International Journal of Multimedia and Ubiquitous Engineering
Vol.11, No.11 (2016), pp.325-334
http://dx.doi.org/10.14257/ijmue.2016.11.11.29

A Bayesian Network Approach to Launch Vehicle Software Failure Prediction

Tinggui Yan[1], Xiaoqian Chen[1], Shipeng Li[2], Li Ma[2], Jian Bai[2] and Zhifang Yang[2]*

[1]College of Aerospace Science and Engineering, National University of Defense Technology, 410073, Changsha, China
[2]Beijing Institute of Astronautical System Engineering, China Academy of Launch Vehicle Technology, 100076, Beijing, China
qwyzf@163.com

Abstract

Launch Vehicle software has played an important role in Launch Vehicle system. However, the reliability assessment of Launch Vehicle software is still a hard problem due to the complexity of Launch Vehicle software. Failure prediction can be an effective approach of Launch Vehicle reliability evaluation, whereas failure prediction of software has not yet been fully explored. In this paper, a Markov Bayesian Network model for Launch Vehicle software failure prediction (MBNG) is proposed. In MBNG, unique features of Launch Vehicle software is considered as an important part in the modeling process, which improves the effectiveness of this novel model. Experiments are conducted to verify the effectiveness of MBNG model and compare its performance with classic models.

Keywords*: Failure prediction, Launch Vehicle software, Bayesian network, Markov*

1. Introduction

With the rapid development of computer hardware and Internet, Launch Vehicle software has been adopted widely in Launch Vehicle systems. In order to guarantee the quality and user experience of Launch Vehicle software, it is necessary to design test plans carefully and rigorously. However, the defects in some software are usually hard to eliminate. It is much less economical to pay too much effort on detect and remove the remaining defects. In this situation, it is important for testing engineering to predict the software failure in a more accurate way.

Regarding software failure prediction, a number of models have been proposed [1-4]. In particular, the popular models are based on classic probability theory [1-3], and there do exist some other models which improves these popular models. The underlying hinders of their application in practice is their restrictive assumptions, for example, mathematical tractability, data completeness and perfect defect removal. These restrictive assumptions have limited the development and applications of these probability models and thus researchers have investigated other approaches to predict software failure. One method is to release the assumptions by making no distribution assumptions on the failure process [5], but this advantage is offset by the additional noise.

In addition to the probability models, some non-probability models, i.g., neural network, have been applied [3,6-8].Recent researches have shown some advantages of neural network in predictive capabilities than the traditional NHPP model [9]. However, there are still some drawbacks by using this approach. Firstly, due to the randomness of the network structure, it is difficult to represent the interrelationships between the input variables. Secondly, it is necessary to ensure the integrality of the neural network. The neural network cannot process effectively with absent data of some input nodes. In this

ISSN: 1975-0080 IJMUE

situation, Bai *et al*. proposed a Markov Bayesian network approach for software failure prediction [4]. In this model, the restrictive assumptions are released because the rationality and accuracy of these assumptions are hard to verify. As prediction of software failures need to take into account of both previous failure data and the structure &status of software under test, Bayesian network is utilized to capture the underlying relationships between the identified factors and the potential ones.

Bayesian network can deal with such complex problems, such as software failure predictions in large avionics systems, by using Bayesian network to consider both prior information and available expert experience. Bayesian networks can explore many relative factors of the system and express their relationships efficiently. The combination of precise probability distributions and expert prior information also make it suitable for uncertain environment. Due to these valuable properties, Bayesian networks have been increasingly applied in many fields [10-14]. In particular, it has been found to use in reliability only in the last seven years [15]. A framework combining the diverse sources of evidence is provided by [16-17] to assess system dependability. It has also been applied in software reliability research. It has been used to predict software reliability in the early phases of the development by incorporating information ahead of testing [18] and to develop a causal model for software defect rates prediction [19]. It has also been combined with fault trees for reliability analysis [20], and used for the certification of COTS (Commercial Off-The-Shelf) software reliability [21].

However, there are several challenges when applying the Markov Bayesian network model for Launch Vehicle software failure prediction. Launch Vehicle software has a specific feature, that is, the test results are related on both the external hardware environment. In previous Markov Bayesian network model, only the interactions between the specific software and its external environment are considered and software status is excluded. Therefore, how to improve the previous Markov Bayesian network model for software failure prediction should be the major research question in this paper. Secondly, in our previous work, only one program is used to validate the effectiveness of proposed Markov Bayesian network model. The applicability of the model to more Launch Vehicle software is another research problem.

In this paper, a Markov Bayesian network model for Launch Vehicle software prediction (MBNG) is proposed. In this new model, both the software status and the external hardware environment are considered in the software failure prediction. In this case, this new model can better reflect the characteristics of Launch Vehicle software testing and thus make more reasonable predictions. Specifically, the software status is a mutable structure that aims to provide a more flexible description of the Launch Vehicle software. Since there is a large variety of the testing environment, operational profile, and test suite partition, the adoption of such a mutable structure could be useful when applying to different Launch Vehicle software. In order to validate the effectiveness of MBNG, experiments with real life software subjects are conducted and the results are analyzed to assess the model.

The rest of this paper is structured as follows. Section 2 describes the model of MBNG. In Section 3, detailed modeling and deduction process is introduced. Experiments are conducted to validate the MBNG model in Section 4. The discussion and future work are concluded in Section 5.

2. Bayesian Network Model

The basic form of a Bayesian network is stated as follows: a Bayesian network is a directed probability graph, connecting the relative variables with arcs, and this kind of connection expresses the conditional dependence between the variables. A Bayesian network is defined as the set of {D, S, P}, where

(1) D is a set of variables (or nodes).

(2) S is a set of conditional probability distributions (CPD).

$S = \{p(D_1), ..., p(D_n / \text{Parents}(D_n))\}.\text{Parents}(D_i) \subset D$ stands for all the parent nodes of D,

$p(D_i / \text{Parents}(D_i))$ is the conditional distribution of D_i.

(3) P is a set of marginal probability distributions.

$P = \{p_1(D_1), ..., p_n(D_n)\}$. $p_i(D_i)$ is the probability distribution of D_i.

In a Bayesian network, variables are used to express the events or objects. The problem could be modeled as the behavior of these variables. In general, we first determine the probability distribution of each variable and the conditional probability distribution between them based on expert experience. Then the joint distributions of these variables can be derived from these distributions. Finally, deductions can be developed for some variables of interest using some other known variables.

A Bayesian network is generally based on Bayesian statistical theory. Compared with other traditional data analysis methods, a Bayesian network has the following properties:

(1) A Bayesian network can increase the precision of the analysis by considering all the variables of interest. A typical example is a problem with two input and one output variables, in which the two input variables might be correlated. If both of the input variables can be observed, most of the models are able to predict the output. However, if one of the inputs cannot be observed, a precise prediction might be a wild wish. A Bayesian network provides a novel way by using the correlations between the two variables.

(2) A Bayesian network can model the causal relationships between variables. Thus, the interdependence of the variables could be studied in order to improve prediction. For example, for a market analyst, a straightforward question is whether it is worthwhile to increase the intensity of some advertising campaign to gain more sales of a certain product.. Bayesian networks can help determine whether the advertisement is an influencing factor and the extent of its influence.

A Bayesian network with Bayesian statistical methods can combine the relative knowledge in the problem domain and the available data effectively. It is useful for the situations with the prior information, especially in which little or no data are available or data collection is costly.

3. Modeling and Deduction

Let $X_i(1 \leq i \leq n)$ and $Y_i(1 \leq i \leq n)$ respectively denote the remaining defect numbers after the ith failure and the time between the (i-1)th and ith failure. During the defects removal process, it is possible to both remove and introduce defects, and hence X_{i+1} is dependent on X_i. The time interval between software failures is influenced by the number of remaining defects. Assuming that the next failure is independent of time, then the Markov Bayesian network can be used to model this problem, and the network graph is given in Figure1. In this paper, we will denote this as the MBN model.

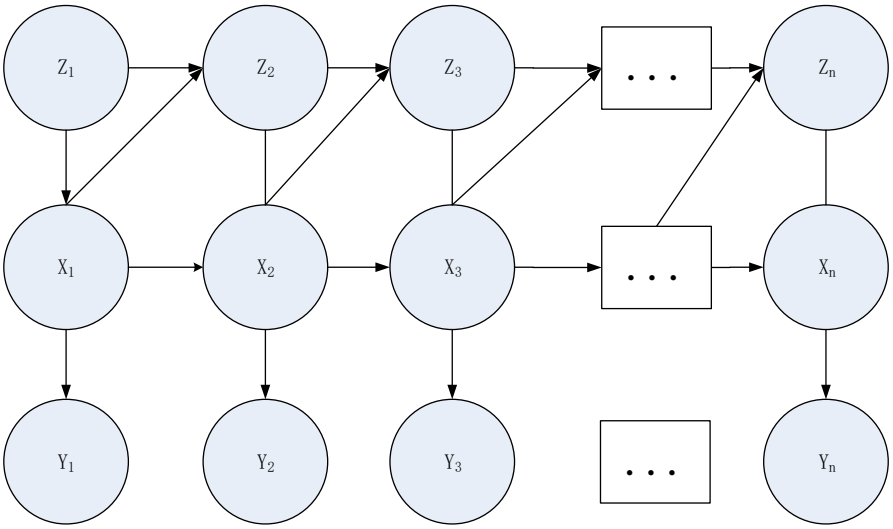


Figure 1. Model of Failure Prediction for Launch Vehicle Software

3.1. Determine the Distributions

In order to use the Markov Bayesian network model to predict software failure, the following three distributions are essential: one is the initial distribution of software defects, another is the distribution of software failure time with respect to the number of defects, and the third is distribution of the number of removal defects under software failure.

The initial distribution of software defects $p_0(x)$ could be estimated using software metrics from the development or software product.

When a software failure occurs, software programmers should try to detect and remove the defect. Sometimes it is a common practice to remove the defect together with introducing a new one. Overall, the tendency of the number of defects is decreasing. In this regard, the distribution of x_{i+1}can be assumed normal with a mean of E[$p(z_i)x_i$] ($p(z_i)$>0) and a standard deviation of z_i). Note that, here the $p(z_i)$ denotes the status of defect removing, and z_i) reflects the skill level differences between the defect-removing programmer. Due to the complexity of Launch Vehicle software, these two parameters should relate to the status of the Launch Vehicle software under test. In general, the software reliability is increasing during the defect removal process, that is, $p(z_i)$< 1. Thus, the relationship between x_{i+1} and x_i follows $x_{i+1}= p(z_i)x_i+w$ where w is a normal distribution random variable with mean 0.

We can now consider the failure of software with x_i defects inside. Assuming that the number of executable code line is N, then the average defects per line is x_i/N. By assuming h is the number of the executive code line per time unit, its failure probability is p when one defect is executed, then the software failure probability per unit time is

$$\begin{aligned} 1-(1-p)^{hx_i/N} &= 1-\exp\left\{\frac{hx_i\ln(1-p)}{N}\right\} \\ &= 1-\exp\{-\beta x_i\} \end{aligned} \tag{1}$$

where $\beta = -\frac{h}{N}\ln(1-p) > 0$.

It can be assumed that when the number of the remaining defects is xi, the software failure is of exponential distribution as following

$$p(y_i \mid x_i) = (1-e^{-\beta x_i})\cdot e^{-(1-e^{-\beta x_i})y_i} \tag{2}$$

In addition, the distribution of software status should be related to its previous status and the defect removal process, *i.e.*, $p(z_{i+1}) = p(z_{i+1} \mid z_i, x_i)$. Note that, the initial distribution of software status relates to the test suite and its partition, that is, $p(z_1) = f(suite, partition)$, which can be expressed with a *beta* distribution $beta(a_i,b_i)$. Besides, the number of remaining defects should be related to the distribution of software status, which means $p(z_{i+1}) = p(z_{i+1} \mid z_i, x_i) = beta(a_{i+1}, b_{i+1})$.

3.2. Software Failure Prediction when Parameters in Distributions are Known

As in standard Markov Bayesian network, the joint probability density of $X_1,\ldots,X_n$, $Y_1,\ldots,Y_n$ is

$$p(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n) = p(x_1) \cdot \prod_{i=1}^{n} p(y_i \mid x_i) \cdot \prod_{i=1}^{n} p(x_{i+1} \mid x_i)$$

and the marginal probability density is

$$p(x_1, x_2, ..., x_n) = p(x_1) \cdot \prod_{i=1}^{n} p(x_{i+1} \mid x_i)$$

.

The conditional probability density is

$$p(x_1, x_2, ..., x_n \mid y_1, y_2, ..., y_n) = \frac{p(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)}{p(y_1, y_2, ..., y_n)}$$

.

According to this equation, the estimates of nodes $X_1, X_2, ..., X_n$ can be estimated as $x_1, x_2, ..., x_n$, and therefore, x_{n+1} can be obtained through $p(x_{n+1} \mid x_n)$. In a similar way, y_{n+1} can be obtained by $p(y_{n+1} \mid x_{n+1})$.

In fact, the density is hard to compute in practice. Therefore, the Gibbs sampling approach can be adopted to approximate the equations. According to the Gibbs sampling theory the key point is to get samples from the distribution.

$$p(x_k \mid x_1, x_2, ..., x_{k-1}, x_{k+1}, ..., x_n, y_1, y_2, ..., y_n) = \frac{p(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)}{p(x_1, x_2, ..., x_{k-1}, x_{k+1}, ..., x_n, y_1, y_2, ..., y_n)}$$

$$(k = 2, ..., n-1) \tag{3}$$

Where the denominator is

$$\begin{aligned} I &= \int_{x_k} p(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n) dx_k \\ &= \int_{x_k} p(x_1) \prod_{i=1}^{n} p(y_i \mid x_i) dx_k \\ &= p(x_1) \prod_{i=1}^{k-1} p(y_i \mid x_i) \prod_{i=k+1}^{n} p(y_i \mid x_i) \prod_{i=1}^{k-2} p(x_{i+1} \mid x_i) \prod_{i=k+1}^{n-1} p(x_{i+1} \mid x_i) \cdot \int_{x_k} p(y_k \mid x_k) p(x_k \mid x_{k-1}) p(x_{k+1} \mid x_k) dx_k \end{aligned} \tag{4}$$

and the numerator is

$$\begin{aligned} II &= p(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n) \\ &= p(x_1) \prod_{i=1}^{n} p(y_i \mid x_i) \prod_{i=1}^{n-1} p(x_{i+1} \mid x_i) \\ &= p(x_1) \prod_{i=1}^{k-1} p(y_i \mid x_i) \prod_{i=k+1}^{n} p(y_i \mid x_i) \prod_{i=1}^{k-2} p(x_{i+1} \mid x_i) \prod_{i=k+1}^{n-1} p(x_{i+1} \mid x_i) \cdot p(y_k \mid x_k) p(x_k \mid x_{k-1}) p(x_{k+1} \mid x_k) \end{aligned} \tag{5}$$

Therefore, the distribution $p(x_k \mid x_1, x_2, ..., x_{k-1}, x_{k+1}, ..., x_n, y_1, y_2, ..., y_n)$ can be sampled.

As for the Launch Vehicle software status, the initial status should be $beta(a_1,b_1)$. And according to $p(z_{i+1}) = p(z_{i+1} \mid z_i, x_i) = beta(a_{i+1}, b_{i+1})$, the values of a_{i+1}, b_{i+1} can be calculated as

$$a_{i+1} = a_i + \gamma_i x_i + f_i$$
$$b_{i+1} = b_i + n_i - f_i \quad (6)$$

Where f_i denotes the number of failures during testing, and n_i is the number of tests between the *i*th and (*i*+1)th failure. In fact, the coefficient $_i$ can be estimated by the test suite and partition, which should be determined by the test engineers and will not be discussed in this paper.

4. Case study

Experiments on three Launch Vehicle software subject programs are conducted to validate the effectiveness of the proposed approach. In order to evaluate the effectiveness, this novel model is compared with two traditional models: the Jelinski–Moranda model (JM model) and the Goel–Okumoto NHPP model (GO model). These two models are usually taken as the benchmark in failure prediction research. In this section, the results of these three models are compared and analyzed.

4.1. Subject Programs

In this paper,Three subject programs are adopted for validation, that is, Launch Vehicle Telemetry Data Real-time Processing Software, Launch Vehicle Dynamical system Control Software and Launch Vehicle Measuring and Control Software.

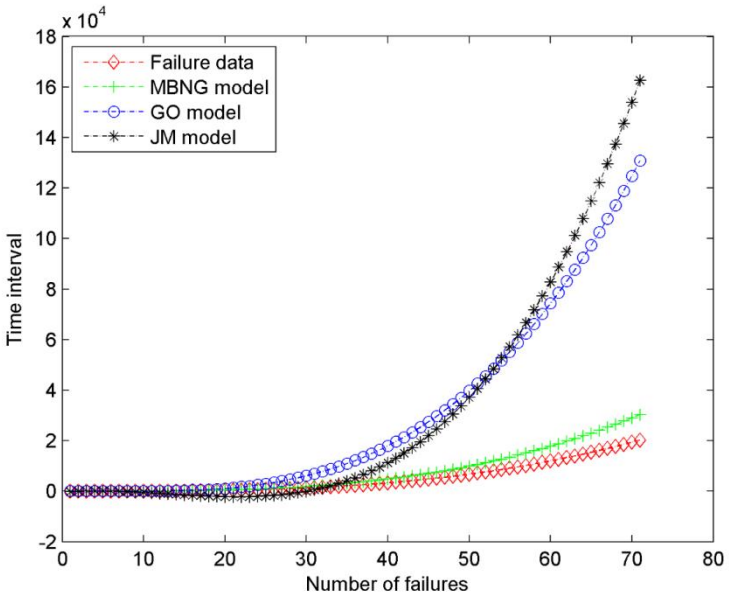
Telemetry Data Real-time Processing Software is an example program alone with Borland C++ Builder 6. This program is coded with C++ using Visual Component Library and contains about 10532 lines of code. The Dynamical system Control software is a standard Win32 application, which is designed according to the Launch VehicleDynamical system. The Measuring and Control Software is an example program alone with QT.

The number of seeded faults relevant to this study seeded in these software are 116, 126, and 71, respectively. In this study, the application should have passed a series of testing procedures before release, and thus “obvious” bugs should be eliminated. In this case, a static analysis tool, *i.e.*, FindBugs, is adopted to verify that none of the reported bugs in this study can be detected by FindBugs.

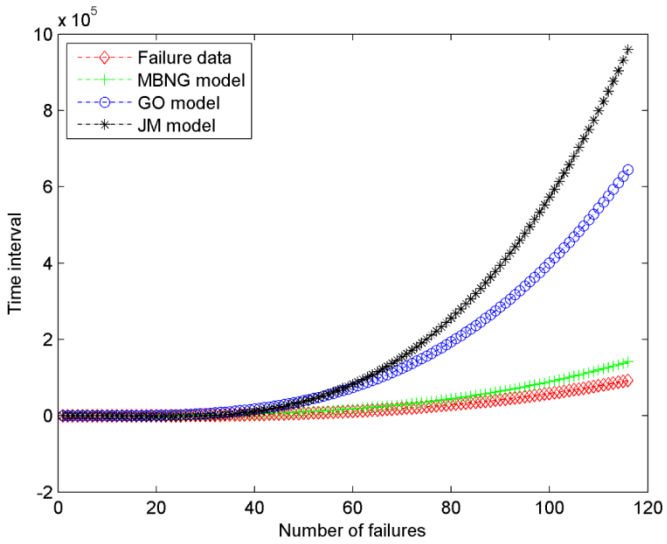
4.2. Experimental Results

With the assumption that the initial remaining number x_1 and the number of remaining defects after the *i*th failure x_{i+1} follow normal distribution, *i.e.* $x_1 \sim N(\mu_1, \sigma_1)$, $x_{i+1} \mid x_i \sim N(\mu_{i+1}, \sigma_{i+1})$,we could assume $x_{i+1} = \gamma x_i + w$, with $w \sim N(0, \sigma)$. The distribution of software failure time interval is determined by Eq. (1). As to the initial value of the parameters in the distribution, we could get them from the analysis of software and the testing process. In this case, it is assumed that $\mu_1 = 17$, $\sigma_0 = 1$, $\gamma_0 = 0.15$, $\beta_0 = 0.85$, $\sigma = 1$. The computing process is described in Section 3. The failure prediction is a dynamic rolling process, which means the (*i*+1)th failure time is predicted with the former *i* failure times available.

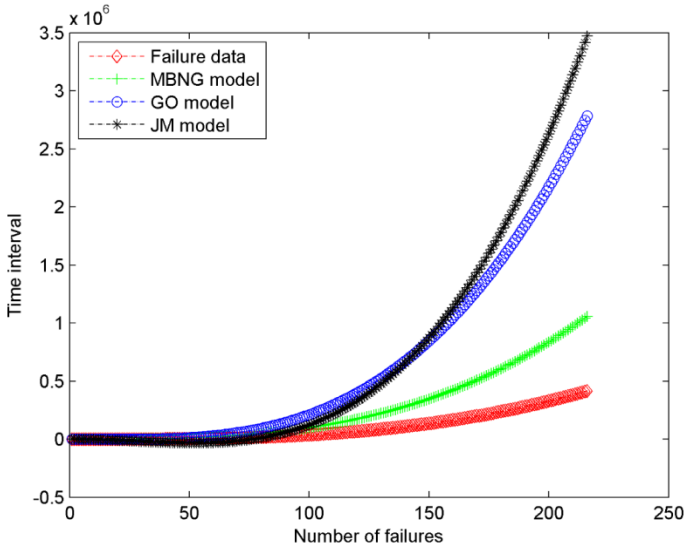
Figure 3(a), (b), (c) illustrate the time interval between failures (*y*) with the failure order number (*n*) for the three models and the data. From the results, we see that the MBNG model has good estimation and prediction capabilities.



(a) Telemetry Data Real-time Processing Software Failure Data and Prediction



(b) Dynamical System Control Software Failure Data and Prediction



C) Measuring and Control Software Failure Data and Prediction

Figure 2. Failure Data and Prediction Data from Three Different Methods

According to the above results, this novel MBNG model provides the most accurate failure data prediction among three prediction models. With the increasing number of failures, JM and GO models provide inaccurate failure data. Comparing JM and GO models, it can also be seen that JM model provides more accurate failure data than GO model when the number of failures is small. As the number of failures increase, GO model provides more accurate failure data than JM model.

4.3. Threats to Validity

Threats to construct validity in this case study are mainly on the measurement of the performance of a prediction model. In this study, providing accurate failure prediction is the only target for these models. Thus, the prediction of Launch Vehicle software failure using all three models are recorded and compared with the true failure data to determine the performance of each model.

Threats to *internal validity* usually are about possible bias in experimental design. In this study, all the three models are adopted to predict the failure date of Launch Vehicle software. These models used the same failure data produced by the same Launch Vehicle operation process. Thus, the design bias can be avoided as far as we can.

The external validity for such an empirical study is whether the obtained results can be generalized. In this study, the proposed MBNG model does not rely on any specific assumption of Launch Vehicle software, thus this model should be a general one. In addition, the four software subjects can represent a variety of Launch Vehicle software, which have different lines of code, defects and operational profile. Thus, we are confident about the effectiveness of MBNG model.

5. Discussion

Applications of Launch Vehicle software have been important in Launch Vehicle system. The reliability of such kind of software should be paid more attention to guarantee good user experience. Due to the complexity of Launch Vehicle, failure prediction models for common software might ignore the features of Launch Vehicle software, which might reduce their performance. Thus, a novel MBNG model is proposed by considering more information, including the initial distribution of software defects and the software status. In this way, MBNG is considered more effective in failure prediction

for Launch Vehicle software. Experiments on three Launch Vehicle software show that MBNG model can provide more accurate failure data than JM and GO models. In the future, more Launch Vehicle software should be considered for further validation of MBNG model. Another direction is to reconsider the assumptions of MBNG model, such as, the normal assumption of existing number of defects in the software.

References

[1] A. L. Goel, "Software reliability models: assumptions, limitations, and applicability", IEEE Transactions on Software Engineering, SE-11, no. 12, **(1985)**, pp. 1411-1423.
[2] Z. Jelinski and P. Moranda, "Software Reliability Research, Statistical Computer Performance Evaluation", W. Freiberger, Ed., New York: Academic Press, **(1972)**, pp. 465-484.
[3] K. Y. Cai, L. Cai and W. D. Wang, "On the neural network approach in software reliability modeling", Journal of Systems and Software, vol. 58, no. 1, **(2001)**, pp. 47-62.
[4] C. G. Ba, Q. P. Hu, M. Xie and S. H. Ng, "Software failure prediction based on a Markov Bayesian network model", Journal of Systems and Software, vol. 74, no. 3, **(2005)**, pp. 275-282.
[5] J. D. Pfefferman and B. C. Frias, "A non-parametric non-stationary procedure for failure prediction", IEEE Transactions on Reliability, vol. 51, no. 4, **(2002)**, pp. 434-442.
[6] N. Karunanithi and D. Whitley, "Prediction of software reliability using feed forward and recurrent neural nets", International Joint Conference on Neural Networks, vol. 1, **(1992)**, pp. 800-805.
[7] T. M. Khoshgoftaar and R. M. Szabo, "Using neural networks to predict software faults during testing", IEEE Transactions on Reliability, vol. 45, no. 3, **(1996)**, pp. 456-462.
[8] R. Sitte, "Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration", IEEE Transactions on Reliability, vol. 48, no. 3, **(1999)**, pp. 285-291.
[9] S. L. Ho, M. Xie and T. N. Goh, "A study of the connectionist models for software reliability prediction", Computers and Mathematics with Application, vol. 46, no. 7, **(2003)**, pp. 1037-1045.
[10] T. W. Bickmore, "Real-time sensor data validation. NASA Contractor Report 195295", National Aeronautics and Space Administration, **(1994)**.
[11] D. Madigan, K. Mosurski and R. G. Almond, "Explanation in belief networks", Journal of Computational and Graphical Statistics, vol. 6, **(1996)**, pp. 160-187.
[12] R. W. Lewis and R. S. Ransing, "A semantically constrained Bayesian network for manufacturing diagnosis", International Journal of Production Research, vol. 35, no. 8, **(1997)**, pp. 2171-2187.
[13] E. Castillo, J. M. Sarabia and C. Solares, "Uncertainty analyses in fault trees and Bayesian networks using FORM SORM methods", Reliability Engineering and Systems Safety, vol. 65, no. 1, **(1999)**, pp. 29-40.
[14] W. P. Zhu, "Using Bayesian network on network tomography", Computer Communication, vol. 26, no. 2, **(2003)**, pp. 155-163.
[15] J. H. Sigurdsson, L. A. Walls and J. L. Quigley, "Bayesian belief nets for managing expert judgment and modeling reliability", Quality and Reliability Engineering International, vol. 17, **(2001)**, pp. 181-190.
[16] M. Neil, B. Littlewood and N. Fenton, "Applying Bayesian belief networks to system dependability assessment", Proceedings of Safety Critical Systems Club Symposium, Springer-Verlag, Berlin, **(1996)**.
[17] M. Bouisso, F. Martin and A. Ourghanlian, "Assessment of a safety-critical system including software: a Bayesian belief network for evidence sources", Proceedings of the Annual Reliability and Maintainability Symposium, **(1999)**, pp. 142-150.
[18] C. Smidts, R. W. Stoddard and M. Stutzke, "Software reliability models: an approach to early reliability prediction", IEEE Transactions on Reliability, vol. 47, **(1998)**, pp. 268-278.
[19] N. Fenton and M. Neil, "Software metrics: successes, failures and new directions", Journal of Systems and Software, vol. 47, no. 2-3, **(1999)**, pp. 149-157.
[20] G. J. Pai, "Combining Bayesian belief networks with fault tree to enhance software reliability analysis", Proceedings of the IEEE International Symposium on Software Reliability Engineering, **(2001)**.
[21] Y. Yu and B. W. Johnson, "A BBN approach to certifying the reliability of COTS software system reliability", Proceedings of Annual Reliability and Maintainability Symposium, **(2003)**, pp. 19-24.
[22] M. I. Jordan, "Learning in Graphical Models", MIT Press, Cambridge, **(1999)**.
[23] E. Gelfand and F. M. Smith, "Sampling-Based Approaches to Calculating Marginal Densities", Journal of the American Statistical Association, vol. 85, no. 410, **(1990)**, pp. 398-409.
[24] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm", Journal of the Royal Statistical Society, vol. 39, no. 1, **(1977)**, pp. 1-38.