Simulated Annealing Neural Network for Software Failure Prediction

Mohamed Benaddy and Mohamed Wakrim

Ibnou Zohr University, Faculty of Sciences-EMMS, Agadir Morocco
m.benaddy@uiz.ac.ma, m.wakrim@uiz.ac.ma

Abstract

Various models for software reliability prediction were proposed by many researchers. In this work we present a hybrid approach based on the Neural Networks and Simulated Annealing. An adaptive simulated Annealing algorithm is used to optimize the mean square of the error produced by training the Neural Network, predicting software cumulative failure. To evaluate the predictive capability of the proposed approach various projects were used. A comparison between this approach and others is presented. Numerical results show that both the goodness-of-fit and the next-step-predictability of our proposed approach have greater accuracy in predicting software cumulative failure compared with other approaches.

Keywords: *Software Reliability, Neural Network, Simulated Annealing, Cumulative Software Failure*

1. Introduction

Software reliability is defined as the probability of failure-free software operations for a specified period of time in specified environment. Reliable software is a necessary component. Controlling faults in software requires that one can predict problems early enough to take preventive action.

Neural Networks approach has been used to evaluate the software reliability, it has proven to be a universal approximates for any non-linear continuous function with an arbitrary accuracy [6, 13, 18, 19]. Consequently, it has become an alternative method in software reliability modeling, evolution and prediction. Karunanithi, et. al., [10,11] were the first to propose using neural network approach in software reliability prediction. Aljahdali, et. al., [2, 17], Adnan, et. al., [1], Park, et. al., [9] and Liang, et. al., [18,19] have also made 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 [17]. But, the optimal training of neural network using conventional gradient-descent methods is complicated due to many attractors in the state space, and it is vulnerable to the problem of premature convergence [14]. Premature convergence occurs whenever the algorithm gets stuck in local minimum and value of the difference between the desired output and the computed output value is still higher than the allowed tolerance limit. That is why several solutions have been proposed to solve these problems encountered by the back propagation learning algorithm. We have proposed a real coded genetic algorithm to learn a Neural Network and we have obtained results better than the back propagation learning algorithm [4]. Leung, et. al., [13] have used an

improved genetic algorithm to perform the structure and parameters of the Neural Network. Liang, et. al., [18, 19] proposed a genetic algorithm optimizing the number of delayed input neurons and the number of neurons in the hidden layer of the neural network, predicting software reliability and software failure time. We have proposed other non parametric models based on auto regression order 4, 7 and 10, to fix the parameters of these models we have used a real coded genetic algorithm [3, 5].

In this work we propose a non parametric failure count model to predict cumulative software failure. A simulated annealing algorithm is used to learn the neural network predicting cumulative software failure.

2. Software Reliability Data set

The Software Reliability Dataset was compiled by John Musa of Bell Telephone Laboratories [15]. 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 used data from three different projects. They are Military (System Code: 40), Real Time Command & Control (System Code: 1) and Operating System (System Code: SS1C). The failure data were initially stored in arrays, ordered by day of occurrence so that it could be processed.

3. Neural Network

The neural Network used here is used by Aljahdali, et. al., [17]. It is a multi-layer feed-forward network. It consists of an input layer with four inputs, which are the observed faults four days before the current day, one hidden layer and one output layer. The hidden layer consists of two nonlinear neurons and two linear neurons. The output layer consists of one linear neuron which produces the predicted value of the fault. There is no direct connection between the network input and output. Connections occur only through the hidden layer. The structure of the adopted Neural Network is shown in Figure 1.

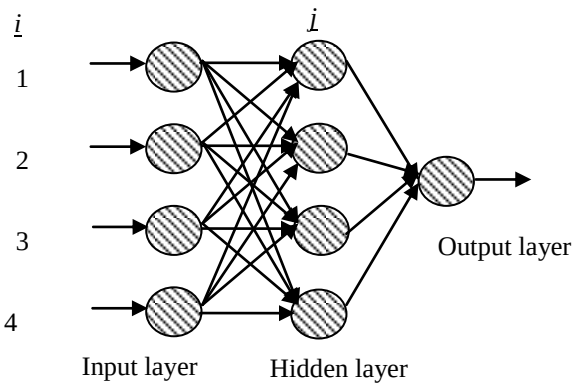

Figure 1. Feed-forward Neural Network

4. Simulated Annealing

Simulated annealing (SA) is a metaheuristic for the global optimization. The method was independently described by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983 [12] and by Vlado Černý in 1985 [7]. The method is an adaptation of the Metropolis, a Monte Carlo method to generate sample states of a thermodynamic system, invented by M.N. Rosenbluth in a paper by N. Metropolis et al. in 1953 [16].

The basic idea of the method is to generate a random configuration (trial point) iteratively through perturbation, and evaluate the objective function and the constraints after determining the state variables by using the simulator. The infeasible trial point result is rejected and a new one is generated. If the trial point is feasible and the objective function's value is good, then the point is accepted and the record for the best value is updated. If the trial point results in feasibility but the objective function is higher (for a minimization problem) or less (for a maximization problem) than the current best value, then the trial point is either accepted or rejected using the Metropolis criterion. The entire process is terminated after performing a fairly large number of trials or chains (iterations). The method uses temperature and other annealing parameters by trial and error to attain near-optimal solutions.

5. The Simulated Annealing to train Neural Network

The algorithm adopted in this work is given bellow:

1. Initialize a vector W_{opt} of the weights of the Neural Network with random values.
2. $F_{opt} = F(W_{opt})$
3. $T = T_{max}$
4. T_{min}, *iter = 0, maxIter*
5. *while*($T > T_{min}$) {
6. *while*(*iter*<*maxIter*){
7. $W_{neighbor}$ = *neighborOf* (W_{opt})
8. *if*($\Delta f = F(W_{neighbor}) - F(W_{opt}) < 0$) *then*{
9. *If*($F(W_{neighbor}) > F(W_{opt})$) *then*{
10. $W_{opt} = W_{neighbor}$
11. *actualize* F_{opt}
12. }
13. } *else*
14. *if*(*random*<$exp(-\Delta f / T)$) *then*
15. $W_{opt} = W_{neighbor}$
16. } $T = T*(1 - epsilon)$
17. }

A solution *W* consists of all the Neural Network weights. One element of a *W* represents a single weight value. In our case there are 4x4 weights for the hidden-layer, 4x1 biases, 4x1 weights and 1x1 biases for the output-layer. The length of the *W* is then $l = 4x4 + 4x1 + 4x1 + 1x1 = 25$. The weights and biases of the Neural Network are placed on a matrix with four lines and five columns as shown in Figure 2.

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & b_1 \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & b_2 \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & b_3 \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & b_4 \\ w_{5,1} & w_{5,2} & w_{5,3} & w_{5,4} & b_5 \end{pmatrix} \begin{matrix} \} \text{Hidden-layer} \\ \\ \\ \\ \} \text{Output-layer} \end{matrix}$$

Figure 2. The Representation of the Neural Network Weights (*W*)

The objective function: *F* should reflect the individual's performance, in the current problem. We have chosen *F=1/ (1+MSE)* as the objective function to maximize (e.g. minimize the error between the predicted and the observed value), where *MSE* is the mean squared error during training phase of the neural network defined in the equation bellow.

$$MSE = \frac{1}{n}\sum (x - y)^2 \quad \textbf{(1)}$$

Where *n* is the number of training faults used during the training process, *x* and *y* are the actual and the predicted output respectively during the learning process.

For the neighborhood generation we have implemented two methods inspired from the mutation genetic algorithm operator used in [3, 4, 5, 8]. The first method is called the one point update method (mutation) in which a randomly element in *W* is selected and modified with a parameter *β*, the formula used in this first generation method is defined by the following equation.

$$W_{i,j} = W_{i,j} + randomValueIn(\frac{-\beta}{2}, \frac{\beta}{2}) \quad \textbf{(2)}$$

The second method is called multipoint update method. In this method a random range of values are selected and updated randomly, by using the formula defined in the above equation.

To perfect the global search space we can use another generation method as described in the following equation.

$$W_{i,j} = W_{i,j} + randomValueIn(\frac{-2*\beta}{3}, \frac{4*\beta}{3}) \quad \textbf{(3)}$$

With *randomValueIn(a,b)* is a function that return a random value in the interval [a,b].

Experimental results show that the multipoint generation strategy is very efficient and converge faster than the one point. So we have adopted this method to generate the neighborhood solution candidate. To prove this choice, we plot in Figure 3 the values of the objective function using the two generation methods.

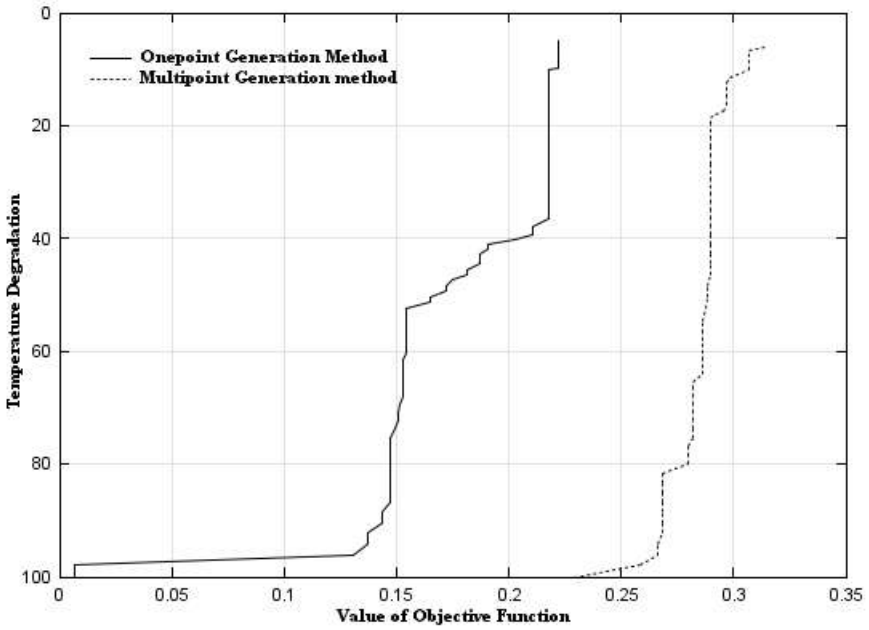

Figure 3. One Point versus Multipoint Generation Neighborhood Methods

6. Experimental Results

The initial weights were randomly chosen in the interval [0, 1]. For each project [15], a supervised training is performed to give the optimal parameters values of the algorithm. To prove the performance of our approach, a Normalize Root Square Error (NRMSE, Eq. 4) is computed and compared with these obtained in [2, 3, 4, 5].

$$NRMSE = \frac{1}{n}\sqrt{\frac{\sum_{i=1}^{n}(x(i)-y(i))^2}{\sum_{i=1}^{n}(x(i))^2}} \quad \textbf{(4)}$$

The Table 1 presents results obtained:

1. By Aljahdali et al. [2] with the same Neural Network in testing phase.

2. By using a Real Coded Genetic Algorithm (RCGA) for learning the same Neural Network [4].

3. *MSE* and *NRMSE* results obtained our proposed simulated annealing algorithm in training and testing phases are given in the following table.

Table 1. Comparison between our Present Approach and Other Approaches

		Project Name	Military	Real Time Control	Operating System
		Number of Faults	101	136	277
		Training Data	71	96	194
Aljahdali et al. [2]	Testing Phase	NRMSE	1.0755	0.5644	0.7714
RCGA [4]	Testing Phase	*MSE*	4.2277226	2.6617646	3.0758123
		NRMSE	4.7373e-5	3.1216e-4	1.5705e-5
The present approach	Training Phase	*MSE*	1.7323943	2.4329896	1.9329897
		NRMSE	1.60628E-4	5.869615E-4	3.318441E-5
	Testing Phase	*MSE*	2.8415842	2.4044118	2.4187725
		NRMSE	3.88384E-5	2.966878E-4	1.392739E-5

From these presented results, we observe that the training using simulated annealing algorithm is better than classical method. A little difference is observed compared with real coded genetic algorithm, but, the simulated annealing performance execution time is better than the real coded genetic algorithm because of the search space. That is in the genetic algorithm there is a population of solutions to perform, but, simulated annealing performs one solution candidate.

In Figure 4 to 11 we are plotting the training, the testing and the error difference results for various projects using the neural network trained by adaptive simulated annealing algorithm.

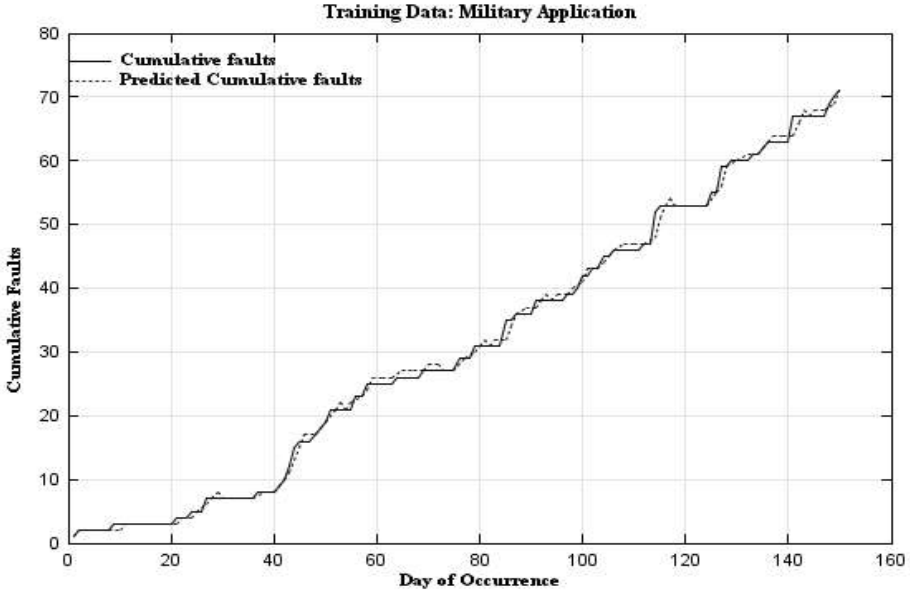

Figure 4. Actual and Predicted Cumulative Faults in Training Phase: Military Application

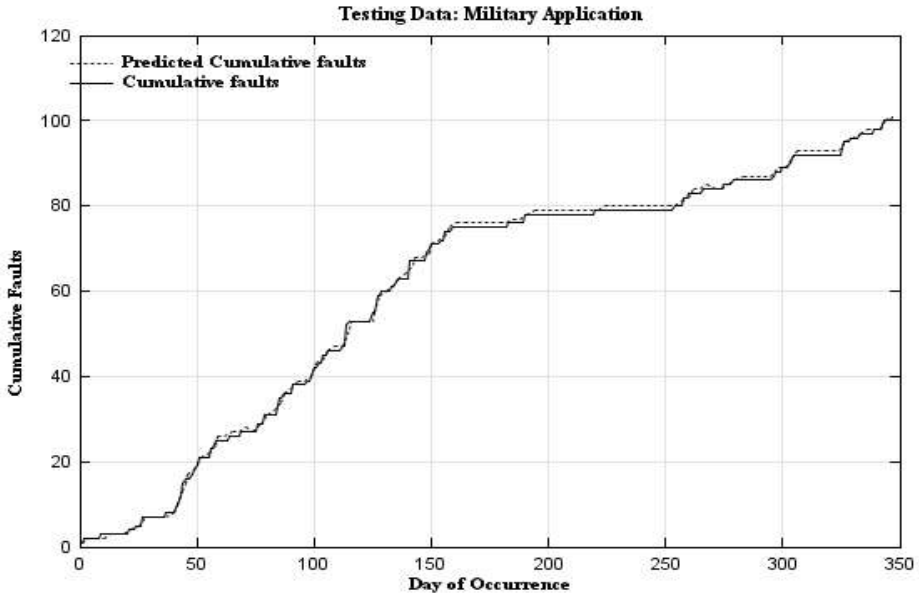

Figure 5. Actual and Predicted Cumulative Faults in Testing Phase: Military Application

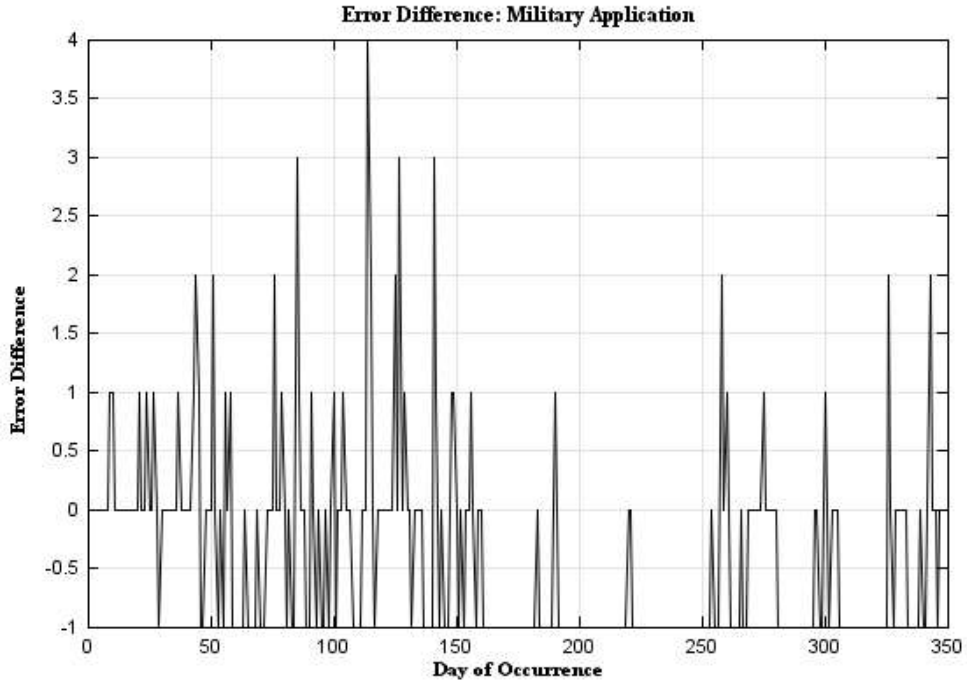

Figure 6. Prediction Error in Training and Testing Phases: Military Application

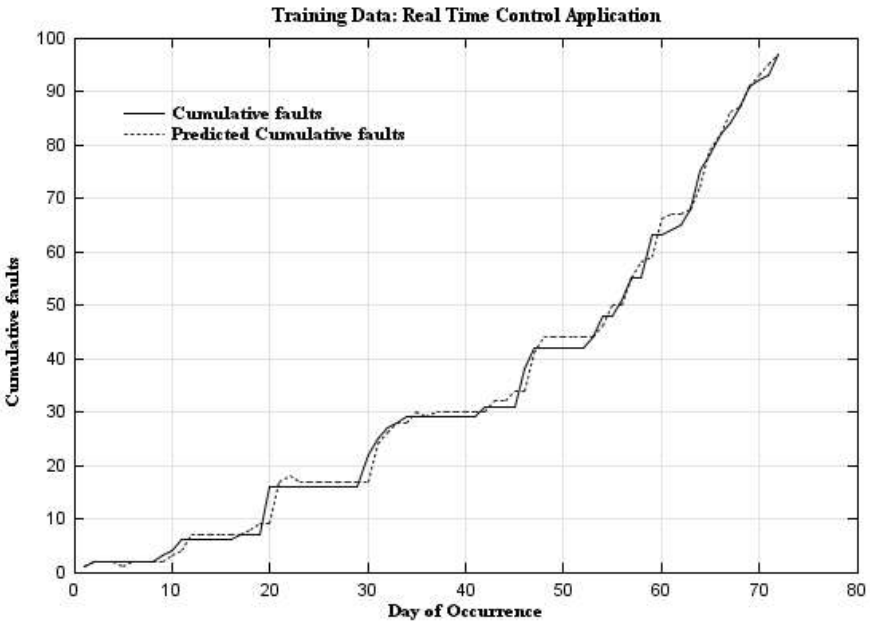

Figure 7. Actual and Predicted Cumulative Faults in Training Phase: Real Time Control

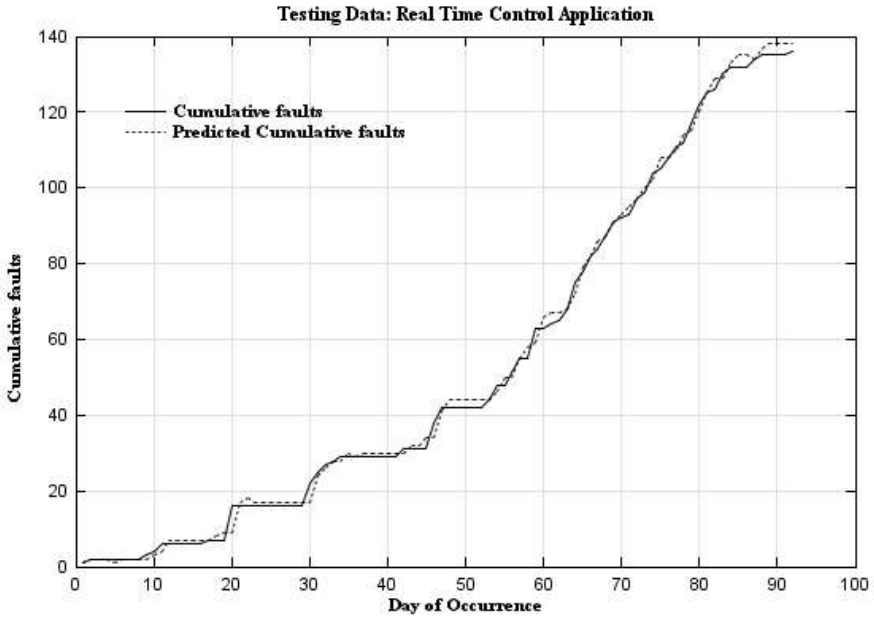

Figure 8. Actual and Predicted Cumulative Faults in Testing Phase: Real Time Control

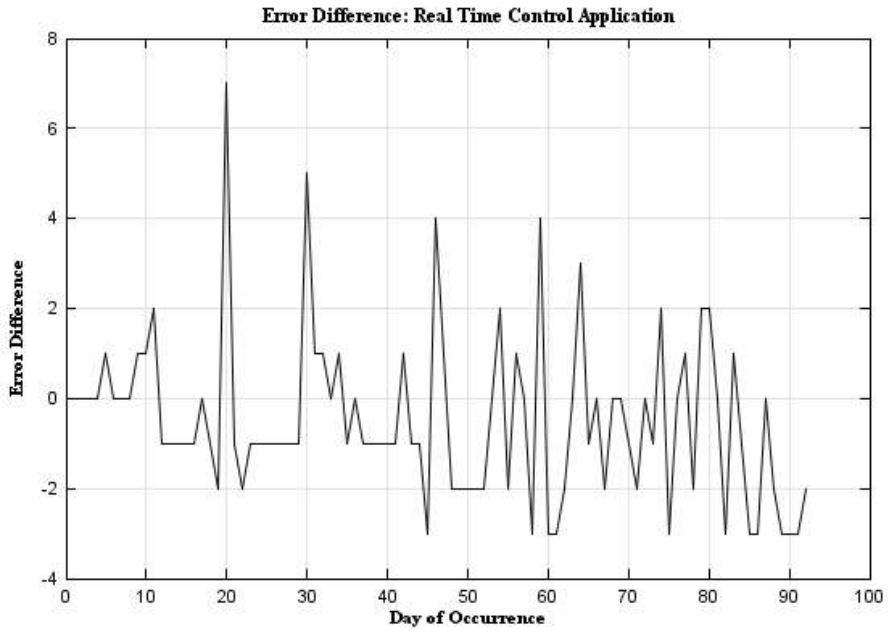

Figure 9. Prediction Error in Training and Testing Phases: Real Time Control

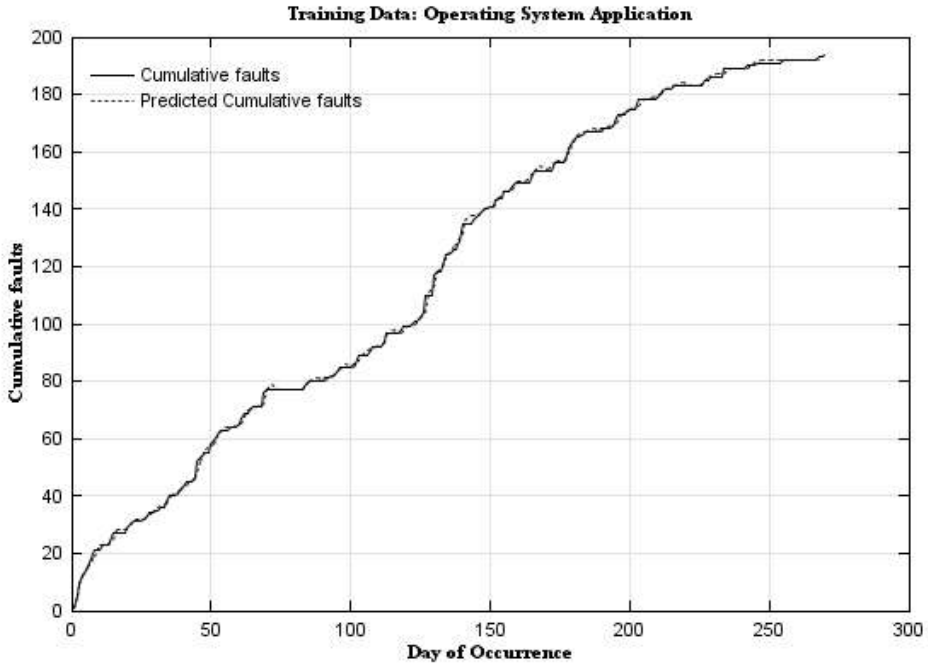

Figure 10. Actual and Predicted Cumulative Faults in Training Phase: Operating System

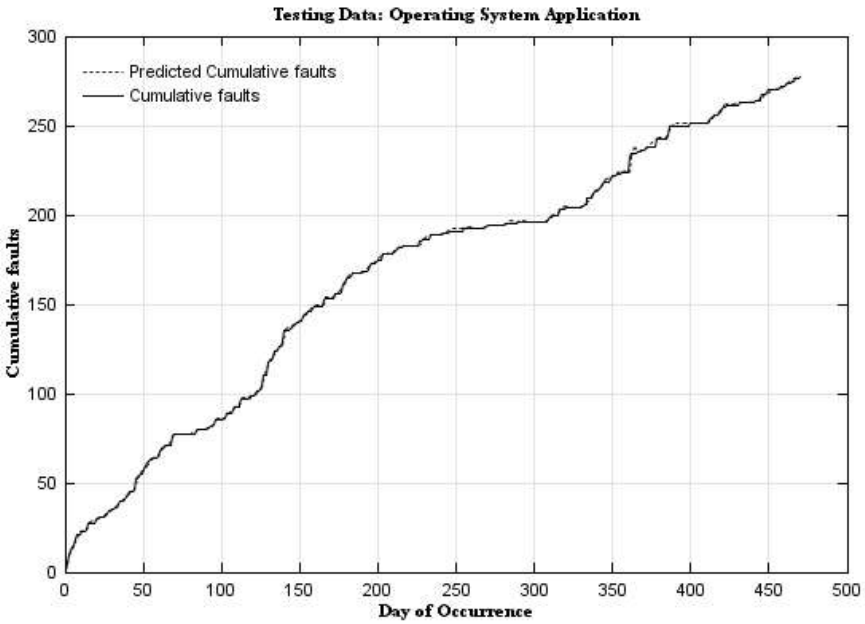

Figure 11. Actual and Predicted Cumulative Faults in Testing phase: Operating System

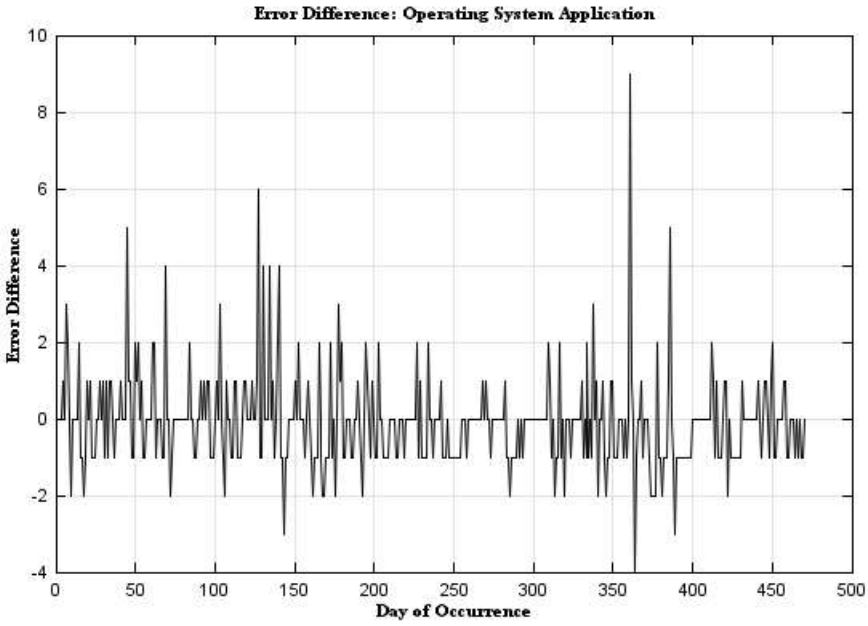

Figure 12. Prediction Error in Training and Testing Phases: Operating System

7. Conclusion

For predicting software failure, an adaptive Simulated Annealing is developed, mainly, to perform the training phase of the Neural Network process used to predict the software failure. This algorithm is used to minimize the mean square of the error between the predicted and the observed cumulative failure in software. For generating a neighborhood of the good solution. We have developed one point and multipoint strategies inspired from the mutation phase in the genetic algorithm.

By using this approach, good results are obtained, relating to the prediction of the software cumulative failure. From Figures 6, 9 and 12 we see that the difference of the error between the predicted and the real observed cumulative failure does not exceed 9 in the worst case for the operating system application. In fact, a better performance is obtained compared with the RCGA and the back propagation learning algorithm as shown in Table 1, for all tested projects. The performance in execution time of the proposed adaptive Simulated Annealing is better than the RCGA, because of the search space, which reduced from a population of solutions for the RCGA to one solution for the proposed Simulated Annealing. The proposed approach can be used for other software projects to predict the cumulative failure.

References

[1] W. A. Adnan and M. H. Yaacob, "An integrated neural-fuzzy system of software reliability prediction", In Proc. Conf. First Int Software Testing, Reliability and Quality Assurance, **(1994)**, pp. 154–158.

[2] S. H. Aljahdali, D. Rine and A. Sheta, "Prediction of software reliability: A comparison between regression and neural network non-parametric models", Computer Systems and Applications, ACS/IEEE International Conference on, 0:0470, **(2001)**.

[3] M. Benaddy, S. Aljahdali and M. Wakrim, "Evolutionary prediction for cumulative failure modeling: A comparative study", In Proc. Eighth Int Information Technology: New Generations (ITNG) Conf., **(2011)**, pp. 41–47.

[4] M. Benaddy, M. Wakrim and S. Aljahdali, "Evolutionary neural network prediction for cumulative failure modeling", In Proc. IEEE/ACS Int. Conf. Computer Systems and Applications AICCSA 2009, **(2009)**, pp. 179–184.

[5] M. Benaddy, M. Wakrim and S. Aljahdali, "Evolutionary regression prediction for software cumulative failure modeling: A comparative study", In Proc. Int. Conf. Multimedia Computing and Systems ICMCS '09, **(2009)**, pp. 286–292.

[6] K. -Y. Cai, L. Cai, W. -D. Wang, Z. -Y. Yu and D. Zhang, "On the neural network approach in software reliability modeling. J. Syst. Softw., vol. 58, **(2001)** August, pp. 47–62.

[7] V. Cerný, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm", Journal of Optimization Theory and Applications, vol. 45, **(1985)**, pp. 41–51, doi: 10.1007/BF00940812.

[8] J. H. Holland, "Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence", University of Michigan Press, **(1975)**.

[9] J. -H. Park J. -Y. Park and S. -U. Lee, "Neural network modeling for software reliability prediction from failure time data", Journal of Electrical Engineering and information Science, vol. 4, no. 4, **(1999)**, pp. 533–538.

[10] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Prediction of software reliability using connectionist models", IEEE Trans. Softw. Eng., vol. 18, **(1992)** July, pp. 563–574.

[11] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Using neural networks in reliability prediction", IEEE Softw., vol. 9, **(1992)** July, pp. 53–59.

[12] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by simulated annealing", Science, vol. 220, no. 4598, **(1983)**, pp. 671–680.

[13] F. H. F. Leung, H. K. Lam, S. H. Ling and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm", vol. 14, no. 1, **(2003)**, pp. 79–88.

[14] M. R. Lyu, "Handbook of Software Reliability Engineering", IEEE Computer Society Press and McGraw-Hill Book Company, **(1996)**.

[15] J. D. Musa, "Software Reliability Data", https://www.thedacs.com/databases/sled/swrel.php, Data & Analysis Center for Software, **(1980)**.

[16] M. N. Rosenbluth, A. H. Teller, N. Metropolis, A. W. Rosenbluth and E. Teller, "Equation of state calculations by fast computing machines", Journal of Chemical Physics, vol. 21, **(1953)**, pp. 1087–1092.

[17] K. A. Buragga and S. Aljahdali, "Evolutionary neural network prediction for software reliability modeling", In The 16th International Conference on Software Engineering and Data Engineering (SEDE-2007), **(2007)**.

[18] L. Tian and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction", Reliability Engineering & System Safety, vol. 87, no. 1, **(2005)**, pp. 45 – 51.

[19] L. Tian and A. Noore., "On-line prediction of software reliability using an evolutionary connectionist model", Journal of Systems and Software, vol. 77, no. 2, **(2005)**, pp. 173 – 180.

Authors

Mohamed BENADDY

Is currently a Phd student At Ibn Zohr University, Faculty of Sciences Agadir. He received his DESA in Computer Sciences and Applied Mathematics from Ibn Zohr University. His research interests are in the Software and System Reliability Engineering, Prediction and optimization.

Mohamed WAKRIM

Professor of Applied Mathematics and computer science at Ibnou Zohr University. Director of the Laboratory of Mathematical Modeling and Simulation.