2011 Eighth International Conference on Information Technology: New Generations

EVOLUTIONARY PREDICTION FOR CUMULATIVE FAILURE MODELING: A COMPARATIVE STUDY

Mohamed Benaddy
Department of Mathematics
and Computing Science
Ibn Zohr University
Agadir, Morocco
benaddym@yahoo.fr

Sultan Aljahdali
Department of Computer Science
College of Computers and Information
Technology
Taif University, Taif, Saudi Arabia
Aljahdali@tu.edu.sa

Mohamed Wakrim
Department of Mathematics
and Computing Science
Ibn Zohr University
Agadir, Morocco
wakrim_m@yahoo.fr

ABSTRACT

In the past 35 years more than 100 software reliability models are proposed. Most of them are parametric models. In this paper we present a comparative study of different non-parametric models based on the neural networks and regression model learned by the real coded genetic algorithm to predict the cumulative failure in the software. Experimental results show that the training of different models by our real coded genetic algorithm have a good predictive capability across different projects.

Keywords: *Software Reliability, Genetic Algorithms, Real Coded Genetic Algorithms, Feed-forward Neural Networks, Auto Regression Model.*

1. INTRODUCTION

Software reliability is defined as the probability of failure free software operation for a specified period of time in a specified environment [1]. Society's reliance on large complex systems mandates high reliability. Reliable software is a necessary component. Controlling faults in software requires that one can predict problems early enough to take preventive action. In the past 35 years more than 100 software reliability models have been developed to solve reliability models [19]. Most of these models as the models of software reliability growth depend on a certain a priori assumptions about the nature of software faults and the stochastic behavior of software process [5]-[6]. As a result, different models have different predictive performance at different testing phases across various projects. A single universal model that can provide highly accurate predictions under all circumstances without any assumptions is most desirable [22]-[14]. Neural network approach has proven to be a universal approximator for any non-linear continuous function with an arbitrary accuracy [6]-[16]-[17]-[18]. Consequently, it has become an alternative method in software reliability modeling, evolution and prediction. Karunanithi [14]-[15] were the first to propose using neural network approach in software reliability prediction. Aljahdali [4]-[3], Adnan [2], Park [22] and Liang [17]-[18] have also mad contributions to software reliability predictions using neural networks, and have gained better results compared to the traditional analytical models with respect to predictive performance.
The most popular training algorithm for feed-forward neural networks is the back-propagation algorithm, the back propagation learning algorithm provides a way to train multilayered feed-forward neural networks [3] but the optimal training of neural network using conventional
gradient-descent methods is complicated due to many attractors in the state space.
Regression models are the most popular methods for building models and are used to calibrate almost all of the models [3]. Regression prediction models are one of the proposed models to predict the number of faults in the software as shown by Aljahdali [4]-[3]. Least square estimation is the most technique used to estimate linear models as observed in the literature [4]. In this paper we have developed a real coded genetic algorithm (RCGA) as an alternative to estimate both the neural network and the auto-regression models proposed by Aljahdali [4]-[3], that optimizes the error made by those models. Evolving the parameters set for a neural network and the regression model with the inverted error as a fitness function.

2. SOFTWARE RELIABILITY DATA SET

The Software Reliability Dataset was compiled by John Musa of Bell Telephone Laboratories [7]. His objective was to collect failure interval data to assist software managers in monitoring test status and 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. In our case, we use data from three different projects. They are Military, Real Time Control and Operating System. The failure data were initially stored in arrays, ordered by day of occurrence so that it could be processed.

3. Neural Network Architecture

The architecture of the network used for modeling software reliability problem is a multi-layer feed-forward network. It consists of an input layer, one hidden layer and an output layer [4]-[3]. The input layer contains a number of neurons equal to the number of delayed measurements allowed to build neural networks model in our case, there are four inputs to the network, they are *Y(k-1), Y(k-2), Y(k-3), and Y(k-4). Y(k-1)* is the observed faults one day before the current day. The hidden layer consists of two nonlinear neurons and two linear neurons. The output layer consists of one output neuron producing the estimated value of the fault. There is no direct connection between the network input and output. Connections occur only through the hidden layer. The hidden units are fully connected to both the input and output. The structure of the adopted neural network is shown in figure 1.

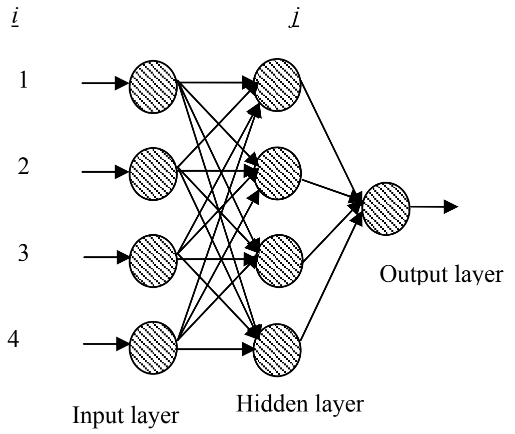

Figure 1: Feed-forward neural network structure

4. Regression model

One of the most famous regression models is the Auto-Regressive models. This model has been used in many applications. This model can work as the reliability growth model if the model variables are redefined as follows [4]:

$$Y(k) = \theta_0 + \sum_{i=1}^{n} \theta_i Y(k - \tau_i) \qquad \textbf{(1)}$$

$Y(k - \tau_i)$ is the observed cumulative failure *n* day before the current day.

5. The Genetic Algorithm

The genetic algorithm was developed and formalized by Holland [13]. It was further developed and shown to have wide applicability by Goldberg [10]. Schaffer [24] showed that it could be used to improve the learning ability of neural networks for simple pattern discrimination on a small data set. Evolving the weight set for a neural net with the inverted error as a fitness function has also been studied [12]. Although there are many possible varieties on the basic GAs, the operational of every genetic algorithm is described in the following steps:

1. Randomly create an initial population of chromosomes.
2. Compute the fitness of every member of the current population.
3. If there is a member of the current population that satisfies the problem requirements then stop. Otherwise continue to the next step.
4. Create an intermediate population by extracting members from the current population using a selection operator.
5. Generate a new population by applying the genetic operators of crossover and mutation to this intermediate population.
6. Go back to step 2.

6. Real Coded Genetic Algorithms

The most common representation in GAs is binary [11]. The chromosomes consists of a set of genes, which are generally characters belonging to an alphabet {0, 1}. Therefore, a chromosome is a vector *C* consisting of *l* genes c_i: $C=(c_1,c_2,\ldots,c_l)$, $c_i=\{0,1\}$, Where *l* is the length of the chromosome.

However in the optimization problems of parameters with variables in continuous domains, it is more natural to represent the genes directly as a real numbers since the representation of solution are very close to the natural formulation, i.e. there are no differences between the genotype and the phenotype. The use of this real-coding in numerical optimization on continuous domains appears in Michalewicz [20].

In this case, a chromosome is a vector of floating point numbers. The chromosome length is the vector length of the solution of the problem; thus, each gene represents a variable of the problem. The gene values are forced to remain in the interval established by the variables they represent, so many genetic operators are developed for them, such as, Flat crossover [23], Arithmetic crossover [21] and BLX-α crossover [9] for the crossover operators and Random mutation and non-uniform mutation [21].

7. THE RCGA TO ESTIMATE NEURAL NETWORK AND AUTO-REGRESSION PARAMETRS FOR SOFTWARE RELIABILITY PREDICTION

As mentioned above, real coding is the most suitable coding for continuous domains. Since our goal is feed-forward neural network training and auto-regression parameters which predict the cumulative future faults in the software, it appears logical to use this coding and genetic operators associated to it. Among the advantages of using real-valued coding over binary coding is increased precision. Binary coding of real-valued numbers can suffer loss of precision depending on the number of bits used to represent one number. Moreover, in real-valued coding chromosome string become much shorter. For real-valued optimization problems, real-valued coding is simply much easier and more efficient to implement, since it is conceptually closer to the problem space. In particular, our aim is to train a feed-forward NN and auto-regression to predict future faults in the software from the previous four discovered faults.

A. THE RCGA NEURAL NETWORK

A chromosome consists of all the network weights. One gene of a chromosome represents a single weight value. In our case there are *4x4* weights for the input-layer plus *4x1* biases plus *4x1* weights plus *1x1* biases for the output-layer so, the length of the chromosome is *l= 4x4 + 4x1 + 4x1 + 1x1 = 25*. The weights and biases of the neural network are placed on a chromosome as shown in figure 2.

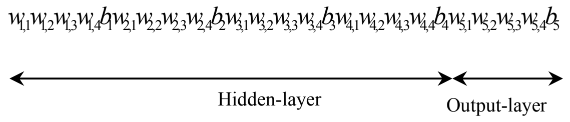
$w_{1,1}w_{1,2}w_{1,3}w_{1,4}b_1w_{2,1}w_{2,2}w_{2,3}w_{2,4}b_2w_{3,1}w_{3,2}w_{3,3}w_{3,4}b_3w_{4,1}w_{4,2}w_{4,3}w_{4,4}b_4w_{5,1}w_{5,2}w_{5,3}w_{5,4}b_5$
Hidden-layer
Output-layer

Figure 2: The chromosomal representation of the neural network

B. THE RCGA AUTO-REGRESSION MODEL

A chromosome consists of all the parameters. One gene of a chromosome represents a single parameter value. In our case there are 5 parameters for the 4-auto-regression. The length of the chromosomes is *l=* 5 and the parameters of the auto-regression are placed on a chromosome as shown in figure 3.

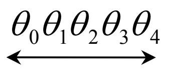

Figure3: The chromosomal representation of the auto-regression model.

Fitness function: the fitness function should reflect the individual's performance in the current problem. We have chosen 1/(1+mse) as a fitness function Eq. (3), where *mse* is the mean squared error during training defined in Eq (2).

$$mse = \frac{1}{n}\sum(\beta_i - \hat{\beta}_i)^2$$

(2)

Where n is the number of training faults used during the training process, β_i and $\hat{\beta}_i$ are the actual and the predicted output respectively during the learning process.

$$fitness = \frac{1}{1+mse} \quad \textbf{(3)}$$

Selection mechanism: The roulette wheel selection is used to create the intermediate population. For each chromosome Ci in a population P, the probability ps(Ci), of including a copy of this chromosome in the intermediate population P' is calculated as in Eq. (4)

$$p_s(C_i) = \frac{fitness(C_i)}{\sum_{j=1}^{P} fitness(C_j)} \quad \textbf{(4)}$$

Where P is the number of individuals in the population.

8. THE REAL CODED GENETIC ALGORITHM TRAINING AND TESTING RESULTS

The initial parameters were randomly chosen in the interval [0, 1]. For each project we performed a number of simulations with a population of 200 individuals and a maximum of generation equal to Gmax. After the training process the Normalize Root Square Error (NRMSE, see Eq. 6) is computed to compare the results obtained by real coded genetic algorithm with these obtained by the least square estimation.

$$NRMSE = \frac{1}{n}\sqrt{\frac{\sum_{i=1}^{n}(\beta(i)-\hat{\beta}(i))^2}{\sum_{i=1}^{n}(\beta(i))^2}} \quad \textbf{(5)}$$

The results of *NRMSE* obtained by the neural network and the regression model [3], respectively learned by back-propagation learning algorithm and least square estimation, are given as follows.

Table 1: A comparison between Regression model order 4 and neural network model in testing case (NRMSE)[3].

Project Name	Military	Real Time Control	Operating System
Number of Faults	101	136	277
Training Data	71	96	194

Testing Data	101	136	277
Regression Model	3.1434	1.7086	1.0659
Neural Networks	1.0755	0.5644	0.7714

The results of *MSE* and *NRMSE* obtained, by the training with our real coded genetic algorithm in training and testing phases are given in table 2.

Table 2: Results for the *MSE* and NRMSE obtained using NNs and AR-4 trained by RCGA in the training and testing phases.

Project Name	Military	Real Time Control	Operating System
Number of Faults	101	136	277
Training Data	**71**	**96**	**194**
Neural network			
MSE	2.859155	2.0515463	2.0515463
NRMSE	2.0635e-4	5.3898e-4	3.4186e-5
Auto-Regression order 4 model			
MSE	1.7323943	2.6185567	1.7474227
NRMSE	1.6062E-4	6.0893E-4	3.1551E-5
Testing Data	**101**	**136**	**277**
Neural network			
MSE	4.2277226	2.6617646	3.0758123
NRMSE	4.7373e-5	3.1216e-4	1.5705e-5
Auto-Regression order 4 model			
MSE	2.6930692	2.7867646	2.1227436
NRMSE	3.7809E-5	3.1940E-4	1.3047E-5

In figure 4 to 12 we give the testing results and the error difference for various projects using the neural network and auto-regression order 4 trained by our real coded genetic algorithm.

9. CONCLUSION

In this paper, we present a comparative study of two approaches for modeling the software cumulative failure. Real Coded Genetic algorithms are used in the training phases of both Neural Networks and Auto-regression models.

Experimental results show that our proposed approaches adapts well across different projects, and has a better performance compared to the results obtained by classical learning methods.

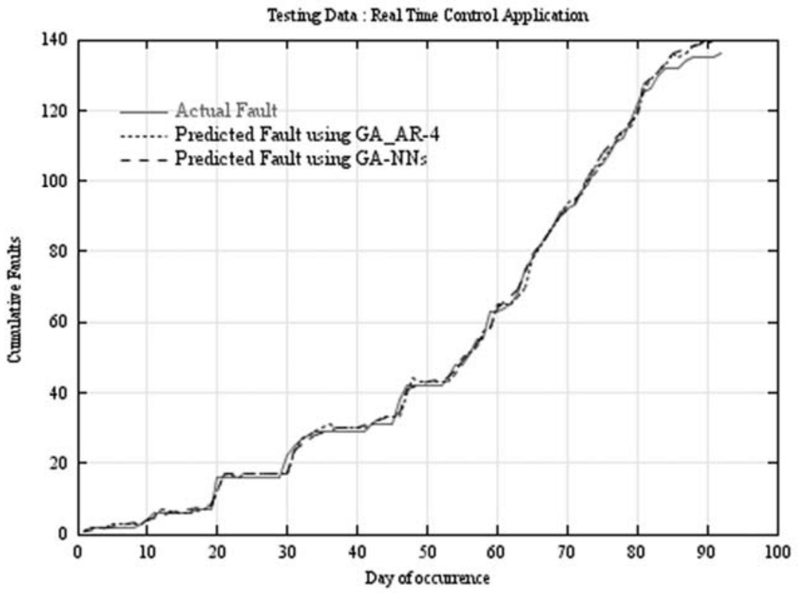

Figure 4: Actual and Predicted Faults using GA-AR and GA-NNs in Testing phase: Real Time Application.

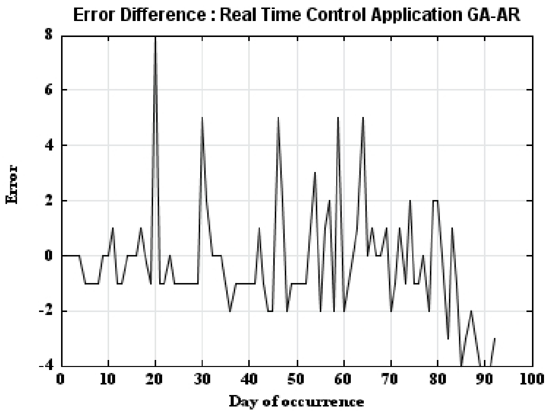

Figure 5 : GA-AR-4 Error Prediction in testing phase: Real Time Control Application.

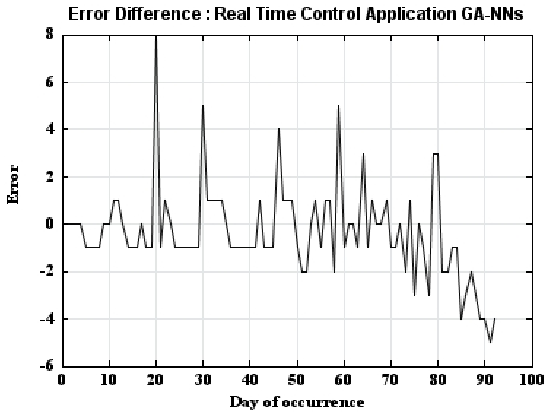

Figure 6 : GA-NNs Error Prediction in testing phase: Real Time Control Application.

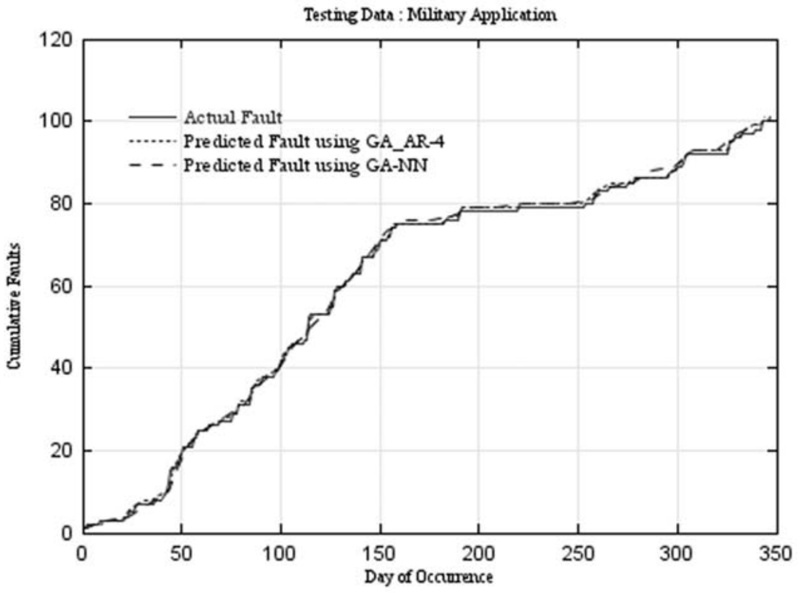

Figure 7: Actual and Predicted Faults using GA-AR and GA-NNs in Testing phase: Military Application.

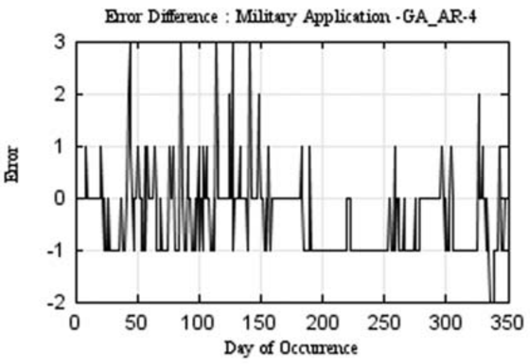

Figure 8 : GA-AR-4 Error Prediction in testing phase: Military Application.

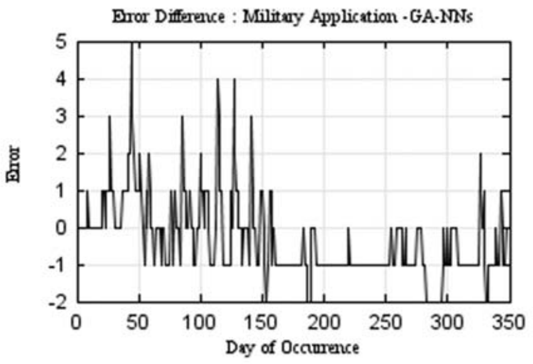

Figure 9 : GA-NNs Error Prediction in testing phase: Military Application.

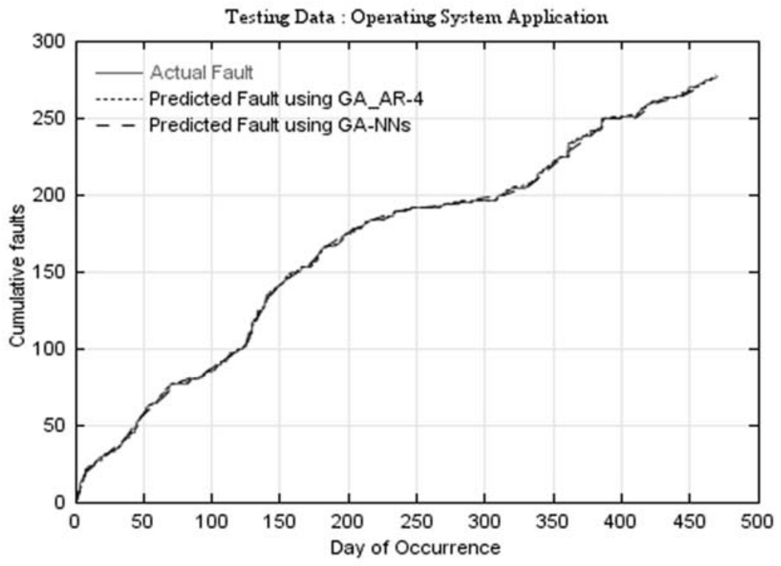

Figure 10 : Actual and Predicted Faults using GA-AR and GA-NNs in Testing phase: Operating System Application.

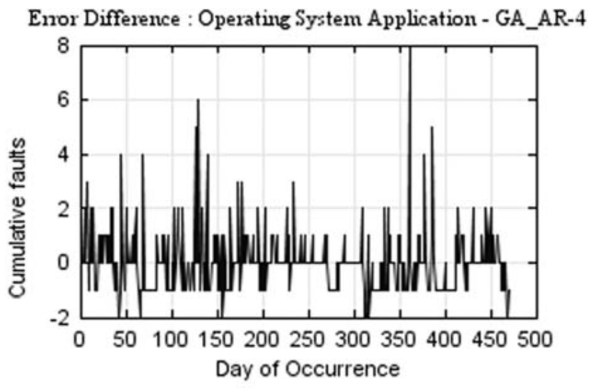

Figure 11: GA-AR-4 Error Prediction in testing phase: Operating System Application.

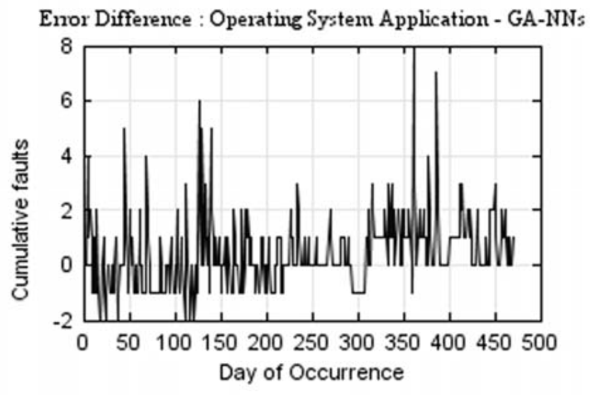

Figure 12: GA-NNs Error Prediction in testing phase: Operating System Application.

10. REFERENCES

[1] ANSI /IEEE, "Standard Glossary of Software Engineering Terminology," STD-729-1991, ANSI /IEEE, 1991.

[2] W.A. Adnan, M.H. Yaacob, "An integrated neural-fuzzy system of software reliability prediction." In: Proceeding of the First International Conference on software Testing, Reliability and Quality Assurance, New Delhi, India, 1994.

[3] S. Aljahdali, K.A. Buragga, "Evolutionary Neural Network Prediction for Software Reliability Modeling" The 16th International Conference on Software Engineering and Data Engineering (SEDE-2007).

[4] S. Aljahdali, A. Sheta, and D. Rine, "Prediction of Software Reliability: A Comparison between regression and neural network non-parametric Models", Proceeding of the IEEE/ACS Conference, 25-29, June 2001.

[5] K.Y. Cai, C.Y. Xen, M.L. Zhang, "A critical review on software reliability modeling. Reliability Engineering Safety," 1991.

[6] Cai K.Y., L. Cai, W.D. Wang, Z.Y. Yu and D. Zhang, "On the Neural Network approach in software reliability modeling," J Syst Software 2001.

[7] Data & Analysis Centre for Software DACS https://www.thedacs.com/databases/sled.

[8] K. De Jong, "An Analysis of the Behavior of a class of Genetic Adaptive Systems." Doctorate dissertation, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

[9] L.J. Eshelman & J.D. Schaffer, "Real coded genetic algorithms and interval schemata" In L. Durrel Whitely, Foundation of genetic algorithms 2 (pp. 187-202). San Mateo: Morgan Kaufman.

[10] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning". Addison Wisley New York, 1989.

[11] D.E. Goldberg, "Real-coded genetic algorithms. Virtual alphabets and blocking." Complex Systems, 5, 1991, 139-167.

[12] R. Hochman et al., "Using the genetic algorithm to build optimal neural networks for fault-prone module detection" In Proc. the 7th Int. Symposium on Software Reliability Engineering 1996.

[13] J.H. Holand, "Adaptation in Natural and Artificial Systems". Cambridge, Mass: MIT press, 1975.

[14] N. Karunanithi, D. Withtely and Y.K. Malaiya, "Prediction of Software Reliability using Connectionist Models," IEEE Trans Software Eng 1992.

[15] N. Karunanithi, D. Withtely and Y.K. Malaiya, "Using neural networks in reliability prediction," IEEE Software 1992.

[16] E.H.F. Leung, H.K. Lam, S.H. Ling, P.K.S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," IEEE Trans Neural Networks 2003.

[17] T. Liang, N. Afzel, "Evolutionary neural network modeling for software cumulative failure time prediction," ELSEVIER Reliab Eng & Sys Safety 2005.

[18] T. Liang, N. Afzel, "On-line prediction of software reliability using an evolutionary connectionists model," ELSEVIER the Journal of Sys & Software 2005.

[19] M.R. Lyu, "Software Reliability Engineering: A Roadmap". Future of Software Engineering (FOSE'07) IEEE CS Press 2007.

[20] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", Springer 1996.

[21] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", New-York : Springer, 1992.

[22] J.Y. Park, S.U. Lee, J.H. Park, "Neural Network Modeling for Software Reliability Prediction from Failure Time Data," J Electr Eng Inform Sc 1999.

[23] N.J. Radcliffe, "Equivalence class of genetic algorithms" Complex Systems, 1991, 5 (2), 183-205.

[24] J. D. Schaffer, R.. A.. Caruana, and L.J. Eshelman, "Using genetic search to explicit the emergent behavior of neural networks." In S. Forrest (Ed.), Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks (pp. 244-248). Cambridge, MA: MIT Press, 1991.