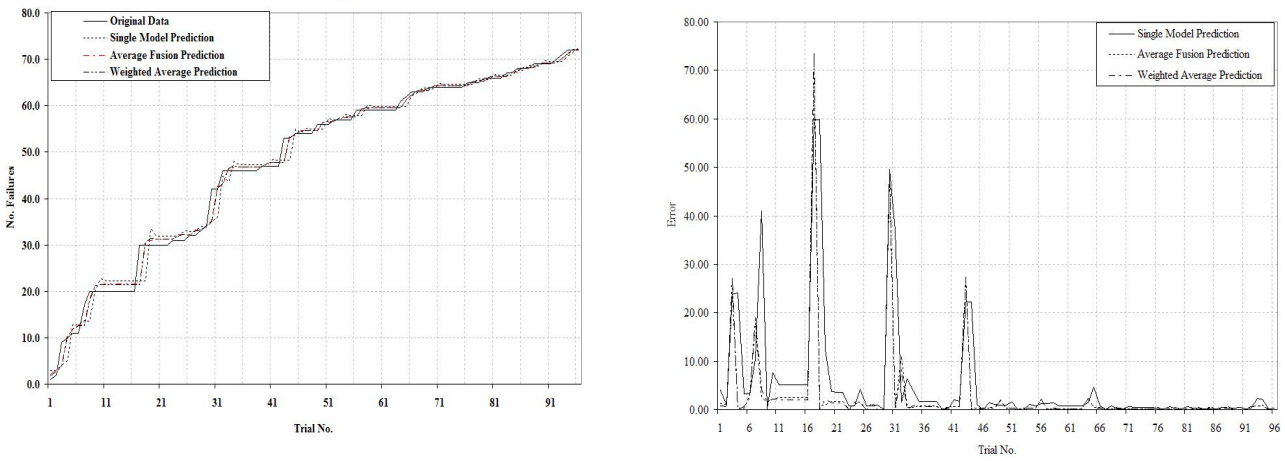


(a)



(b)

Figure 4, (a) Actual and Estimated Faults(b) Prediction error: Operating System Application



(a) (b)

Figure 5, (a) Actual and Estimated Faults(b) Prediction error: Real Time and Control Application

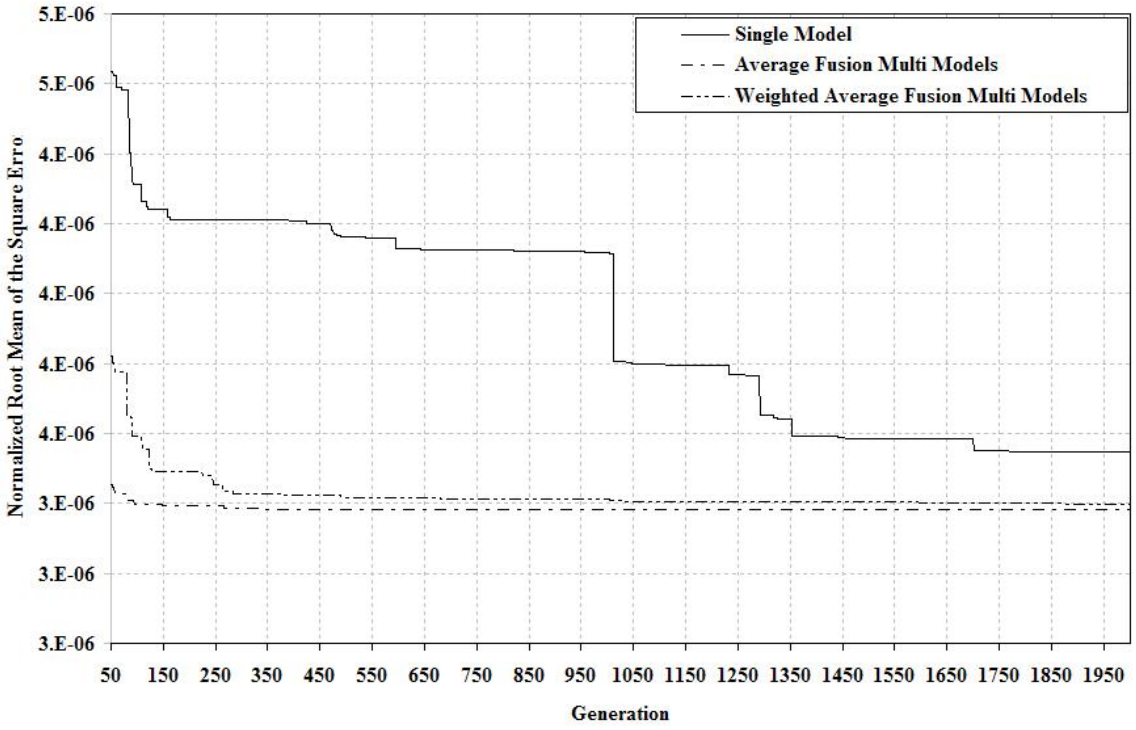


Figure 6, Learning rate of GA for Military Application

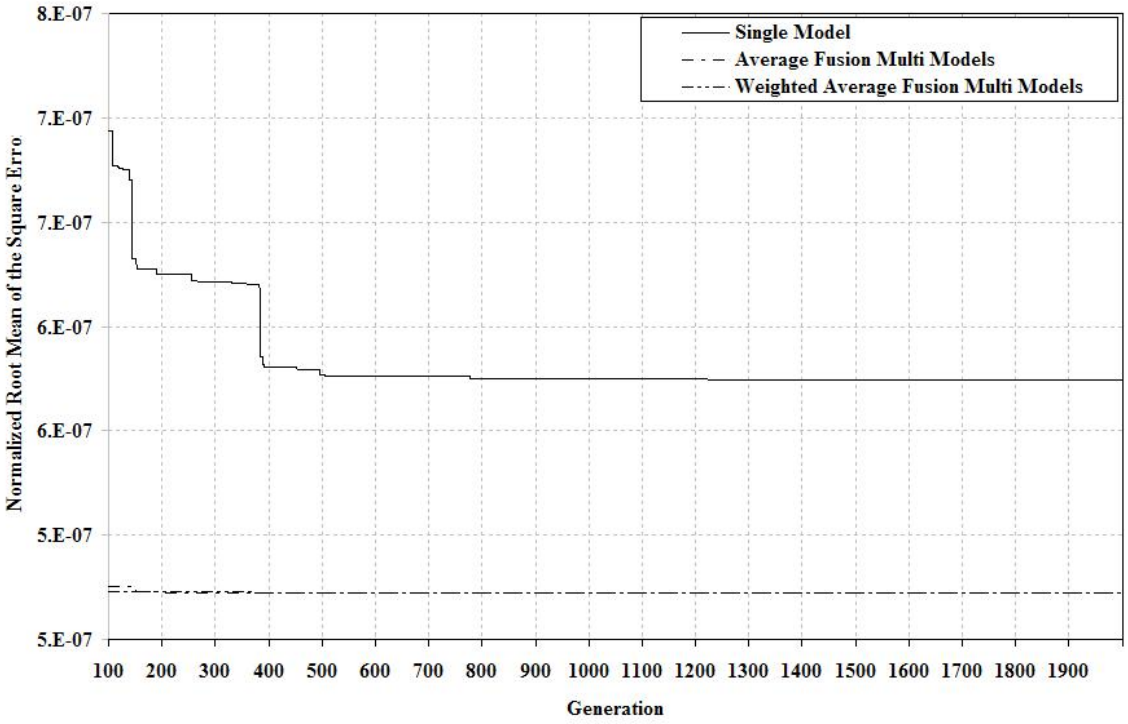


Figure 7, Learning rate of GA for Operating System Application

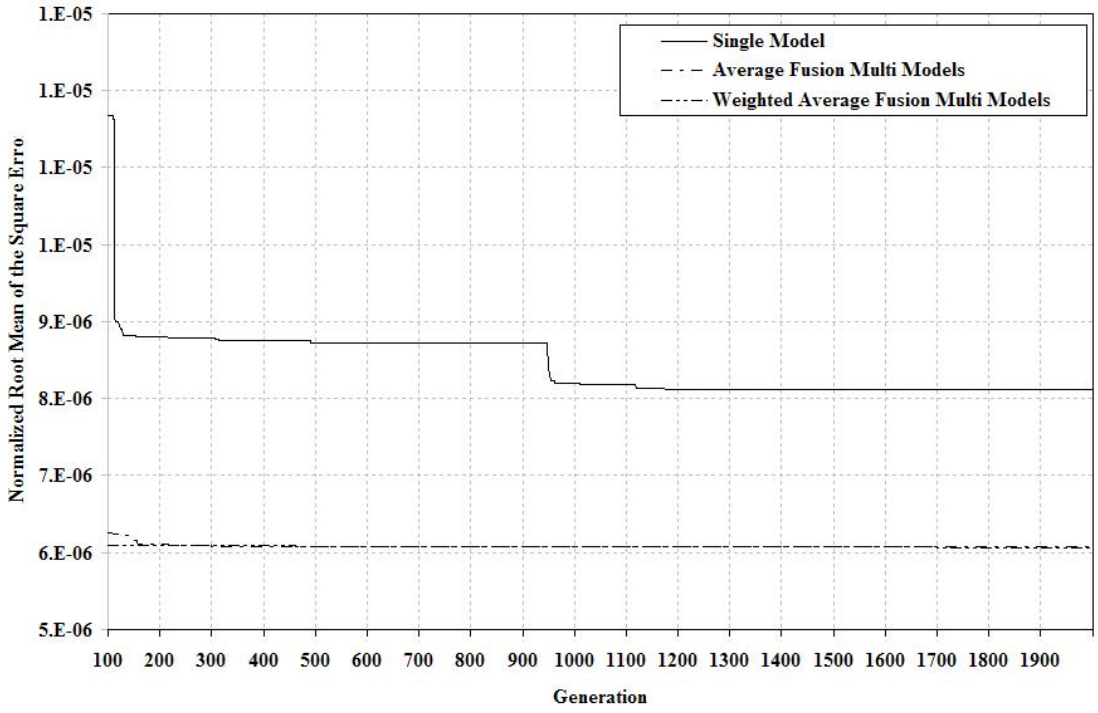


Figure 8, Learning rate of GA for Real Time and Control Application