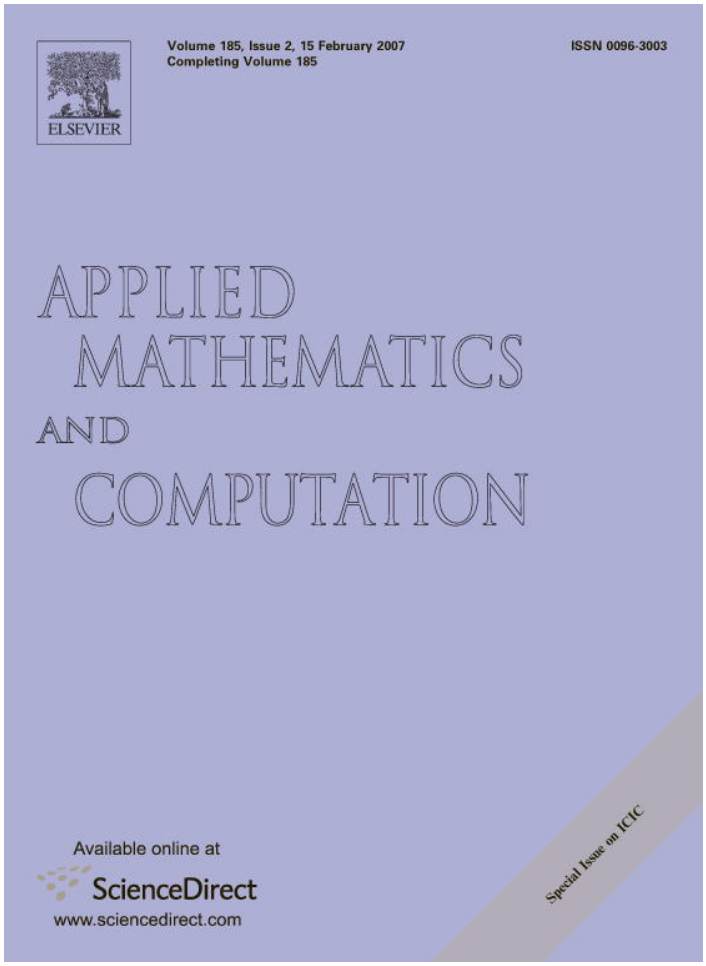

Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 185 (2007) 1120–1130
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

A hierarchical mixture model for software reliability prediction

Shaoming Li [a,b], Qian Yin [a,b], Ping Guo [a,b,*], Michael R. Lyu [c]

[a] Department of Computer Science, Beijing Normal University, Beijing 100875, China
[b] Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China
[c] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong, China

Abstract

It is important to develop general prediction models in current software reliability research. In this paper, we propose a hierarchical mixture of software reliability models (HMSRM) for software reliability prediction. This is an application of the hierarchical mixtures of experts (HME) architecture. In HMSRM, individual software reliability models are used as experts. During the training of HMSRM, an Expectation–Maximizing (EM) algorithm is employed to estimate the parameters of the model. Experiments illustrate that our approach performs quite well in the later stages of software development, and better than single classical software reliability models. We show that the method can automatically select the most appropriate lower-level model for the data and performances are well in prediction.

Keywords: Software reliability model; HME; EM algorithm; Prediction

1. Introduction

Software reliability is one of a number of aspects of computer software which can be taken into consideration when determining the quality of the software. Nowadays, software reliability is widely used by such users as Alcatel, AT&T, Bellcore, CNES (France), ENEA (Italy), Ericsson Telecom, Hewlett Packard, Hitachi, IBM, NASAs Jet Propulsion Laboratory, Lockheed-Martin, Lucent Technologies, Microsoft, Mitre, Motorola, Nortel, Saab Military Aircraft, Tandem Computers, the US Air Force, and the US Marine Corps. There are over 65 published articles by users of software reliability engineering, and the number continues to grow [1].

Building good reliability models is one of the key problems in the field of software reliability. A good software reliability model should give good predictions of future failure behavior, compute useful quantities and be widely applicable. Therefore, a very important goal of current software reliability research is to develop general prediction models [2]. Existing models typically rely on assumptions about development environments, the nature of software failures and the probability of individual failure occurrences. Thus each model can be shown to perform well with a specific failure data set, but no model appears to perform well for all cases.

* Corresponding author. Address: Department of Computer Science, Beijing Normal University, Beijing 100875, China.
E-mail addresses: pguo@ieee.org (P. Guo), lyu@cse.cuhk.edu.hk (M.R. Lyu).

0096-3003/$ - see front matter
doi:10.1016/j.amc.2006.07.028

In recent years, many methods have been proposed to improve the quality of reliability models. Some non-parametric methods have been introduced into the field [3–5], such as Neural Network and Genetic Programming. These types of method are flexible, but they often lack theoretical basis. Other work has focused on building a unified scheme to solve the generalizability problem mentioned above [6,7]. Such schemes are often theoretically rigorous, but some of them are not practical because there are too many parameters to estimate. Another approach aims try to use various methods to improve the estimation performance of reliability models [8]; our work belongs to this type.

In this paper, a hierarchical mixture of software reliability models (HMSRM) is proposed, which is based on the principle of "divide-and-conquer". HMSRM can be considered as an application of the hierarchical mixtures of experts (HME) architecture [9]. The model takes the outputs of several "experts" as inputs and integrates them together effectively using gating networks. Unlike many other divide-and-conquer algorithms, HME makes use of "soft" splits of data. Thus it can employ as much input information as possible in order to make decisions.

HMSRM is designed to exploit the advantages of several classical software reliability models, which are treated as "experts". By this means, we can get more accurate predictions. In order to adjust the "contribution" of each expert, an Expectation–Maximization (EM) algorithm [9,12] is employed. The experimental results show that the generalization performance of HMSRM is better than individual classical software reliability models, and that it is a good approach to model selection.

2. Classical software reliability models

In the course of development of software reliability research, many models have been built to predict future failures. Some of the models are described as Nonhomogeneous Poisson Process (NHPP) models, because the mean value function $M(t)$ represents the cumulative number of faults exposed up to time t [10]. In practice, many of the NHPP models are proved to be effective only in a particular environment. In order to construct HMSRM, we choose four classical NHPP models as the "experts" in the HMSRM. They are as follows:

(1) Goel and Okumoto model (G–O model)

$$M(t) = a(1 - \mathrm{e}^{-bt}). \tag{1}$$

(2) Duane model

$$M(t) = at^{b}. \tag{2}$$

(3) S-shaped model

$$M(t) = a(1 - (1 + bt)\mathrm{e}^{-bt}). \tag{3}$$

(4) K-stage Erlangian (gamma) growth curve model ($k = 3$)

$$M(t) = a\left(1 - \left(1 + bt + \frac{(bt)^2}{2}\right)\mathrm{e}^{-bt}\right). \tag{4}$$

In these NHPP models, usually parameter a usually represents the mean number of software failures that will eventually be detected, and parameter b represents the probability that a failure is detected in a constant period.

3. HMSRM and the EM algorithm

The HMSRM we used here is based on the HME architecture. The failure data are given in pairs as $X = \{(x^{(t)}, d^{(t)})\}$, where $x^{(t)}$ represents the execution time and $d^{(t)}$ represents the number of failures. In the training algorithm, we divide the input space into a set of sub-regions and assign each "expert" to certain regions. The bounds of the regions are adjusted dynamically by the algorithm.

The architecture of a two-level HMSRM [9,13] is shown in Fig. 1. It has exactly the same tree-like structure as the HME model. Classical reliability models, which are expert networks in the HME model, sit at the leaves

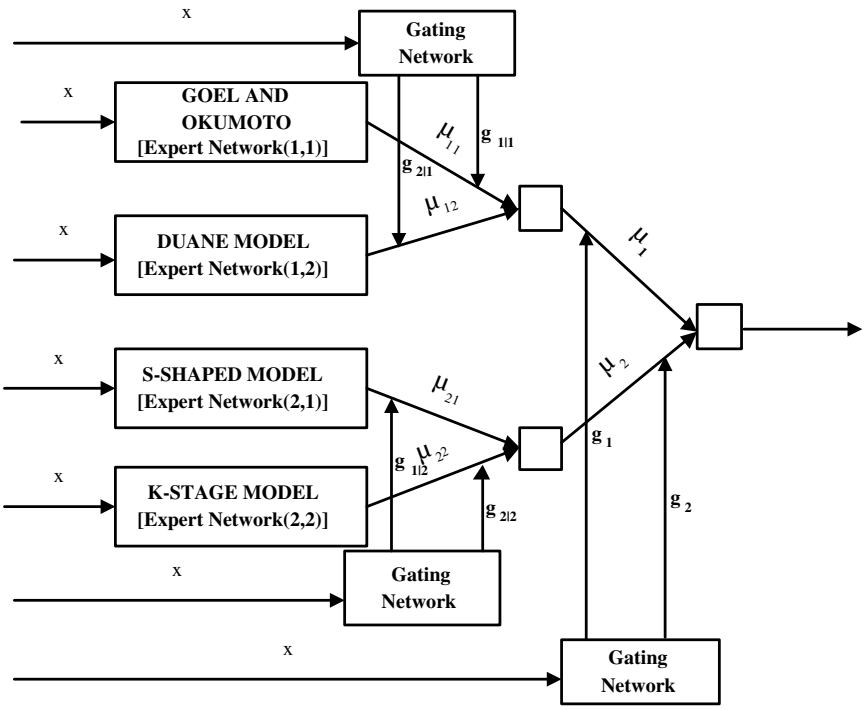

Fig. 1. A two-level hierarchical mixture of software reliability model.

of the tree, and gating networks control the nonterminals of the tree. Each classical model receives the x (execution time) as input and produces a result μ_{ij} (number of failures). In the meantime, the gating networks receive the same x and compute the weights of each lower-level model.

In the architecture, each lower-level model produces its output μ_{ij} as a function of the input x:

$$\mu_{ij} = f_{i,j}(x). \tag{5}$$

The gating networks are considered as generalized linear. We define the intermediate variables:

$$\lambda_i = v_i^{\mathrm{T}} x, \tag{6}$$

where v_i is a weight vector. The top-level gating network is defined as follows:

$$g_i = \frac{e^{\lambda_i}}{\sum_k e^{\lambda_k}}. \tag{7}$$

Similarly, the lower-level networks are defined as follows:

$$g_{j|i} = \frac{e^{\lambda_{ij}}}{\sum_k e^{\lambda_{ik}}}, \qquad \lambda_{ij} = v_{ij}^{\mathrm{T}} t, \tag{8}$$

where v_{ij} is also a weight vector for the expert network (i,j). Thus, the output of the whole HMSRM is

$$\mu = \sum_i g_i \mu_i, \qquad \mu_i = \sum_j g_{j|i} \mu_{ij}. \tag{9}$$

As g_i and $g_{j|i}$ sums to one for each x respectively, they can be considered as "prior" probabilities. Using Bayes' rules, we can define the posterior probabilities as follows:

$$h_i = \frac{g_i \sum_j g_{j|i} P_{ij}(y)}{\sum_i g_i \sum_j g_{j|i} P_{ij}(y)}, \tag{10}$$

and

$$h_{j|i} = \frac{g_{j|i} P_{ij}(y)}{\sum_j g_{j|i} P_{ij}(y)}, \tag{11}$$

where P_{ij} is the distribution of the outputs of the classical models. Also, joint posterior probabilities are defined as

$$h_{ij} = \frac{g_i g_{j|i} P_{ij}(y)}{\sum_i g_i \sum_j g_{j|i} P_{ij}(y)}. \tag{12}$$

We employ the EM algorithm to solve the learning problem of HMSRM. The EM algorithm is a general technique for maximum likelihood estimation; it is an iterative approach. Each iteration of an EM algorithm has two steps: an Estimation (E) step and a Maximization (M) step. The M step involves the maximization of a likelihood function that is redefined in each iteration by the E step [9].

In order to apply the EM algorithm to HMSRM, we define indicator variables z_i and $z_{j|i}$, such that one and only one of the z_i is equal to one, and one and only one of the $z_{j|i}$ is equal to one. Thus their expectation can be written as follows:

$$E[z_{ij}^{(t)}|X] = P\left(z_{ij}^{(t)} = 1 | y^{(t)}, x^{(t)}, \theta^{(p)}\right) = \frac{P(y^{(t)}|z_{ij}^{(t)} = 1, x^{(t)}, \theta^{(p)}) P\left(z_{ij}^{(t)} = 1 | x^{(t)}, \theta^{(p)}\right)}{P(y^{(t)}|x^{(t)}, \theta^{(p)})} = \frac{P(y^{(t)}|x^{(t)}, \theta^{(p)}) g_i^{(t)} g_{j|i}^{(t)}}{\sum_i g_i^{(t)} \sum_j g_{j|i}^{(t)} P(y^{(t)}|x^{(t)}, \theta^{(p)})} = h_{ij}^{(t)}, \tag{13}$$

where θ stands for the parameters of the expert networks. Also note that $E[z_i^{(t)}|X] = h_i^{(t)}$ and $E[z_{j|i}^{(t)}|X] = h_{j|i}^{(t)}$.

The complete-data likelihood is:

$$l_c(\theta; y) = \sum_t \sum_i \sum_j z_{ij}^{(t)} \ln\left(g_i^{(t)} g_{j|i}^{(t)} P_{ij}(y^{(t)})\right), \tag{14}$$

and then the expectation of the complete-data likelihood is

$$\begin{aligned} Q(\theta, \widehat{\theta}(n)) &= \sum_t \sum_i \sum_j E[z_{ij}^{(t)}]\left(\ln g_i^{(t)} + \ln g_{j|i}^{(t)} + \ln P_{ij}(y^{(t)})\right) \\ &= \sum_t \sum_i \sum_j h_{ij}^{(t)}\left(\ln g_i^{(t)} + \ln g_{j|i}^{(t)} + \ln P_{ij}(y^{(t)})\right). \end{aligned} \tag{15}$$

The M step requires maximization of the $Q(\theta, \widehat{\theta}(n))$. According to Eq. (15), we can adjust the lower-level model parameters and the gating network weights. Thus the M step reduces to the following optimization problems:

$$w_{ij}^{(p+1)} = \arg\max_{w_{ij}} \sum_t h_{ij}^{(t)} \ln P_{ij}(y^{(t)}), \tag{16}$$

$$v_i^{(p+1)} = \arg\max_{v_i} \sum_t \sum_k h_k^{(t)} \ln g_k^{(t)}, \tag{17}$$

$$v_{ij}^{(p+1)} = \arg\max_{v_{ij}} \sum_t \sum_k h_k^{(t)} \sum_l h_{l|k}^{(t)} \ln g_{l|k}^{(t)}, \tag{18}$$

where w_{ij} is the parameters of the (i,j) lower-level model. To simplify the problem, we assume the distribution of P_{ij} is normal.

$$P_{ij}(y^{(t)}) = k\mathrm{e}^{-(d^{(t)} - y_{ij}^{(t)})^2}, \tag{19}$$

where k is an constant value. Thus Eq. (16) reduces to

$$w_{ij}^{(p+1)} = \arg\min_{w_{ij}} \sum_t h_{ij}^{(t)}\left(d^{(t)} - y_{ij}^{(t)}\right)^2, \tag{20}$$

where y_{ij} is the output of the (i,j) lower-level model and $d^{(t)}$ is the corresponding result of $x^{(t)}$.

4. Experiments

The training and testing data SYS1 and CSR1 are selected from the CD-ROM of the ''Handbook of Software Reliability Engineering'' [11]. In data set SYS1, there are 136 samples that contain time and number of failures, while there are 397 failures and corresponding time-to-failure data in CSR1. We take the early part of the software running data to train the HMSRM.

In the experiments, we construct a two level HMSRM. At the terminal nodes sit the four single models as shown in Fig. 1. In this part, two experiments are performed to verify the model using data SYS1. Before the experiment, the data are normalized. In the first experiment, the model is trained with the first 60% of the samples on the time axis, while the remaining failure data are used for test. The data is divided into 114 training samples and 22 testing samples. In the second experiment, the model is trained with the first 75% of the samples on the time axis, which means there are 128 training samples and eight testing samples. The data are used to train the HMSRM with the EM algorithm mentioned above. As a comparison, we also train individual NHPP models. In order to simplify the problem, the gradient descent learning algorithm is adopted to estimate the parameters of the classical models. Also, the variance is used to measure the quality of the models. The results are shown in Tables 1 and 2.

Figs. 2 and 3 show the curves of the HMSRM and individual models, respectively, in comparison with the real data. The two figures indicate that HMSRM works well in the prediction and the DUANE model is much better than other individual models. From Table 3, we can find that HMSRM is as good as the DUANE model in prediction, or even better under some conditions. This suggests HMSRM is an effective method in failure prediction.

Table 1
Prediction results based on 60% data

Model	Parameters	Variance (training data)	Variance (testing data)
Goel and Okumoto model	$a = 0.8014$ $b = 5.8456$	7.4913e−004	0.0195
Duane model	$a = 1.1054$ $b = 0.4897$	7.5958e−004	0.0021
S-shaped model	$a = 0.7354$ $b = 15.1317$	0.0040	0.0374
K-stage model ($k = 3$)	$a = 0.7256$ $b = 23.0171$	0.0070	0.0411
HMSRM		3.2064e−004	0.0022

Table 2
Prediction results based on 75% data

Model	Parameters	Variance (training data)	Variance (testing data)
Goel and Okumoto model	$a = 0.8703$ $b = 5.0047$	0.0014	0.0133
Duane model	$a = 1.0976$ $b = 0.4860$	6.8968e−004	0.0045
S-shaped model	$a = 0.7911$ $b = 13.7296$	0.0056	0.0339
K-stage model ($k = 3$)	$a = 0.7742$ $b = 21.7276$	0.0089	0.0403
HMSRM		3.6517e−004	0.0011

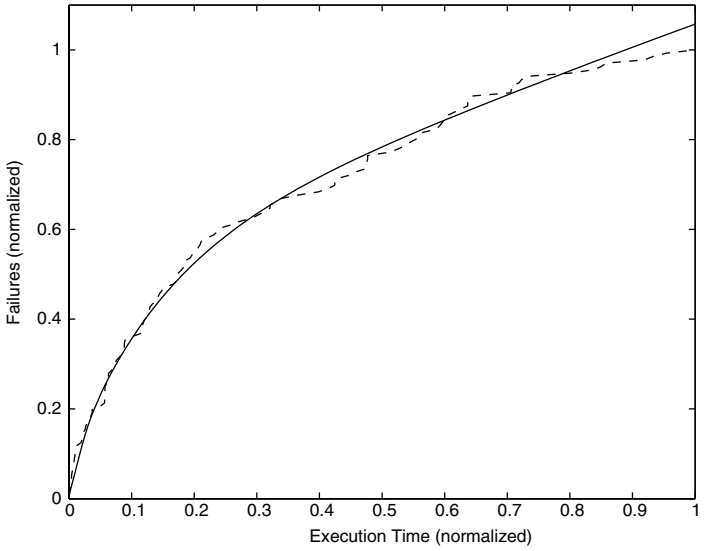

Fig. 2. The prediction curve of HMSRM in 75% prediction: the solid line shows the HMSRM prediction result and the dashed line shows the real failure data.

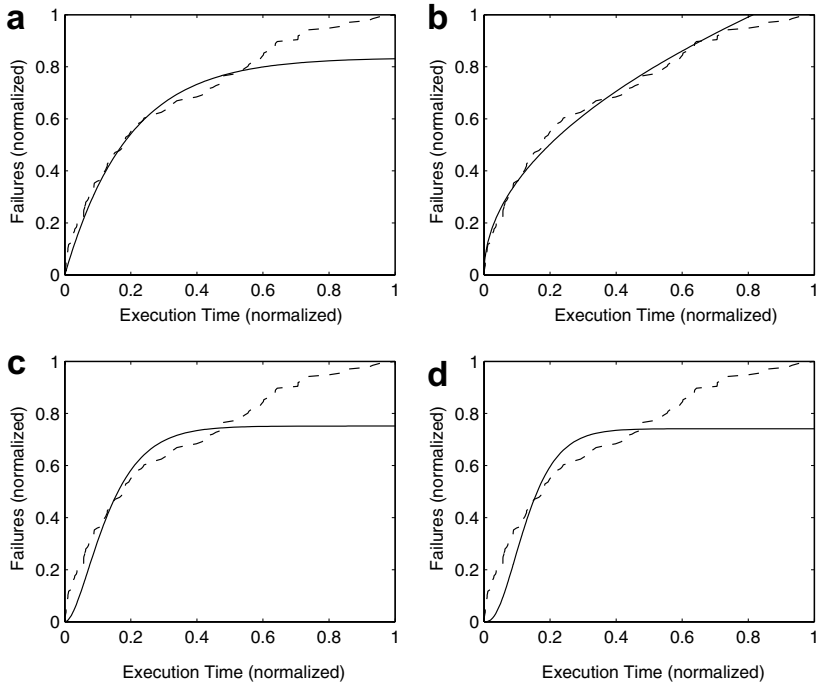

Fig. 3. The prediction curves of individual models in 75% prediction: the solid lines show the prediction result and the dashed lines show the real failure data. (a) G–O model, (b) Duane model, (c) S-shaped model, (d) *K*-stage model.

Table 3
Prediction results (data normalized)

Model	60% Variance (training data)	60% Variance (testing data)	75% Variance (training data)	75% Variance (testing data)
Goel and Okumoto	7.4913e−004	0.0195	0.0014	0.0133
Duane	7.5958e−004	0.0021	6.8968e−004	0.0045
S-shaped	0.0040	0.0374	0.0056	0.0339
K-stage ($k = 3$)	0.0070	0.0411	0.0089	0.0403
HMSRM	3.2064e−004	0.0022	3.6517e−004	0.0011

Also, Figs. 4 and 5 show "contribution" curves of the final result of the HMSRM in the experiments. They represent the effects of every single models in the final result in various time periods. As DUANE model works relative well for a longer time period, it has the largest contribution. G–O model also works well with specific time period. The other two models are not suitable in this region, therefore they are restrained by the gating networks.

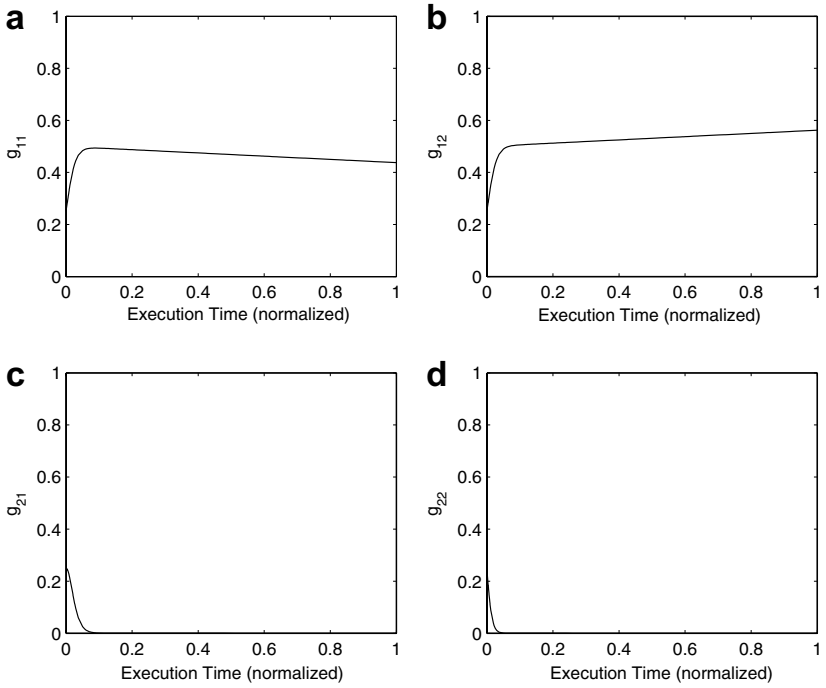

Fig. 4. The contribution curves of the lower-level models in the HMSRM in 60% prediction. (a) G–O model, (b) Duane model, (c) S-shaped model, (d) *K*-stage model.

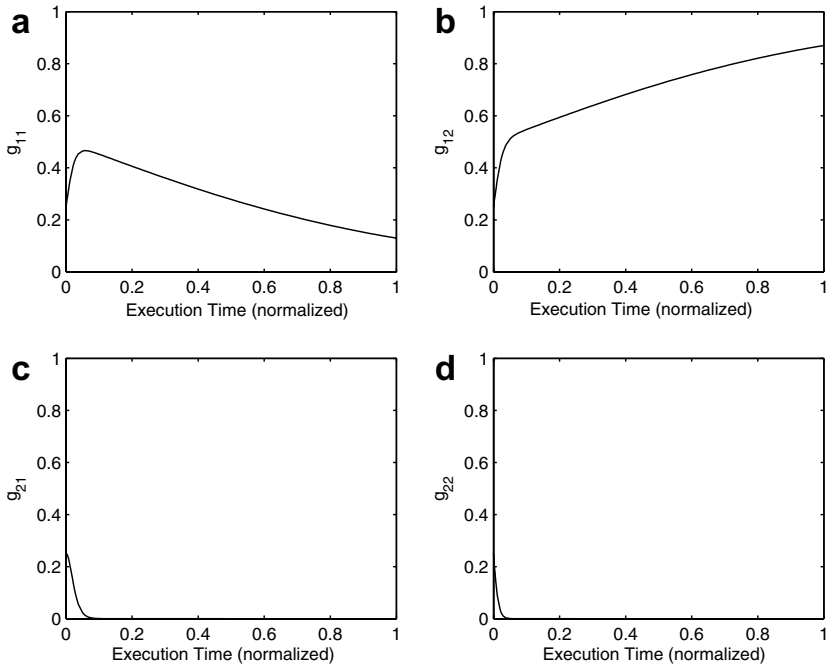

Fig. 5. The contribution curves of the lower-level models in the HMSRM in 75% prediction. (a) G–O model, (b) Duane model, (c) S-shaped model, (d) *K*-stage model.

As shown in Eqs. (7) and (8), g_i and $g_{j|i}$ have an exponential function form, models that are less suitable are gradually suppressed, and one model ultimately has the biggest weight as time t increases. Thus the most suitable model has a dominating effect on the prediction, as the DUANE model does above. This makes the prediction close to that of the best model. As the best model is selected automatically by the gating networks during the training, HMSRM is a good approach for selecting appropriate lower-level models in software reliability prediction. Besides the dominant one, other lower-level software reliability models in the HMSRM may work for part of the data, just as the G–O model does in the above experiment. This may help the HMSRM improve part of the prediction data, and so the HMSRM may get better results than the best single lower-level model.

Table 3 also shows that if we have a larger set of failure data, the accuracy of prediction is improved. More information about the failures can help the gating networks choose the best lower-level model and decide the most suitable regions for the lower-level models more accurately. More information is also useful for training the lower-level models, which will consequently lead to better predictions. This result can also be found by comparing Figs. 4 and 5. In Fig. 5, the HMSRM depends upon the DUANE model much more, whereas G–O model has a relatively low effect. We can see that leads to a better prediction result, as shown in the tables.

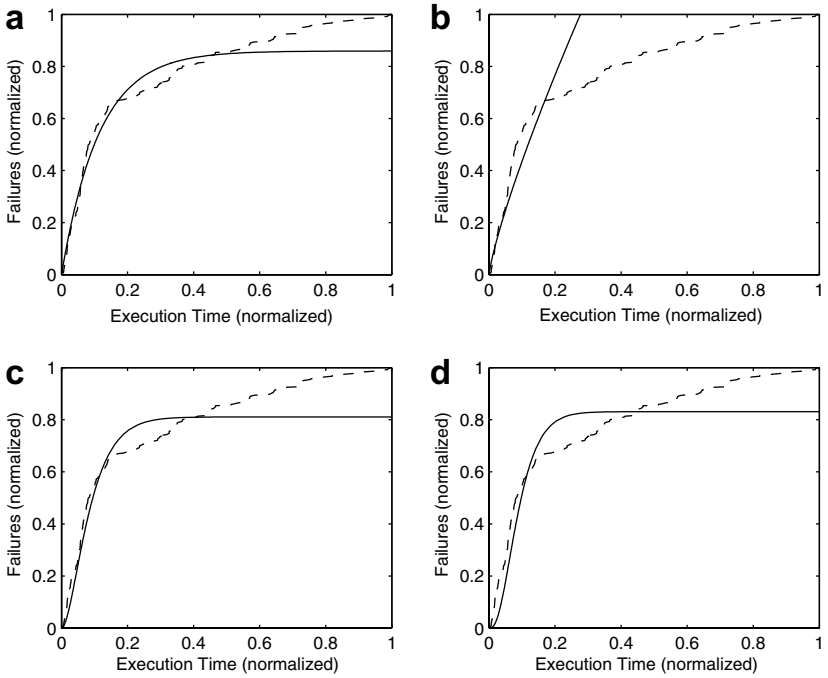

Fig. 6. The prediction curves of individual models in CSR1 prediction: the solid lines show the prediction result and the dashed lines show the real failure data. (a) G–O model, (b) Duane model, (c) S-shaped model, (d) K-stage model.

Table 4
Prediction results of CSR1 based on the first 300 samples

Model	Parameters	Variance (training data)	Variance (testing data)
Goel and Okumoto model	$a = 0.8079$ $b = 9.9644$	0.0011	0.0101
Duane model	$a = 2.8549$ $b = 0.8165$	0.0127	1.0798
S-shaped model	$a = 0.7478$ $b = 25.7710$	7.6480e−004	0.0222
K-stage model ($k = 3$)	$a = 0.7545$ $b = 38.3136$	0.0031	0.0205
HMSRM		7.9633e−004	0.0098

Another experiment is done to verify the model using data set CSR1. In the experiment, we attempt to derive a prediction of the software quality early in the testing process. There are 397 samples in the data set. First the data set is converted into the form of time and corresponding number of failures, and then it is normalized. As before, we use the earlier 300 samples as the training set and the remaining 97 samples to verify the prediction result. This means that the training data covers 33% of the time axis. The prediction results of individual models are shown in Fig. 6. We can see the most seriously problematic of the individual models is the DUANE model, which is not suitable for the data. It fits some of the early data, but the prediction result has a large disparity with the real data.

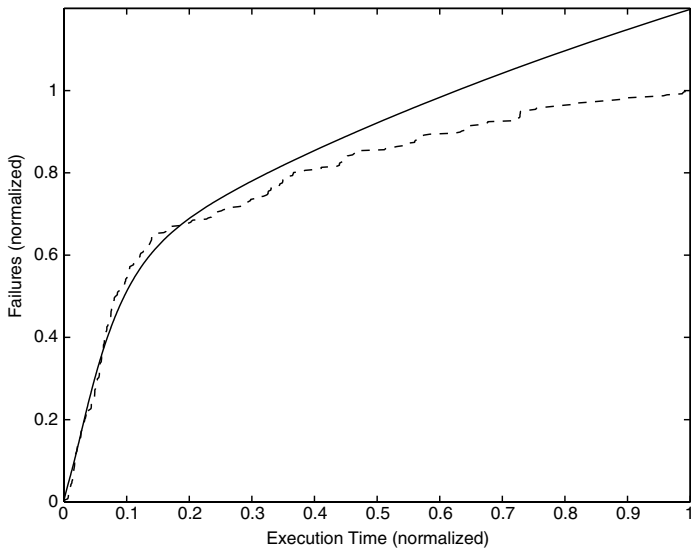

Fig. 7. The prediction curve of HMSRM in CSR1 prediction: the solid line stands for the HMSRM prediction result and the dashed line stands for the real failure data.

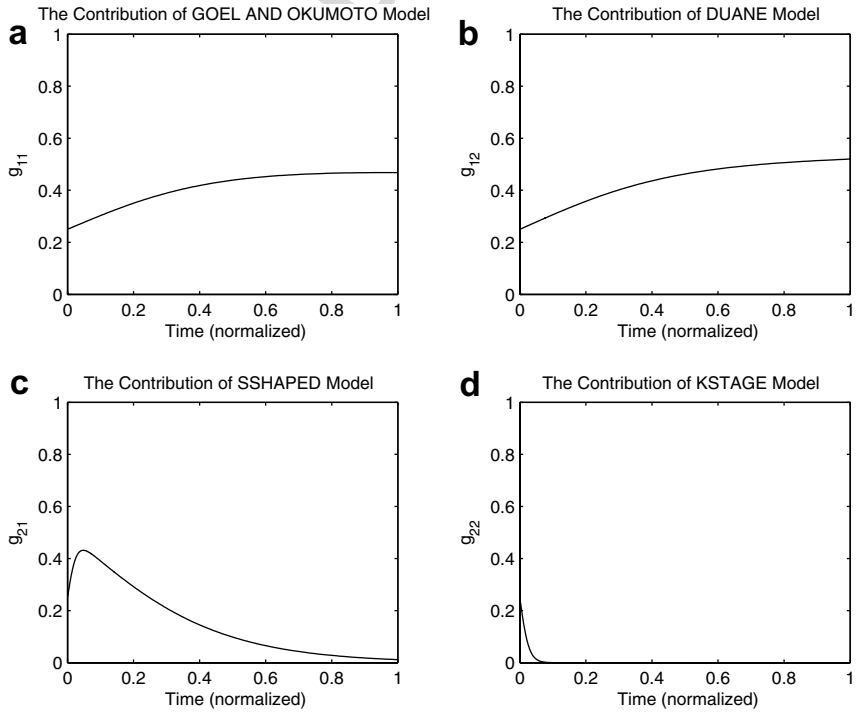

Fig. 8. The contribution curves of the lower-level models in the HMSRM in CSR1 prediction. (a) G–O model, (b) Duane model, (c) S-shaped model, (d) *K*-stage model.

Table 5
Prediction results of CSR1 based on the first 50% samples

Model	Parameters	Variance (training data)	Variance (testing data)
Goel and Okumoto model	$a = 0.8710$ $b = 8.3477$	0.0015	0.0052
Duane model	$a = 1.5970$ $b = 0.7013$	0.0191	0.1232
S-shaped model	$a = 0.8491$ $b = 18.8443$	0.0056	0.0080
K-stage model ($k = 3$)	$a = 0.8444$ $b = 28.5820$	0.0116	0.0088
HMSRM		5.6953e−004	4.1115e−004

HMSRM is applied to solve the problem and the result is shown in Table 4. The structure of the network is the same as we used in the first experiment. Though the DUANE model is very effective for the data set SYS1, it is not suitable for the current CSR1 samples. From the result, we can see HMSRM is not affected by the irrelevancy of each individual models and can still produce an acceptable prediction. This proves our proposed technique is not dependent on single specific models, and demonstrates that it can select the best "contribution curve" for various data sets automatically.

Fig. 7 gives the prediction curve of our mixture models and Fig. 8 gives the contribution curves of individual models. From these curves, we can see there are some problems remaining. Though the prediction of the mixture model is a little better than individual models, it has some disagreement with the real data. From the contribution curves, we can see there is no single dominant model. According to Fig. 6, none of the lower-level models is suitable for the data set. However, the mixture model is based on these single models. Thus the prediction accuracy of the mixture model is impaired. Also, the insufficient amount of training data is an important reason for the bad performance.

As a comparison, we also use 50% of the data in time axis as training data. The result is given in Table 5. From Tables 4 and 5, it can be seen that the performances of all the models are improved by the provision of more information in the training phase. This also benefits the mixture model. Again, it shows that the performance of the mixture model is restricted by some of the lower-level models, and indicates that our mixture model is not suitable for early software failure rate prediction.

5. Conclusions

In this paper, we apply the HME architecture to mix classical software-reliability models and to form a novel model, HMSRM. During the training phase, the EM algorithm is used to ensure the convergence of the training. The experimental results show that the HMSRM can select the appropriate lower-level models for the given failure data sets automatically and can produce a good prediction.

Since its prediction quality is based on the lower-level models employed HMSRM can be built with a large number of successful reliability models in order to achieve a good performance, thanks to its expandable architecture. Also, it can suppress irrelevant models in the architecture. From the above discussion, we conclude that HMSRM is a promising method in reliability prediction for the later part of the software development period.

Acknowledgements

This work was fully supported by a grant from the National Natural Science Foundation of China (Project No. 60275002) and a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4205/04E).

References

[1] John D. Musa, More reliable software faster and cheaper, Tutorial, 16th International Symposium on Software Reliability Engineering (ISSRE 16), Chicago, IL, 2005.
[2] Karunanithi Nachimuthu, Whitley Darrell, K. Malaiya Yashwant, Using neural networks in reliability prediction, IEEE Transactions on Software Engineering July/August (1992) 53–59.
[3] R. Sitte, Comparison of software reliability growth predictions: neural networks vs parametric recalibration, IEEE Transactions on Software Engineering 48 (3) (1999) 291–385.
[4] Miyoung Shin, Amrit L. Goel, Empirical data modeling in software engineering using radial basis functions, IEEE Transactions on Software Engineering 26 (6) (2000) 567–576.
[5] Eduardo Oliveira Costa, Silvia R. Vergilio, Aurora Pozo, Gustavo Souza, Modeling software reliability growth with genetic programming, in: 16th International symposium on software reliability engineering (ISSRE 16), Chicago, IL, 2005, pp. 171–180.
[6] Chin-Yu Huang, Michael R. Lyu, Sy-Yen Kuo, Unified scheme of some nonhomogenous Poisson process models for software reliability estimation, IEEE Transactions on Software Engineering 29 (3) (2003) 261–269.
[7] Tadashi Dohi, Shunji Osaki, Kishor S. Trivedi, An infinite server queueing approach for describing software reliability growth unified modeling and estimation framework, in: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04). pp. 120–129.
[8] Hiroyuki Okamura, Atsushi Murayama, Tadashi Dohi, EM algorithm for discrete software reliability models: a unified parameter estimation method, in: Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04), pp. 219–228.
[9] Michael I. Jordan, Rober A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, Neural Computation 6 (1994) 181–214.
[10] T.M. Khoshgoftaar, T.G. Woodcock, Software reliability model selection: a case study, in: Proceedings of the 2nd International Symposium on Software Reliability Engineering, Austin, TX, IEEE Computer Society, (1991), pp. 183–191.
[11] Michael R. Lyu, Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGraw Hill, New York, 1996.
[12] Haykin Simon, Neural Networks: A Comprehensive Foundation, 2/E, Prentice Hall, 1999, pp. 351–386.
[13] Yin Qian, Li Shaoming, Guo Ping, A case study on the hierarchical software reliability model, in: Proceedings of 2004 National Annual Software and Applications Conference (NASAC 2004), pp. 242–246 (in Chinese).