IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 2, Ver. I (Mar.-Apr. 2017), PP 57-64
www.iosrjournals.org

A Development Model for Predicting Software Reliability Using Ant Colony Optimization Technique for Change Oriented Software Process

[1]D. Hema Latha , [2]Prof. P. Premchand
[1]*Research Scholar, Dept of Computer Science, Rayalaseema University, Kurnool, Andhra Pradesh, India*
[2]*Professor, Dept of Computer Science and Engineering, UCE, Osmania University, Hyderabad, TS, India*

Abstract: *Software reliability prediction is very challenging in maintenance phase as well as in the starting phases of software development. In the past few years many software reliability models have been proposed for assessing reliability of software but developing accurate reliability prediction models is difficult due to the recurrent or frequent changes in data in the domain of software engineering. As a result, the software reliability prediction models built on one dataset show a significant decrease in their accuracy when they are used with new data. The objective of this paper is to introduce a new approach that optimizes the accuracy of software reliability predictive models when used with raw data. Ant Colony Optimization Technique (ACOT) is proposed to predict software reliability based on data collected from literature. An ant colony system by combining Travelling Sales Problem (TSP) algorithm has been used, which has been changed by implementing different algorithms and extra functionality, in an attempt to achieve better software reliability results with new data for change oriented systems. The intellectual behavior of the ant colony framework by means of a colony of cooperating artificial ants are resulting in very promising results. The method is validated with real dataset using Mean Time to Failure (MTTF) and Mean Time Between Failure (MTBF).*

Keywords: *Software Reliability, Bio-inspired Computing, Ant Colony Optimization technique*

I. Introduction

As the past decades have seen the computerization of all the functionalities in all the fields turn out to be supplementary multifaceted and therefore, there is a constant demand for discovering innovative well organized methodologies to software development and preservation. There is a prerequisite of the enormous scope of effort, time and currency to arrange and build up any feasible software apart from the human resource and their organization. For outstanding rising competition, today's profitable conditions have become very dynamic. Corporate industries require proceeding extremely fast to unstable needs of the market. Hence, software engineering which emphasizes with all these regions has become an individual study from researchers. The software crisis is defined as mismatch between what the software can deliver and the capacities of computer systems, as well as the expectations of their users and where software problems cause the system tasks to be delayed, expensive, and/or not amenable to the user's desires. Apart from software can be developed to meet the various stages of reliability, security, portability, usability, effective cost and response time.

Developing awfully trustworthy software from the user's perspective is a demanding profession for all software engineers. However, Software Reliability is a significant aspect influencing system reliability. The following four practical aspects which are related to achieving reliable software systems and these aspects furthermore be treated as four fault Lifecycle techniques:

1) Fault avoidance: to avoid, by building, error existence. 2) Fault elimination: to identify, by confirmation and proof, the presence of faults and eliminate them. 3) Fault tolerance: to specify, by redundancy, facility conforming to the requirement in spite of faults having rising. 4) Fault/failure Predicting to estimate: by the assessment, the occurrence of faults and consequences of failures.

Software reliability is the probability that software will not cause the failure of a system for a particular point in time underneath particular circumstances. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. According to ANSI, Software Reliability is defined as: "the probability of failure free software operation for a particular period of time in a particular atmosphere". Software reliability evaluation is significant to evaluate and forecast the trustworthiness and performance of software systems. Reliability representation is a crucial ingredient of the reliability evaluation procedure and it also validate whether a product meets up its reliability objective and is ready to distribute. The fundamental intention of most of software reliability models is making them to understand distinctiveness and reasons to fail software, and try to enumerate software reliability. The current article lay emphasis on about a bio inspired computing technique Swarm Intelligence known as the Ant Colony Optimization Technique to predict software reliability. The anticipated method is employed into a TSP problem with software failure datasets to predict software reliability and the results of our approach are reported. And, thus, the focus of the

discussion to be presented here is an ACO for discrete optimization that has been used to predict software reliability using the Travelling Sales Person Problem where failure data is given as input and the result is calculated through Mean Time to Failure (MTTF) and Mean Time Between Failure (MTBF) to predict the reliability.

II. Methodology

A Bio Inspired Computing

Natural computing [22] is a term presented to comprise three classes of methods: (1) those that take motivation from nature for the development of novel problem-solving techniques; (2) those that are constructed with the use of computers to synthesize natural facts; and (3) those that employ natural resources (e.g., molecules) to compute. The main areas of research that comprise these three branches are the artificial neural networks, evolutionary algorithms, swarm intelligence, artificial immune systems, fractal geometry, artificial life, DNA computing, and quantum computing, among others. Bio-inspired Computing is the combination of computational aptitude and collective intelligence. These computational approaches are used to resolve multifaceted problems, and developed after design principles confronted in natural / biological systems, and tend to be adaptive, responsive, and distributed. The aim of bio-inspired computing is to develop computational tools with enhanced strength, scalability, flexibility and which can interact more efficiently with humans. It can offer biologists, for example, with an IT-oriented concept for looking at how cells compute or process information, or help computer scientists build algorithms based on natural systems, such as evolutionary and genetic algorithms. Biocomputing [23] has the potential to be a very powerful tool. The association of bio-inspired computing are artificial neural networks, evolutionary algorithms, swarm intelligence, artificial immune systems, fractal geometry, artificial life, DNA computing and quantum computing.

B Ant Colony Optimization Technique

Ant Colony [24-27] is one of the techniques of bio inspired computing. The main concept of this is technique is that the self-organizing rules which allow the highly synchronized behavior of real ants can be utilized to manage populations of artificial agents that cooperate to solve computational problems. Various distinctive attributes of the behavior of ant colonies have inspired different kinds of ant algorithms. Examples are foraging, distribution of labor, issue sorting, and cooperative transport. Ants coordinate their activities via stigmergy, a form of implicit interaction mediated by changes in the environment. For example, a foraging ant deposit a chemical on the ground which raises the probability that other ant will follow the same path. Biologists have presented that many colony-level behaviors witnessed in social insects can be described through relatively simple models in which only stigmergic communication is present. In other words, biologists have shown that it is often sufficient to consider stigmergic, indirect communication to explain how social insects can attain self-organization. The notion behind ant algorithms is to use a form of artificial stigmergy to coordinate societies of artificial agents. One of the most effective examples of ant algorithms is known as ‘‘ant colony optimization’’, or ACO. ACO is motivated by the foraging behavior of ant colonies, and targets discrete optimization problems. The ants may deposit a pheromone on the ground while returning back to their nests. The ants follow with high probability pheromone trails their sense on the ground.

Each Ant evaluates the next move to another vertex based on Gambardella et al., [28, 29],

$$p_{ij}^{k} = \begin{cases} \dfrac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}} & \text{if j} \in \text{ allowed} \quad \text{k} \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

$p^{k}{}_{ij}$ *is the probability for a worker K to move to vertex “ij”*

ij τ *is the amount of pheromone deposited on edge to “ij”*

η is the inverted distance, describes how fast ants select their path.

The tour cost of each ant is given by *di j* the tour cost from the city *i* to city *j* (edge weight) is calculated and hence the shortest path is found. This is applied to the Travelling Sales Person Problem and optimized solutions are obtained using

$$d_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \qquad (2)$$

The amount of pheromone deposited by each ant is given by

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij} \quad (3)$$

The value of $\Delta\tau_{ij}$ is calculated using

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \quad (4)$$

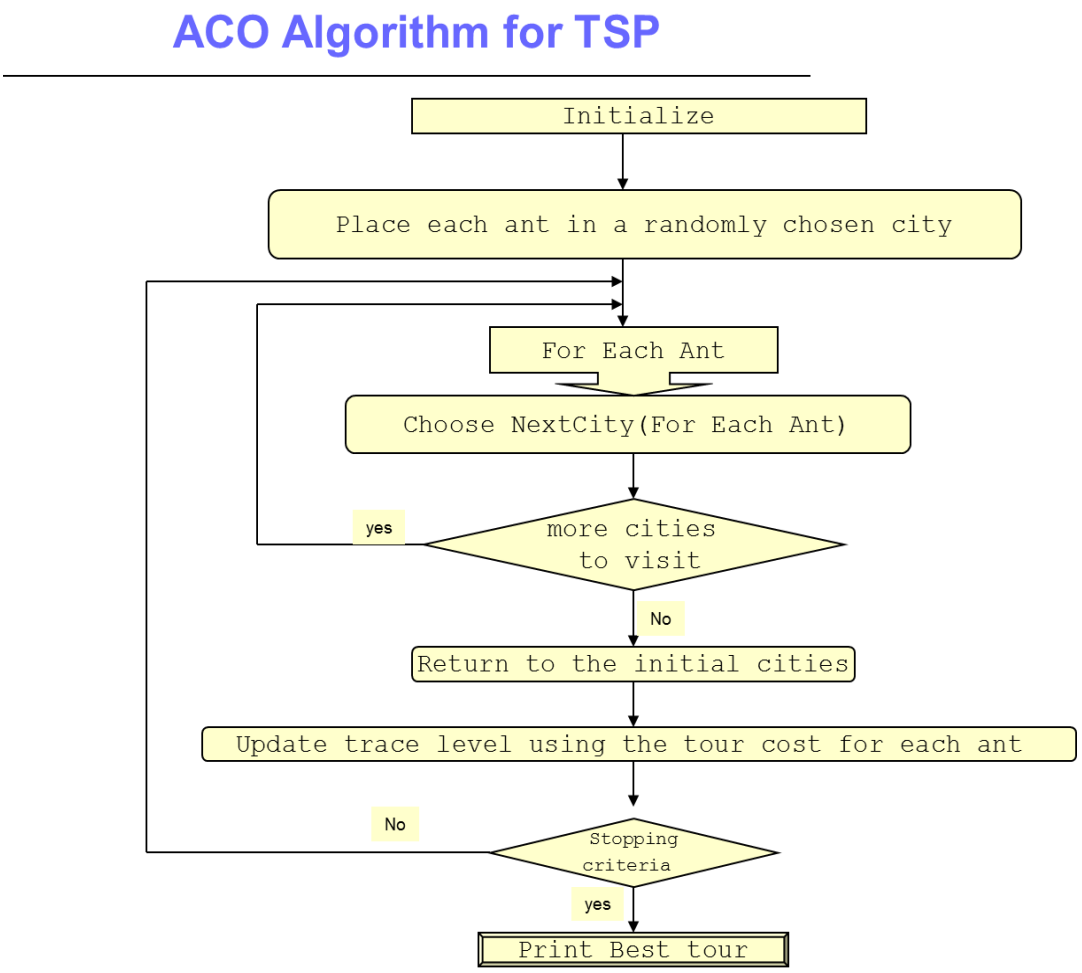
$\Delta\tau_{i,j}^{k}$ is calculated using

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} \; if\,(i,j) \in \text{bestTour} \\ 0 \qquad \text{otherwise} \end{cases} \quad (5)$$

C Algorithm

The ACO algorithm [30] which has been proposed based on the study that real ants are skilled in finding the shortest path from a food source to the nest without using visual signals. From the originating point the ants start the tour selecting randomly any path.

1. Set the initial parameters.
2. Initialize pheromone trails.
3. Calculate the maximum specific ways in which the ants can travel.
4. Loop //iteration
5. Each ant is positioned at a given node randomly selecting the node according to some distribution strategy (each node has at least one ant)
6. For k=1 to m do //steps in a loop
7. The first step: move each ant in a different route
8. Repeat //till all the nodes are visited
9. Select node j to be visited next // the next node must not be an already visited node
10. Apply local updating rule
11. Until ant k has completed a tour
12. End for
13. Apply sub tour that is sub Local search // to improve tour
14. Apply global updating rule
15. Compute entropy value of current pheromone trails
16. Update the heuristic parameter
17. Until End_condition
18. End



The flow chart of the ACO algorithm is presented in Fig. 1.

III. Experimental Design

In our experiment, time series forecasting model is employed to predict software reliability which has only one dependent variable and no explanatory variables in strict sense. In this paper, the software failure data obtained from Musa [21] data sets is employed in this study. It is used to demonstrate the forecasting performance of Ant colony optimization techniques. The data contains 101 observations of the pair (*t*, *Yt*) pertaining to software failure. Here *Yt* represents the time to failure of the software after the *tth* modification has been made. We created five datasets viz.lag # 1,2,3,4 and 5 in view of the foregoing discussion on generating lagged data sets out of a time series. The experiments are performed by dividing the data into training and test set in the ratio of 80:20.

Simulation Results

Reliability is calculated as:

- **Reliability** = MTBF / (1+MTBF)

MTBF : Mean Time Between Failure

$$MTTF = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt$$

MTTF : Mean Time to Failure

To calculate MTTF in MS-Excel the following formula is used:

MTTF = f(x+dx)-f(x)/dx

Or

If the relaibility is checked on hourly basis the following formula can be used:

MTTF = 1/ failure rate

Mean Time to Failure

- Measures time between observable system failures

For example, assume you tested 3 identical systems starting from time 0 until all of them failed. The first system failed at 10 hours, the second failed at 12 hours and the third failed at 13 hours. The MTTF is the average of the three failure times, which is 11.6667 hours. If these three failures are random samples from a population and the failure times of this population follow a distribution with a probability density function (*pdf*) of $f(t)$, then the population MTTF can be mathematically calculated by:

Mean Time between Failures

The points on the plot are the observed cumulative MTBFs. These values are calculated by the following equation:

$$MTBF(t) = \frac{t}{N(t)}$$

where:

- t is the cumulative operating time.
- $N(t)$ is the observed number of failures by time t.

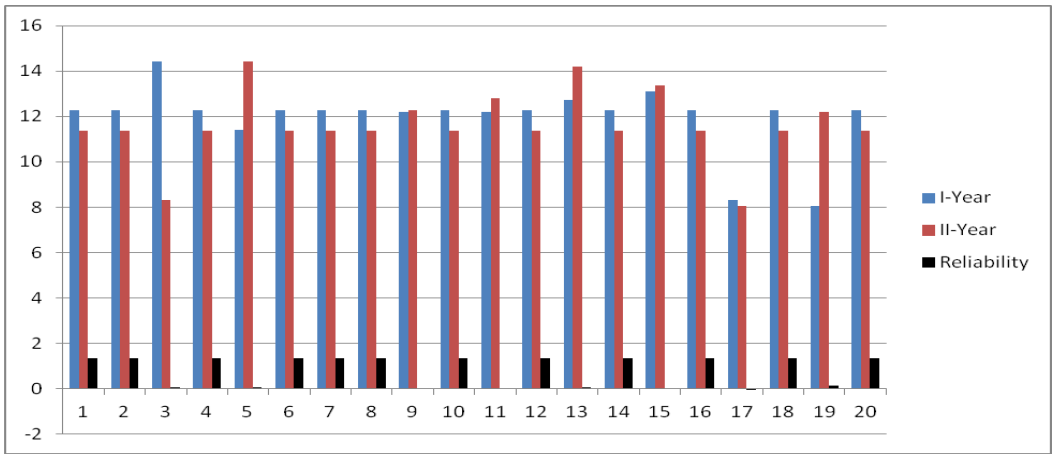
Or

MTBF = 1/MTTF

Reliability Validations:

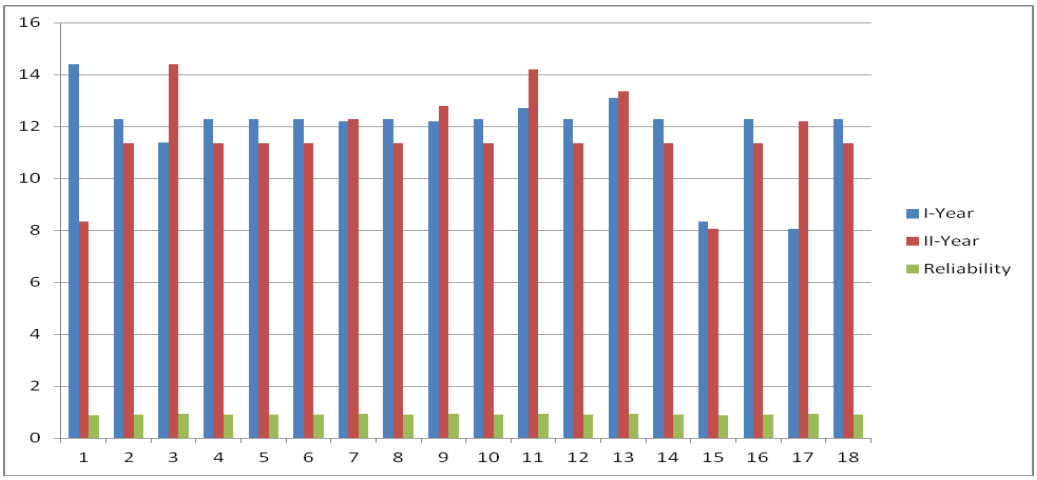
20 Projects Failure Rates (Data Set –I)						
I-Year	**II-Year**	**III-Year**	**IV-Year**	**MTTF**	**MTBF**	**Reliability**
12.2793	11.3667	12.2021	12.7831	-0.263392571	-3.796614299	1.35757523
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
14.4113	8.3333	14.192	11.3704	18.11824044	0.055192998	0.052306069
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
11.3923	14.4113	12.0907	13.0977	12.14683428	0.082325977	0.076063939
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
12.2793	11.3667	13.0977	13.368	-0.263392571	-3.796614299	1.35757523
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
12.2021	12.2793	9.7981	12.0907	170.1831902	0.005876021	0.005841695
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
12.2021	12.7831	12.0907	13.0977	32.62299329	0.030653226	0.029741552
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
12.7206	14.192	9.7981	12.0907	19.89443583	0.050265311	0.047859632
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
13.0977	13.368	13.368	12.7206	61.28355982	0.01631759	0.016055601
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
8.3333	8.0709	12.0907	13.0977	-23.16230305	-0.0431736	-0.045121664
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523
8.0709	12.2021	9.7981	12.0907	5.89334543	0.16968291	0.145067444
12.2793	11.3667	12.2793	11.3667	-0.263392571	-3.796614299	1.35757523

Graph for Data Set –I		
I-Year	**II-Year**	**Reliability**
12.2793	11.3667	1.35757523
12.2793	11.3667	1.35757523
14.4113	8.3333	0.052306069
12.2793	11.3667	1.35757523
11.3923	14.4113	0.076063939
12.2793	11.3667	1.35757523
12.2793	11.3667	1.35757523
12.2793	11.3667	1.35757523
12.2021	12.2793	0.005841695
12.2793	11.3667	1.35757523
12.2021	12.7831	0.029741552
12.2793	11.3667	1.35757523
12.7206	14.192	0.047859632
12.2793	11.3667	1.35757523
13.0977	13.368	0.016055601
12.2793	11.3667	1.35757523
8.3333	8.0709	-0.045121664
12.2793	11.3667	1.35757523
8.0709	12.2021	0.145067444
12.2793	11.3667	1.35757523



20 Projects Failure Rates (Data Set -II)						
I-Year	**II-Year**	**III-Year**	**IV-Year**	**MTTF**	**MTBF**	**Reliability**
14.4113	8.3333	14.192	11.3704	18.11824	0.055193	0.052306
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
11.3923	14.4113	12.0907	13.0977	12.14683	0.082326	0.076064
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
12.2793	11.3667	13.0977	13.368	-0.26339	-3.79661	1.357575
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
12.2021	12.2793	9.7981	12.0907	170.1832	0.005876	0.005842
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
12.2021	12.7831	12.0907	13.0977	32.62299	0.030653	0.029742
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
12.7206	14.192	9.7981	12.0907	19.89444	0.050265	0.04786
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
13.0977	13.368	13.368	12.7206	61.28356	0.016318	0.016056
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
8.3333	8.0709	12.0907	13.0977	-23.1623	-0.04317	-0.04512
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575
8.0709	12.2021	9.7981	12.0907	5.893345	0.169683	0.145067
12.2793	11.3667	12.2793	11.3667	-0.26339	-3.79661	1.357575

Graph for Data Set –II		
I-Year	**II-Year**	**Reliability**
14.4113	8.3333	0.89285676
12.2793	11.3667	0.919137684
11.3923	14.4113	0.935112547
12.2793	11.3667	0.919137684
12.2793	11.3667	0.919137684
12.2793	11.3667	0.919137684
12.2021	12.2793	0.924694826
12.2793	11.3667	0.919137684
12.2021	12.7831	0.927447381
12.2793	11.3667	0.919137684
12.7206	14.192	0.934175882
12.2793	11.3667	0.919137684
13.0977	13.368	0.930400891
12.2793	11.3667	0.919137684
8.3333	8.0709	0.889757356
12.2793	11.3667	0.919137684
8.0709	12.2021	0.924254475
12.2793	11.3667	0.919137684



IV. Conclusions

In this paper, ACOT using software reliability datasets can be employed. The performance of ACOT that of BPNN, TANN, PSN, MARS, GRNN, MLR, TreeNet, DENFIS, Morlet based WNN and Gaussian based WNN can be compared. It is observed that the performance of ACOT is better when compared with other techniques when combined with error checking computational method. Thus, ACOT holds a very good promise in the field of software reliability, Reliability calculated using Mean Time Between Failure (MTBF) and Mean Time To Failure (MTTF) gave good results.

References

[1]. R. K. Mohanty, V. Ravi, and M. R. Patra, "Hybrid intelligent Systems for predicting Software reliability," Elsevier, Applied Soft Computing, vol. 13, No. 1, pp. 189-200, 2013.
[2]. R. K. Mohanty, V. Ravi, and M. R. Patra, "Application of Machine learning Techniques to Predict software reliability," International Journal of Applied Evolutionary Computation, vol. 1, No.3, pp. 70-86, 2010.
[3]. K. Cai, C. Yuan, and M. L. Zhang, "A critical review on software reliability modeling," Reliability engineering and Systems Safety, vol. 32, pp. 357-371, 1991.
[4]. T. Dohi, Y, Nishio, and S. Osaki, "Optional software release scheduling based on artificial neural networks," Annals of Software engineering, vol. 8, pp. 167-185, 1999.
[5]. N. Karunanithi, Y. K. Malaiya, and D. Whitley, "The scaling problem in neural networks for software reliability prediction," In Proceedings of the Third International IEEE Symposium of Software Reliability Engineering, Los Alamitos, CA, pp. 76- 82, 1992a.
[6]. N. Karunanithi, D. Whitley, and Y.K. Malaiya, "Prediction of software reliability using connectionist models," IEEE Transactions on Software Engineering, vol. 18, pp. 563-574, 1992b.
[7]. T. M. Khoshgoftaar, D. L. Lanning, and A. S. Pandya, "A neural network modeling for detection of high-risk program," In Proceedings of the Fourth IEEE International Symposium on Software reliability Engineering, Denver, Colorado, pp. 302-309, 1993.
[8]. T. M. Khoshgoftaar, and P. Rebours, "Noise elimination with partitioning filter for software quality estimation," International Journal of Computer Application in Technology, vol. 27, pp. 246-258, 2003.
[9]. T. M. Khoshgoftaar, A.S. Pandya, and H.B. More, "A neural network approach for predicting software development faults," In Proceedings of the third IEEE International Symposium on Software Reliability Engineering, Los Aiamitos, CA, pp. 83- 89, 1992.
[10]. T. M. Khoshgoftaar, E. B. Allen, and J.P. Hudepohl, S.J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system," IEEE Transactions on Neural Networks, vol. 8, No. 4, pp. 902-909, 1997.
[11]. T. M. Khoshgoftaar, E.B. Allen, W. D. Jones, and J. P. Hudepohl, "Classification –Tree models of software quality over multiple releases," IEEE Transactions on Reliability, vol. 49, No. 1, pp. 4-11, 2000.
[12]. J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection". Cambridge, MA: The MIT Press, 1992.
[13]. J. D. Musa, Iannino, A., and K. Okumoto, "Software Reliability, Measurement, Prediction and Application," McGraw-Hill, New York, 1987.
[14]. J. D. Musa,"Software reliability data," IEEE Computer Society- Repository, 1979.
[15]. N. Karunanithi, D. Whitley, and Y.K. Malaiya, "Prediction of software reliability using neural networks," International Symposium on Software Reliability, pp. 124-130, 1991.
[16]. T.M. Khoshgoftaar, and R.M. Szabo, "Predicting software quality, during testing using neural network models: A comparative study," International Journal of Reliability, Quality and Safety Engineering, vol. 1, pp. 303-319, 1994.
[17]. L. Tian, and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," Reliability Engineering and System Safety, vol. 87, pp. 45-51, 2005b.
[18]. N. Rajkiran, and V. Ravi. "Software Reliability prediction by soft computing technique," The Journal of Systems and Software, vol. 81, No.4, pp. 576-583, 2007.
[19]. N. Rajkiran, and V. Ravi, "Software Reliability prediction using wavelet Neural Networks," International Conference on Computational Intelligence and Multimedia Application (ICCIMA, 2007), pp. 195-197, 2007
[20]. V. Ravi, N. J. Chauhan, and N. RajKiran., "Software reliability prediction using intelligent techniques: Application to operational risk prediction in Firms," International Journal of Computational Intelligence and Applications, vol.8, No. 2, pp. 181-194, 2009
[21]. E. Bonabeau, M. Dorigo, and G. Théraulaz, "Inspiration for optimization from social insect behavior," Nature, pp. 39–42, 2000.

[22]. A. Coloni, M. Dorigo, and V. Maniezzo, "Ant system: Optimization by a colony of cooperating agent," IEEE Trans. Systems Man and Cybemetics-Part B: Cybemetics, vol. 26, No.1, pp. 29-41, 1996.
[23]. M. Dorigo and G. Di Caro, "The Ant Colony OptimizationMeta-Heuristic," In D. Corne, M. Dorigo and F. Glover, editors, New Ideas in Optimization, McGraw-Hill, pp. 11-32, 1999.
[24]. M. Dorigo, and L. M. Gambardella, "Ant colonies for the traveling salesman problem", BioSystems 43, pp. 73–81, 1997.
[25]. M. Dorigo, and. L. M. Gambardella, "Ant Colony System: A cooperative learning approach to the traveling salesman problem," IEEE Transactions on Evolutionary Computation, vol. 1, No.1, pp.53–66, 1997.
[26]. M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: An autocatalytic optimizing process," Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.
[27]. L. M. Gambardella, E. D. Taillard, and M. Dorigo, "Ant colonies for the quadratic assignment problem," Journal of the Operational Research Society, vol.50, No.2, pp.167–176, 1999.
[28]. V. Maniezzo, and A. Colorni, "The Ant System applied to the quadratic assignment problem," IEEE Transactions on Data and Knowledge Engineering, Vol.11, No. 5, pp. 769– 778, 1999.
[29]. L. M. Gambardella, E. D. Taillard, and M. Dorigo, "Ant Colonies for the Quadratic Assignment Problem," Journal of the Operational Research Society, The Journal of the Operational Research Society, vol. 50, No.2, pp. 167-176, 1999.
[30]. L. M. Gambardella, E. D. Taillard, and G. Agazzi, "MACSVRPTW: A multiple ant colony system for vehicle routing problems with time windows," In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, pp. 63–76. Hill, London, UK, 1999.
[31]. R. Poli, and W.B. Langdon, J.R. Koza, "A field guide to Genetic Programming," ISBN: 978-1-4092-0073-4, publisher- Lulu.com , United Kingdom, 2008.
[32]. P. F. Pai, and W. C. Hong, "Software reliability forecasting by support vector machine with simulated annealing algorithms," Journal of System and Software, vol.79, No.6, pp. 747-755, 2006.