Software Reliability Modelling and Prediction with Hidden Markov Chains

JEAN-BAPTISTE DURAND

OLIVIER GAUDOIN

Institut National Polytechnique, Grenoble, France

Abstract. The purpose of this paper is to use the framework of hidden Markov chains for the modelling of the failure and debugging process of software, and the prediction of software reliability. The model parameters are estimated using the forward-backward EM algorithm and model selection is done with the BIC criterion. The advantages and drawbacks of this approach with respect to usual modelling are analyzed. Comparison is also done on real software failure data. The main contribution of hidden Markov chain modelling is that it highlights the existence of homogeneous periods in the debugging process, which allow one to identify major corrections or version updates. In terms of reliability predictions, the hidden Markov chain model performs well on average with respect to usual models, especially when the reliability is not regularly growing.

Keywords. Software reliability, reliability growth models, debugging, hidden Markov chains, EM, BIC.

1 Introduction: software reliability modelling

Let $X_i, i \geq 1$, be the times between successive failures of an item of software. After each failure, the software is corrected or not, and restarted. Debugging times are negligible or not taken into account. Let N_t be the number of failures between 0 and t. The failure process is equivalently one of the random processes $\mathbf{X} = \{X_i\}_{i\geq 1}$ or $\mathbf{N} = \{N_t\}_{t\geq 0}$. It is completely defined by the failure intensity:

$$\lambda_t = \lim_{dt\to 0} \frac{1}{dt} P\left(N_{t+dt} - N_t = 1 | \mathcal{F}_t\right) \tag{1.1}$$

where $\mathcal{F}_t = \sigma\left(\{N_s\}_{0\leq s\leq t}\right)$ is the internal filtration of the failure process. Equation (1.1) can be interpreted as follows: the future behaviour of the failure process is potentially dependent on its past.

A huge number of models for the failure process have been proposed in the last 30 years. These models are based on more or less sophisticated assumptions, and their goal is to estimate

present and future software reliability, based on the observation of past failures and corrections. For recent reviews, see Lyu (1996) and Pham (2000).

The most popular class of models is that of Non Homogeneous Poisson Processes (NHPP), for which the failure intensity is a continuous function of time. Among them, the most usual are the Power-Law Process (PLP - Duane, 1964), the Goel-Okumoto (GO, 1979) and the S-shaped (S - Yamada *et al.*, 1983) models. The main reason for the wide use of NHPP in software reliability is the simplicity of their use. However, the assumption of a continuous failure intensity is not realistic because debugging should induce a discontinuity.

Moreover, an important feature of software, and a major difference with hardware, is that items of software do not wear-out: if a piece of software is not modified, its ability to fail does not change, so the failure intensity between two debuggings should be constant. Here, the failure intensity is a step function. More sophisticated models are built following Littlewood's (1989) statement, according to which two sources of uncertainty exist in the failure behavior of software undergoing debugging. The first source of uncertainty is in the inputs: software inputs are chosen randomly in the input space according to the operational profile. The second source of uncertainty is the effect of debugging. It is logical to assume that the debugging of a fault at a given time depends only on the software state at this time, and not on its past states. This results in a Markovian modelling of the debugging process. All these assumptions lead to another class of models, sometimes called Markov Failure Rate (MFR) models (Soler 1988, Gaudoin 1990). For these models, there exists a Markov process $\mathbf{\Lambda} = \{\Lambda_i\}_{i\geq 1}$ such that, conditional on $\{\Lambda_i = \lambda_i\}_{i\geq 1}$, the times between failures X_i are independent and exponentially distributed (due to the no wear-out property) with respective parameters λ_i. Λ_i can be understood as the software failure rate after the $(i-1)^{\text{th}}$ debugging. A software reliability model in this framework is given by a model for the process $\{\Lambda_i\}_{i\geq 1}$.

The following assumptions have been made concerning the distribution of the process $\mathbf{\Lambda}$:

- The Λ_i are deterministic. Thus, the times between failures X_i are independent and exponentially distributed. The most famous of these models are the Jelinski-Moranda (JM, 1972) and Moranda geometric (MG, 1979) models.

- The Λ_i are independent and not identically distributed. This is for instance the case of the Littlewood-Verral (LV, 1973) model.

- $\forall i$, Λ_i is a function of Λ_{i-1} and of random variables representing the efficiency of $(i-1)^{\text{th}}$ debugging. This is the case of the models proposed in Gaudoin *et al.* (1994) and Gaudoin (1999).

The parameters of all these models can be estimated by maximum likelihood or other techniques. Then, it is possible to estimate present and future reliability. These models have been more or less successfully used on real software failure data.

However, all these models assume that there is a correction for each failure, and that the debugging efficiency is homogeneous in time. In practice, after software failures, computers are often rebooted without any correction. Debugging happens when a sufficiently large number of failures has occurred. When a software is in its operational life, there is a version update or introduction of a new release instead of debugging, but both concepts can be handled in the same way. Even if a correction is done after each failure, most of them are minor and there are sometimes major corrections, which can be considered as equivalent to version updates.

Software reliability data generally consist of a list of successive times between failures, and the information on whether a correction has been performed or not, or whether corrections are minor or major, is not available. Thus, it would be interesting to build software reliability models which could take this fact into account. This is the case of the hidden Markov chain (HMC) modelling.

Section 2 presents the modelling of the software failure and debugging process with hidden Markov chains. In section 3, model parameters are estimated using the forward-backward *Expectation Maximization* (EM) algorithm. Section 4 deals with the hidden state restoration using the Viterbi algorithm. Section 5 presents the choice of the number of hidden states and transition probability matrix with the BIC criterion. In section 6, the model predictive validity is assessed with the U-plot method. Finally, the HMC approach is applied to real software failure data and compared with usual software reliability growth models, from the points of view of knowledge of the debugging process and reliability prediction.

2 Modelling the software failure process with hidden Markov chains

In order to use the framework of hidden Markov chains, it is necessary to assume that the failure rate process $\mathbf{\Lambda} = \{\Lambda_i\}_{i\geq 1}$ takes values in a finite set. Let K be the cardinal number of this set and $\{\lambda^{(1)}, \ldots, \lambda^{(K)}\}$ be the set of possible values for the Λ_i. Then, $\mathbf{\Lambda}$ is a discrete-valued Markov chain. The assumptions on $\boldsymbol{X} = \{X_i\}_{i\geq 1}$ and $\mathbf{\Lambda}$ are as follows:

1. $\mathbf{\Lambda}$ is a discrete-valued Markov chain;

2. conditional on $\{\Lambda_i = \lambda_i\}_{i\geq 1}$, the times between failures $\{X_i\}_{i\geq 1}$ are independent;

3. conditional on $\{\Lambda_i = \lambda^{(j)}\}$, X_i has an exponential distribution with parameter $\lambda^{(j)}$.

Any process satisfying the three assumptions above is a hidden Markov chain. This name is due to the unknown values $\{\Lambda_i\}_{i\geq 1}$, which are not directly observable, hence *hidden*.

Each trajectory of process $\boldsymbol{X}$ can be split into homogeneous zones, each zone corresponding to one value $\lambda^{(j)}$ of the failure rate process. In a given zone, the failure rate remains constant. Discrete jumps appear in the failure process when there is a transition of the failure rate process at i^{th} failure, from $\Lambda_i = \lambda^{(j)}$ to $\Lambda_{i+1} = \lambda^{(l)}$, where $j \neq l$.

In the software test period, the homogeneous zones can be interpreted as periods where no correction has occurred after failures, or where the corrections introduced were minor and did not improve significantly the failure rate. The jumps correspond to corrections in the first case and major corrections in the second. In the software operational life, the transitions between zones can be interpreted as introductions of version updates or new releases. Consequently, this model fulfills the requirements stated in section 1.

The advantage of the HMC model with regard to NHPP models is that it takes into account the discontinuities in failure intensity caused by the debugging and the no wear-out property of software. The advantage with regard to MFR models is that there is not necessarily a correction after each failure, which leads to the existence of homogeneous periods.

The study of transitions between these homogeneous zones is similar to the classical change-point detection problem. In this framework, the HMC model appears as a particular case of a state-space model, described in Basseville and Nikiforov (1993) as a usual approach for the detection of abrupt changes. In reliability, change-point methods are generally used to detect a modification in the behavior of a system, for instance the beginning of ageing (Zhao, 1993). To our knowledge, the problem of detecting homogeneous zones in software reliability has never been addressed before.

A hidden Markov chain is defined by the following parameters:

1. The distribution of the initial state Λ_1, given by $\pi_j = P(\Lambda_1 = \lambda^{(j)})$, $1 \leq j \leq K$.

2. The transition probabilities $P(\Lambda_{i+1} = \lambda^{(l)} | \Lambda_i = \lambda^{(j)}) = p_{jl}^{(i)}$, $i \geq 1, 1 \leq j \leq K, 1 \leq l \leq K$. In what follows, we assume that the Markov chain $\boldsymbol{\Lambda}$ is homogeneous, *i.e.* the transition probabilities do not depend on i. Thus the transition parameters amount to the matrix P defined by the p_{jl}.

3. The values $(\lambda^{(1)}, \ldots, \lambda^{(K)})$ of the failure rates.

In sections 3 and 4 as well as in the current section, the number K of possible failure rates is considered as known. Let $\boldsymbol{\eta}$ be the set of all parameters of a HMC model. In order to assess and predict software reliability, $\boldsymbol{\eta}$ has to be estimated. Sections 3, 4 and 5 present the used estimation method and related features such as the restoration of the hidden states and the choice of K.

3 Parameter estimation

For any sequence $\{z_i\}_{i\geq 1}$ and any couple of integers (i,j) such that $i < j$, let $\boldsymbol{z}_i^j$ denote $(z_i, \ldots, z_j)$. We use the general notation $P()$ to denote either a probability mass function or a probability density function, the true nature of $P()$ being obvious from the context. This convention makes the notation simpler, when dealing with mixed distributions.

We assume that the n first interfailure times $\boldsymbol{x}_1^n$ of a software are observed. We want to use the HMC model in order to assess present and future reliability at the last failure time. The first step is the estimation of the HMC parameters.

The HMC model is a typical case where the complete data $\boldsymbol{y}_1^n$ can be split into the observed data $\boldsymbol{x}_1^n$ and the missing data $\boldsymbol{\lambda}_1^n$ with $\boldsymbol{y}_1^n = (\boldsymbol{x}_1^n, \boldsymbol{\lambda}_1^n)$. The presence of incomplete data often complicates the estimation of the parameters $\boldsymbol{\eta}$ by likelihood maximization. Thus, we resort to the EM algorithm, dedicated to the likelihood maximization in the context of missing values (McLachlan and Krishnan, 1997). This iterative algorithm starts from an initial value $\boldsymbol{\eta}^{(0)}$ of the parameters and creates a sequence $\{\boldsymbol{\eta}^{(m)}\}_{m\geq 0}$ resulting in non-decreasing likelihoods. At each iteration m, it proceeds as follows:

- *Expectation (E) step* - determination of the Q function defined by

$$Q(\boldsymbol{\eta}, \boldsymbol{\eta}^{(m)}) = E_{\boldsymbol{\eta}^{(m)}}[\log P_{\boldsymbol{\eta}}(\boldsymbol{\Lambda}_1^n, \boldsymbol{X}_1^n = \boldsymbol{x}_1^n) | \boldsymbol{X}_1^n = \boldsymbol{x}_1^n]; \tag{3.1}$$

- *Maximization (M) step* - maximization of $Q(\boldsymbol{\eta}, \boldsymbol{\eta}^{(m)})$ with respect to $\boldsymbol{\eta}$. Then

$$\boldsymbol{\eta}^{(m+1)} = \arg\max_{\boldsymbol{\eta}} Q(\boldsymbol{\eta}, \boldsymbol{\eta}^{(m)}). \tag{3.2}$$

The EM algorithm has the following properties: the sequence $\{\boldsymbol{\eta}^{(m)}\}_{m\geq 0}$ converges to the consistent solution of the likelihood equations when $\boldsymbol{\eta}^{(0)}$ is close to the optimal solution. However, when the mixture components $\{\lambda^{(1)}, \ldots, \lambda^{(K)}\}$ are poorly separated, the estimates obtained strongly depend on the initial value $\boldsymbol{\eta}^{(0)}$ and the convergence rate can be crippling. To cope with this drawback, we start from several initial values and run a few iterations, the final value maximizing the likelihood being kept as a starting value for one further EM run. More precisely,

we consider three initial values determined at random and we run 50 iterations for each value. Thus we obtain three estimates. The one which has maximal likelihood is used as starting position for the EM algorithm. We stop the algorithm when 1,000 iterations have been done, or when the relative increase of the log-likelihood is below a threshold ε, for instance $\varepsilon = 10^{-6}$. For 80 % of the sequences in the data set and for any value of K, this threshold was reached before 800 iterations.

The *M step* is given by the following re-estimation formulae (see Ephraim and Merhav, 2002):

$$\hat{\pi}_j = \gamma_1(j) \tag{3.3}$$

$$\hat{p}_{jl} = \frac{\sum_i \xi_i(j,l)}{\sum_i \gamma_i(j)} \tag{3.4}$$

$$\hat{\lambda}^{(j)} = \frac{\sum_i \gamma_i(j)}{\sum_i \gamma_i(j) x_i} \tag{3.5}$$

where

$$\gamma_i(j) = E_{\boldsymbol{\eta}^{(m)}}[\mathbb{1}_{\{\Lambda_i=\lambda^{(j)}\}}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n] = P_{\boldsymbol{\eta}^{(m)}}(\Lambda_i = \lambda^{(j)}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n) \tag{3.6}$$

$$\begin{aligned}\xi_i(j,l) &= E_{\boldsymbol{\eta}^{(m)}}[\mathbb{1}_{\{\Lambda_i=\lambda^{(l)},\Lambda_{i-1}=\lambda^{(j)}\}}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n] \\ &= P_{\boldsymbol{\eta}^{(m)}}(\Lambda_i = \lambda^{(l)}, \Lambda_{i-1} = \lambda^{(j)}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n).\end{aligned} \tag{3.7}$$

These conditional distributions can be computed by inductive algorithms running along the chain, first from the origin time forward to the future and second from the final time backward to the past. This caused the EM algorithm for the hidden Markov chain model to be called the *forward-backward algorithm.*

In what follows, the parameters $\boldsymbol{\eta}^{(m)}$ are fixed and we write $P = P_{\boldsymbol{\eta}^{(m)}}$ to make the notation simpler. The forward recursion is based on the so-called forward quantities $\alpha_i(j) = P(\boldsymbol{X}_1^i = \boldsymbol{x}_1^i, \Lambda_i = \lambda^{(j)})$, initialized at $i = 1$ by $\alpha_1(j) = \pi_j f_{\lambda^{(j)}}(x_1)$ and computed inductively by

$$\alpha_{i+1}(l) = \sum_j p_{jl}\alpha_i(j) f_{\lambda^{(l)}}(x_{i+1}). \tag{3.8}$$

As a byproduct of this recursion, the likelihood is given by $f_{\boldsymbol{\eta}}(\boldsymbol{x}_1^n) = \sum_j \alpha_n(j)$.

The backward recursion is based on the backward quantities $\beta_i(j) = P(\boldsymbol{X}_{i+1}^n = \boldsymbol{x}_{i+1}^n|\Lambda_i = \lambda^{(j)})$, initialized at $i = n-1$ by $\beta_{n-1}(j) = \sum_l p_{jl} f_{\lambda^{(l)}}(x_n)$ and computed inductively by

$$\beta_i(j) = \sum_l p_{jl}\beta_{i+1}(l) f_{\lambda^{(l)}}(x_{i+1}). \tag{3.9}$$

Finally, the conditional probabilities required by the EM algorithm are given by

$$\gamma_i(j) = \frac{\alpha_i(j)\beta_i(j)}{\sum_l \alpha_n(l)} \tag{3.10}$$

$$\xi_i(j,l) = \frac{\beta_{i+1}(l)p_{jl}f_{\lambda^{(l)}}(x_{i+1})\alpha_i(j)}{\sum_{j'} \alpha_n(j')}. \tag{3.11}$$

It is well-known that the *forward-backward algorithm* above is subject to underflow when n is moderately large. This is why we use the so-called *forward-backward algorithm for posterior probabilities* of Devijver (1985), which basically computes $P(\Lambda_i = \lambda^{(j)}|\boldsymbol{X}_1^i = \boldsymbol{x}_1^i)$ instead of $P(\boldsymbol{X}_1^i = \boldsymbol{x}_1^i, \Lambda_i = \lambda^{(j)})$ and is thus immune to underflow.

The EM algorithm has the following interpretation, in the present context. First, the maximum likelihood estimates in the case of complete data, obtained by maximization of the completed likelihood $P_{\boldsymbol{\eta}}(\boldsymbol{\Lambda}_1^n = \boldsymbol{\lambda}_1^n, \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$, are as follows:

$$\tilde{\pi}_j = \mathbb{1}_{\{\Lambda_1=\lambda^{(j)}\}} \tag{3.12}$$

$$\tilde{p}_{jl} = \frac{\sum_i \mathbb{1}_{\{\Lambda_i=\lambda^{(l)},\Lambda_{i-1}=\lambda^{(j)}\}}}{\sum_i \mathbb{1}_{\{\Lambda_{i-1}=\lambda^{(j)}\}}} \tag{3.13}$$

$$\tilde{\lambda}^{(j)} = \frac{\sum_i \mathbb{1}_{\{\Lambda_i=\lambda^{(j)}\}}}{\sum_i \mathbb{1}_{\{\Lambda_i=\lambda^{(j)}\}} x_i}. \tag{3.14}$$

The estimates of the transition probabilities can be interpreted as counts of the transition numbers. The estimates of $\lambda^{(j)}$ are the classical estimates for an exponential distribution, where the mean is computed only on the variables X_i with $\Lambda_i = \lambda^{(j)}$, *i.e.* the data in j^{th} state. Now, we can see that in the re-estimation formulae (3.3) to (3.5) of the EM algorithm, the unknown random quantities involved through the indicator functions are replaced by their expectations, conditional on $\{\boldsymbol{X}_1^n = \boldsymbol{x}_1^n\}$, with respect to the distribution $P_{\boldsymbol{\eta}^{(m)}}$.

In fact, we consider a slightly simplified model. On the one hand, it can be shown that the maximum likelihood estimate of π is a Dirac distribution $\mathbb{1}_{\{\lambda=\lambda^{(l)}\}}$ if only one sequence is used to estimate the parameters, which is the case in this paper. On the other hand, the distribution $P_{\boldsymbol{\eta}}$ remains invariant to any permutation of the parameters $\boldsymbol{\eta}$ - corresponding to a relabelling of the hidden states - as usual in mixture models. As a consequence, we can make the assumption that $\pi_j = \mathbb{1}_{\{\lambda=\lambda^{(1)}\}}(\lambda^{(j)})$. This means that the failure rate at $i = 1$ is $\Lambda_1 = \lambda^{(1)}$ with probability one. In practice, we estimate the parameters without taking this constraint into consideration. Then we relabel the states in such a way that $\pi_j = \mathbb{1}_{\{\lambda=\lambda^{(1)}\}}(\lambda^{(j)})$.

4 Restoration of the hidden states

It has been emphasized in section 2 that one main contribution of the hidden Markov chain modelling in software reliability lies in the possibility of interpreting the sequences of successive equal states as homogeneous zones in the failure process.

We explain in this section how the unknown failure rates can be restored in the context of hidden Markov chain models. One wants to use a model specified by parameter $\boldsymbol{\eta}$, considered as known (for example estimated by maximum likelihood as described in section 3), to give a value to the unknown state sequence $\boldsymbol{\lambda}_1^n$. A natural way to restore the hidden states is to compute their most likely value, conditional on $\boldsymbol{x}_1^n$. The resulting algorithm is called a MAP algorithm (maximum a posteriori). It consists in computing $\arg\max_{\boldsymbol{\lambda}_1^n} P_{\boldsymbol{\eta}}(\boldsymbol{\Lambda}_1^n = \boldsymbol{\lambda}_1^n | \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$, or equivalently $\arg\max_{\boldsymbol{\lambda}_1^n} P_{\boldsymbol{\eta}}(\boldsymbol{\Lambda}_1^n = \boldsymbol{\lambda}_1^n, \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$. This can be done by the so-called Viterbi algorithm, based on dynamic programming (see Ephraim and Merhav, 2002). The Viterbi algorithm is similar to the forward recursion of section 3 but the summing procedure is replaced by a maximization. This recursion is based on the quantities $\delta_i(j) = \max_{\boldsymbol{\lambda}_1^{i-1}} P_{\boldsymbol{\eta}}(\boldsymbol{\Lambda}_1^{i-1} = \boldsymbol{\lambda}_1^{i-1}, \Lambda_i = \lambda^{(j)}, \boldsymbol{X}_1^i = \boldsymbol{x}_1^i)$, initialized at $i = 1$ by $\delta_1(j) = \pi_j f_{\lambda^{(j)}}(x_1)$ and computed inductively by

$$\delta_{i+1}(l) = \max_j p_{jl}\delta_i(j) f_{\lambda^{(l)}}(x_{i+1}). \tag{4.1}$$

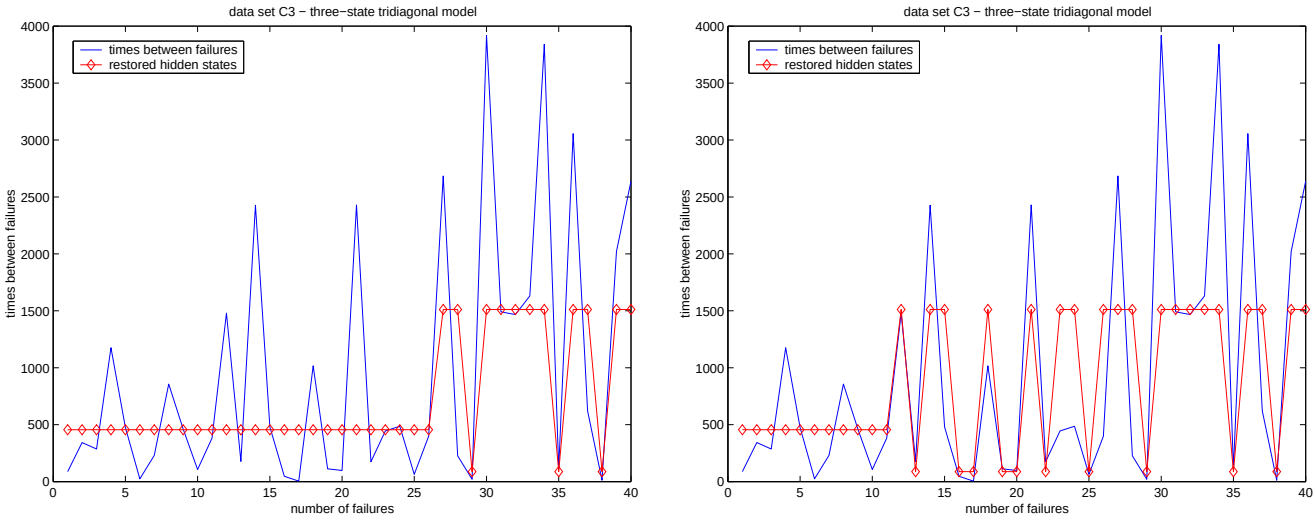
The maximal value of the completed likelihood is given at $i = n$ by $P^* = \max_j \delta_n(j)$, and the restored final state is $\lambda_n^* = \arg\max_j \delta_n(j)$. To retrieve the optimal sequence of failure rates, it is necessary to store for each time i and each state j the optimal failure rate corresponding to the immediate past $i-1$. Thus, the failure rate sequence is determined recursively by a backward procedure from the optimal final state towards the initial state.

Another strategy for the hidden state restoration consists in computing

$$\bar{\lambda}_i = \arg\max_j P(\Lambda_i = \lambda^{(j)} | \boldsymbol{X}_1^n = \boldsymbol{x}_1^n). \tag{4.2}$$

It leads to the maximization of the expected number of correct failure rates. This local method, however, does not take into account the whole sequence of failure rates. We will consider in section 5 some models with forbidden transitions (*i.e.* some transition probabilities equal to 0). A major drawback of restoring the failure rate sequence without using the Viterbi algorithm appears when we obtain a forbidden sequence as a result. Furthermore, the outputs of both algorithms do not generally coincide. Generally, using the Viterbi algorithm tends to give longer homogeneous zones, thus favouring the interpretation of the hidden states.

The comparison between original data and restored hidden states is easy since the expected interfailure time when $\Lambda_i = \lambda^{(j)}$ is $1/\lambda^{(j)}$.



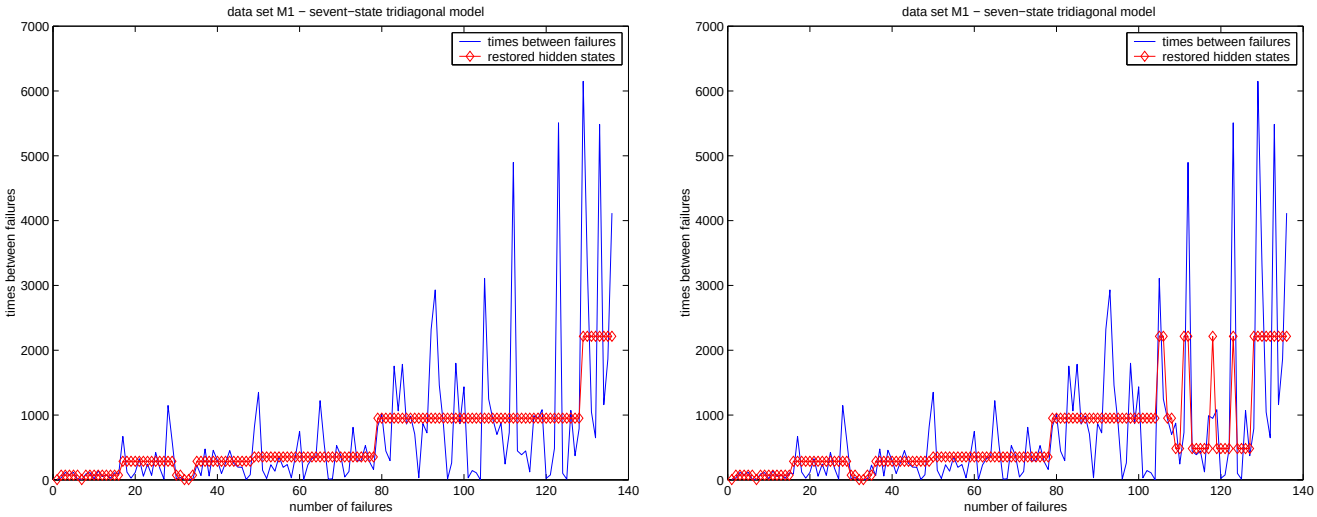
(a) *The globally most likely sequence.* (b) *The individually most likely states.*

Figure 1: Comparison of two methods for the hidden state restoration: the Viterbi algorithm favours homogeneous zones.

Figure 1(a) shows the times between failures for a real software failure data set (C3 in Gaudoin, 1990) superimposed on the optimal state sequence restored by the Viterbi algorithm, using a three-state model with parameters $\hat{\boldsymbol{\eta}}$ estimated by maximum likelihood. Figure 1(b) shows the same real data set superimposed on the optimal state sequence restored using the probabilities $P(\Lambda_i = \lambda^{(j)} | \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$ and the same parameters $\hat{\boldsymbol{\eta}}$. In this case, when the Viterbi algorithm is used, the resulting sequence has longer plateaux than that obtained by local state restoration, since the dependencies between the states are fully taken into account.

Figure 2(a) shows the times between failures for another real data set (M1 in Musa, 1979) superimposed on the optimal state sequence restored by the Viterbi algorithm, using a seven-state model having a tridiagonal transition matrix with parameters estimated by maximum likelihood. Figure 2(b) shows the same real data set superimposed on the optimal state sequence restored using the probabilities $P(\Lambda_i = \lambda^{(j)} | \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$ and the same parameters. The model assumes that any transition from state 5 (characterized by a failure rate of $\lambda^{(5)} \approx 950^{-1}$) to state 7 (characterized by a failure rate of $\lambda^{(7)} \approx 480^{-1}$) is forbidden. See section 5 for more details about assumptions on forbidden transitions. We can see in figure 2 that whereas this constraint is satisfied by the global restoration (or Viterbi) algorithm, it is not satisfied by the local restoration algorithm which allows a transition from state 5 to state 7, at $i = 108$.

Thus, we choose the Viterbi algorithm for the restoration of the hidden states.



(a) *The globally most likely sequence.* (b) *The individually most likely states.*

Figure 2: Comparison of two methods for the hidden state restoration: the Viterbi algorithm respects the constraints induced by the transition probability matrix

5 Choice of the hidden state number and transition matrix

In the sections above, the number of possible failure rates K has been considered as known. Clearly, K is not known in practice and has to be estimated. This quantity plays a peculiar role in the model, since the dimension of parameter $\boldsymbol{\eta}$ depends on K.

For the choice of K, we use the BIC criterion (Bayesian Information Criterion - Kass and Raftery, 1995):

$$BIC(K) = \log(P_{\hat{\boldsymbol{\eta}}_K}(\boldsymbol{x}_1^n)) - \frac{\nu_K}{2}\log(n) \tag{5.1}$$

where $\log(P_{\hat{\boldsymbol{\eta}}_K}(\boldsymbol{x}_1^n))$ is the maximum log-likelihood for the hidden Markov chain with K hidden states and where ν_K is the number of independent parameters in $\boldsymbol{\eta}_K$. ν_K is the sum of:

- the number K of parameters $\lambda^{(l)}$ for the exponential conditional distributions;
- the number $K(K-1)$ of transition probabilities, since each row of the transition probability matrix sums to one.

The BIC criterion is composed of a term measuring the fit between the data and the model (namely, the log-likelihood) and of a term penalizing complex models. Thus, maximizing this criterion hopefully leads to selecting models offering a compromise between fit and complexity. Furthermore, from Gassiat (2002), criteria based on the marginal log-likelihood with the same penalization as that of BIC are proved to be consistent. On the contrary, the AIC criterion tends to overestimate the number of hidden states in the case of mixtures, as stated in Cutler and Windham (1993). In a Bayesian framework, model selection can be performed using reversible

jump MCMC methods (see Robert *et al.*, 2000). The comparison of likelihood and Bayesian analysis of mixtures has been addressed by Aitkin (2001).

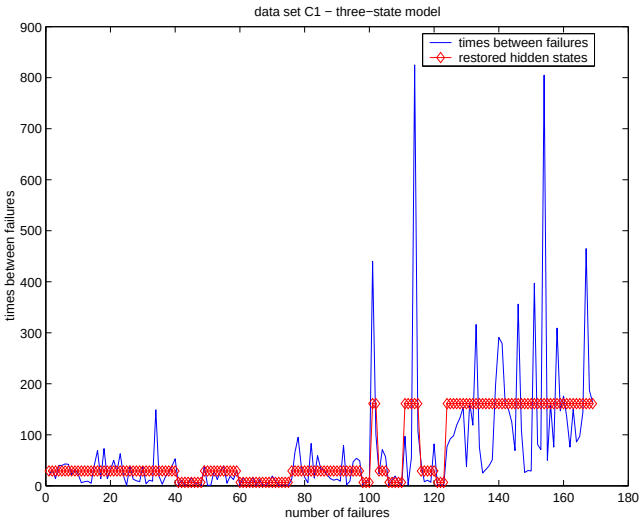
In practice, we set a maximal value K_{max} for K, as well as a minimal value K_{min}. Then we estimate the parameters by the EM algorithm (see section 3) for each value K between K_{min} and K_{max} and we keep the model maximizing $BIC(K)$.

The second issue on model selection concerns the transcription of assumptions on the debugging process into the transition probability matrix P. As shown in section 2, the hidden states can be interpreted as steps in the software debugging process, since the states correspond to discrete values of the failure rate. Until now, the assumption that all kinds of software corrections can occur has been made implicitly. This is the case when no particular assumption is made regarding the transition probability matrix, which means that any improvement or deterioration of the software failure rate is possible.

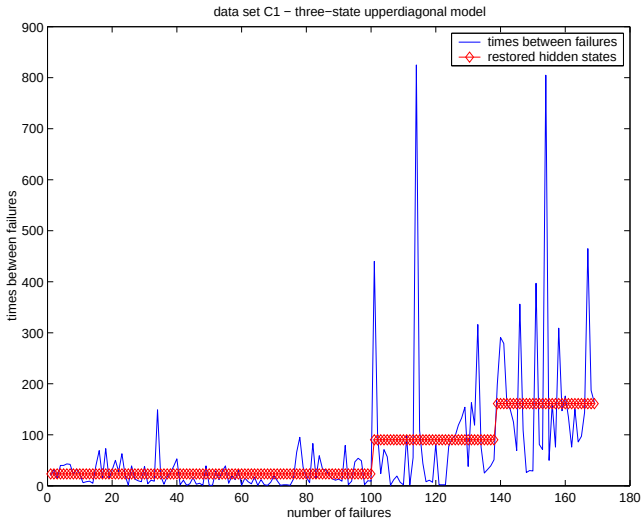
Another possibility is to consider a debugging process where each homogeneous zone is visited only once. This is the case of pure reliability growth models, where each debugging reduces the software failure rate. Consequently, any transition from a hidden state to a previously visited hidden state must be forbidden. This assumption implies an upper diagonal transition probability matrix. Under this assumption the failure rates are ordered in accordance to the transition probability matrix.

Moreover, it is important in practice to take into account the possibility of imperfect debugging. This means that the return to a previously visited state must be allowed. The easiest way to do so is to allow two transitions for each state (except the initial and final ones): one to the last previously visited state, and one to the next new visited state. This assumption implies a tridiagonal transition probability matrix. As for upper diagonal transition probability matrices, tridiagonal transition matrices induce an order on the failure rates.

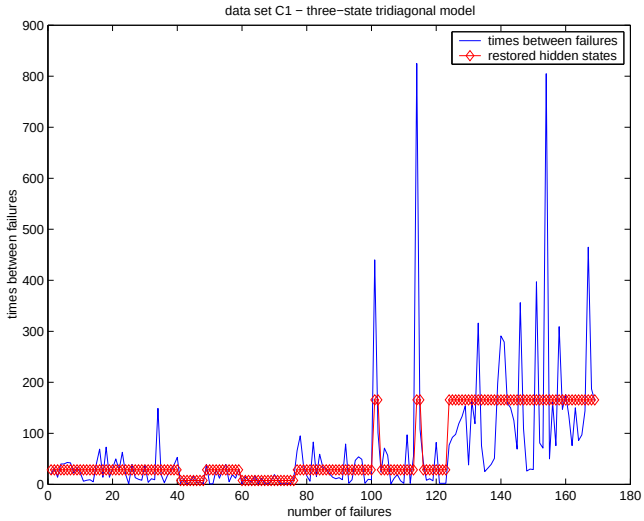
Figure 3 illustrates the effect of different types of transition probability matrix on model interpretation. These essentially lead to different models. The differences appear in the parameters, but they also translate into the restoration of the hidden sequence by the Viterbi algorithm. Figure 3(a) shows the times between failures for a real data set (C1 in Gaudoin, 1990) superimposed on the optimal state sequence restored by the Viterbi algorithm, using a three-state model with an unconstrained transition matrix with parameters estimated by maximum likelihood. In Figure 3(b), a model with an upper diagonal transition matrix has been used, thus leading to new estimates for the parameters (still by maximum likelihood) and a new restored sequence of failure rates. In Figure 3(c), a model with a tridiagonal transition matrix has been used. Figure 3(b) illustrates the fact that a model with an upper diagonal transition probability matrix for-



(a) *Model with no constraint on the transition probability matrix.*



(b) *Model with an upper diagonal transition probability matrix.*



(c) *Model with a tridiagonal transition probability matrix.*

Figure 3: Comparison of the three types of transition probability matrix.

bids any return to a previously visited state. This makes the model interpretation easier, as far as the research of the homogeneous periods is concerned, since transitions between failure rates are accepted less easily and in an irreversible way. It can also be seen from figures 3(a) and (c) that adding milder constraints than an upper diagonal transition probability matrix results in the suppression of some spurious transitions.

In conclusion, we are now facing a new model selection problem where each competing model is characterized by the structure of its transition probability matrix. Once again we can resort to the BIC criterion to select a model, when the difference of complexity is taken into account for each candidate model. The number ν_K of independent parameters to consider for model selection is:

- in absence of any assumption on the transition probability matrix P, $\nu_K = K + K(K-1) = K^2$;
- for an upper diagonal transition probability matrix, $\nu_K = K + (K-1) = 2K - 1$;
- for a tridiagonal transition probability matrix, $\nu_K = K + (2K-2) = 3K - 2$.

6 Predictive validity and model comparison

The quality of reliability predictions provided by the hidden Markov chain model has to be assessed and compared with those given by the usual software reliability growth models presented in section 1. The usual method for comparing software reliability predictions is the so-called *U-plot* method (Keiller *et al.* 1983; Abdel-Ghali *et al.* 1986).

For any model with parameter $\boldsymbol{\theta}$, let $P_{\hat{\boldsymbol{\theta}}_i}(X_i \leq x | \boldsymbol{X}_1^{i-1} = \boldsymbol{x}_1^{i-1})$ be the predicted Cumulative Distribution Function (CDF) of the next time to failure X_i given the first $i-1$ interfailure times, where $\hat{\boldsymbol{\theta}}_i$ is an estimator of $\boldsymbol{\theta}$ based on $\boldsymbol{x}_1^{i-1}$. The idea of the method is to compute the $u_i = P_{\hat{\boldsymbol{\theta}}_i}(X_i \leq x_i | \boldsymbol{X}_1^{i-1} = \boldsymbol{x}_1^{i-1})$. If the model is appropriate and the estimation is of good quality, then the u_i should be close to a sample of the uniform distribution over $[0,1]$. This closeness or predictive validity is measured by the Kolmogorov-Smirnov distance KS between the empirical CDF of the u_i and the true uniform CDF, which is $F(x) = x$ on $[0,1]$. The *U-plot* is the plot of the empirical CDF of the u_i. If several models are competing, the "best model" is the one for which KS is the smallest.

For the hidden Markov chain model, it is easy to prove that:

$$P(X_i \leq x | \boldsymbol{X}_1^{i-1} = \boldsymbol{x}_1^{i-1}) = \frac{\sum_l F_{\lambda^{(l)}}(x) \sum_k \alpha_{i-1}(k) p_{kl}}{P_{\hat{\boldsymbol{\eta}}_i}(\boldsymbol{x}_1^{i-1})} \tag{6.1}$$

where $F_{\lambda^{(l)}}$ denotes the CDF of the exponential distribution with parameter $\lambda^{(l)}$.

Figure 4 shows the U-plot for data set M40 in Musa (1979), using a three-state model with upper diagonal transition probability matrix. In this example, $KS = 1.375$.

7 Application and conclusion

The hidden Markov chain model has been used on two groups of real software failure data sets. The first group consists of times between failures of nine pieces of American control-command software (denoted M1, M2, M3, M4, M6, M14, M17, M27 and M40) in the test period and

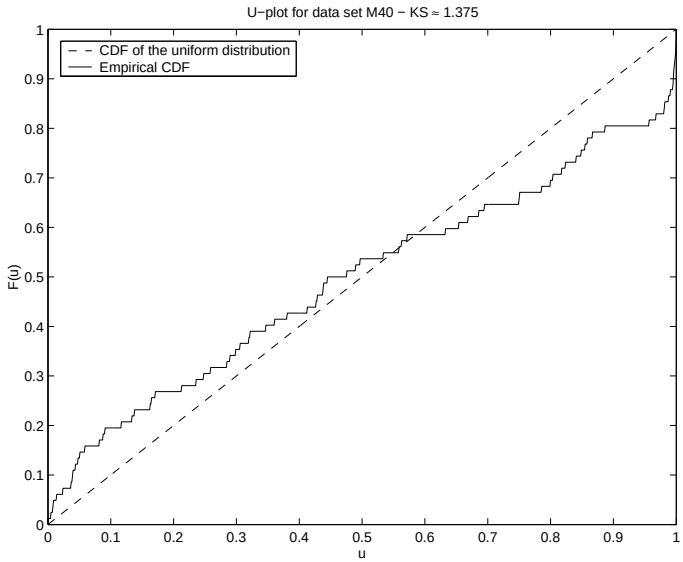


Figure 4: U-plot for the HMC model

operational life (Musa, 1979)[1]. The second group consists of times between failures of four pieces of complex French software (denoted C1, C2, C3, C4) in the test period (Gaudoin, 1990). In both cases, calendar time is used. The identification and selection of hidden Markov chains have been performed with novel MATLAB routines[2].

First of all, we present a complete treatment of one of these data sets, M40, for which $n = 101$. The first step is the choice of the number of hidden states and transition probability matrix type. The minimal and maximal values of K are set to $K_{min} = 1$ and $K_{max} = 7$.

Figure 5 shows that the BIC criterion is maximum for a model with three hidden states and an upper diagonal transition probability matrix. For this model, the parameter estimates are:

$$\left[\ \hat{\lambda}^{(1)} \quad \hat{\lambda}^{(2)} \quad \hat{\lambda}^{(3)}\ \right] = 10^{-4} * \left[\ 0.5035 \quad 0.0908 \quad 0.0175\ \right] \tag{7.1}$$

$$\hat{P} = \begin{bmatrix} 0.9809 & 0.0191 & 0 \\ 0 & 0.9502 & 0.0498 \\ 0 & 0 & 1.0000 \end{bmatrix} \tag{7.2}$$

Figure 5 presents the restoration of the hidden states. Three homogeneous periods clearly appear in this figure.

Finally, the HMC model is compared with some of the most usual software reliability models, using the U-plot method. The MG model has the best predictive validity, followed by PLP and the HMC model. The HMC model performs much better than GO, S and JM models for this data set (see table 1).

[1]data set available at `http://www.dacs.dtic.mil/databases/sled/swrel.shtml`
[2]routines available at `http://www.inrialpes.fr/is2/people/jdurand/Html/Logiciels/logiciel.html`

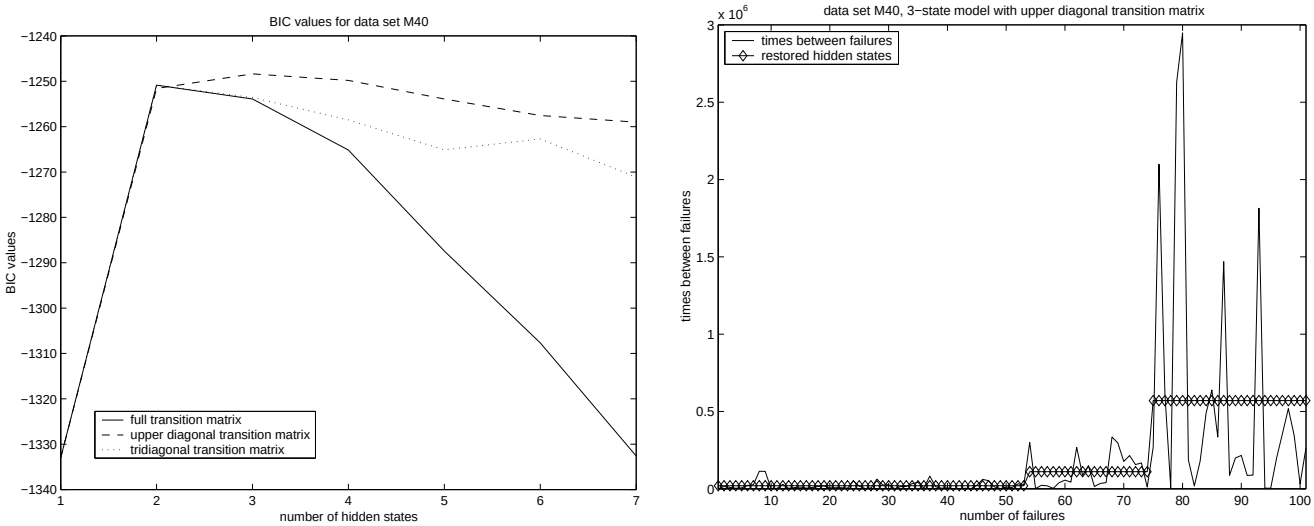


Figure 5: Model selection with BIC and restoration of the hidden states for M40

Table 1: KS distances for all data sets

Data set	M1	M2	M3	M4	M6	M14	M17	M27	M40	C1	C2	C3	C4	
matrix	u	f	u	u	f	f	u	u	u	f	t	f	u	Mean
K	3	2	2	2	2	2	2	2	3	2	3	1	4	rank
HMC	1.82	1.74	1.68	0.75	0.77	0.67	0.74	0.90	1.38	0.91	2.07	1.46	3.39	2.85
PLP	1.40	1.31	1.51	0.66	1.60	1.50	0.93	1.32	1.32	1.70	1.67	1.22	2.98	2.77
GO	1.59	1.16	1.15	0.85	1.67	1.70	0.75	1.00	2.42	1.11	4.28	0.64	2.16	3.08
S	4.43	2.49	2.60	1.96	2.56	0.99	0.99	2.43	3.34	2.93	7.64	1.58	3.28	5.00
JM	5.86	3.60	NC	NC	3.11	0.89	0.57	2.60	4.78	6.29	7.37	0.71	NC	4.92
MG	1.21	0.81	0.77	0.76	1.66	1.47	0.75	0.88	1.22	1.29	1.95	0.75	1.92	2.15

A similar study has been done on the thirteen data sets. Table 1 gives, for all data sets, the type of transition probability matrix ('f' for full, 'u' for upper diagonal, 't' for tridiagonal) and number of hidden states K selected by the BIC criterion, and the KS distances computed in the U-plot method for the selected HMC and 5 usual software reliability models. In the table, NC means that the algorithm which computes the KS distance did not converge. This happened three times, only for the JM model. The table also gives the mean rank of each model for all data sets. The details of the study, with plots of the restoration of the hidden states in HMC models for all data sets, can be found in Durand and Gaudoin (2003).

The main results of this study are the following ones.

- For all data sets, the selected number of hidden states is very small, from 1 to 4. This result was unexpected because we thought we would find more homogeneous periods in the data. Maybe the BIC criterion favours too small values of K. However, if K is set

to a rather large number such as 5, it happens that a lot of failure rates are estimated by very close values. This is illustrated in figure 6 for the data set C4: the first two states are poorly separated, and the values of $BIC(k)$ for $k = 2$ and 3 are hardly lower than $BIC(4)$, so that models with fewer hidden states also fit the data properly. For most of the data sets, models with more than 4 states result in spurious transitions, so a model with fewer states is preferable. This means that for the considered pieces of software, very few major corrections seem to have occurred during the debugging process.

- The choice of initial values for the EM algorithm has a slight effect on the maximum BIC value. In some situations, it can lead to a change of the selected model but it does not affect significantly either the KS distance or the restoration of the hidden states.

- From the BIC point of view, there are some significant differences between all kinds of transition probability matrices. Logically, the upper diagonal matrix gives the best result for data sets exhibiting almost pure reliability growth. For data sets for which reliability is sometimes decreasing (probably due to imperfect debugging), full or tridiagonal matrices are better (M2, M6, M14, C1, C2). This fact is confirmed on simulated data sets: when data are simulated according to a pure reliability growth model, the upper diagonal matrix is always chosen by the BIC criterion.

- The HMC model has the best predictive validity for 3 data sets among 13 (M6, M14, C1). For the ten others, the HMC performs well on average: it is often close to the best model, and in most cases largely better than the worse of them. The mean ranks in table 1 indicate that model MG clearly provides the best reliability predictions for these data sets. PLP, HMC and GO have a similar performance. JM and S have a poor predictive validity, except for a small number of data sets. In fact, usual models perform better than HMC when there is a regular reliability growth, and HMC is the best when there are some significant imperfect debuggings.

- Long homogeneous periods are detected by the restoration of hidden states for almost all data sets, leading to a clear interpretation in terms of identification of major corrections. For some data sets (M6, M14, C3), such periods are not detected, so the conclusion is that no significant corrections have occurred. In case of returns to previous states due to imperfect debugging (C2) or existence of long and very distinct homogeneous periods (C4, see figure 6), the HMC model is more appropriate than usual models. However, the predictive validity of HMC in these cases can be very poor. This is due to the fact that in the U-plot method, the estimates are computed sequentially. At each step, the nature

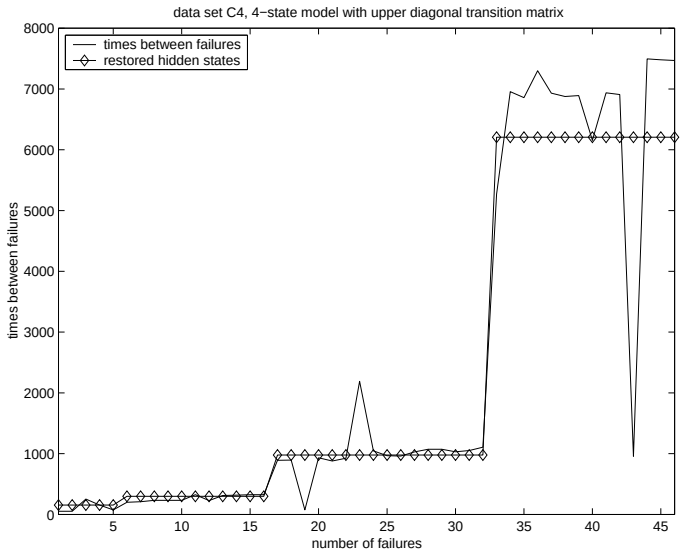


Figure 6: Restoration of the hidden states for C4

of the hidden states changes, which makes prediction of the future time to failure much more difficult than it seems. In this case the goodness-of-fit and predictive validity are not necessarily correlated for the HMC model.

Finally, the hidden Markov chain approach provides a new way of modelling software reliability which can take into account some features that are ignored by usual models, such as the existence of homogeneous periods and the possibility of imperfect debugging. Not surprisingly, the HMC model performs best for data exhibiting these features. For other kinds of data, usual models can be more appropriate. Even for these data, the HMC approach is still interesting since it clearly identifies the location of major corrections in the debugging process.

A possible extension of this approach is to consider that inside the homogeneous periods, the times between failures do not necessarily have a constant failure rate, but can be distributed according to a certain kind of reliability growth model. Such an extension leads to mixtures of usual models with Markovian transitions. For example, a "hidden Markov Moranda geometric model" corresponds to the assumption that conditional on $\{\Lambda_i = \lambda^{(j)}\}$, X_i has an exponential distribution with parameter $\lambda^{(j)} c_j^{i-1}$.

Acknowledgment: The authors are grateful to Gilles Celeux and Jean-Louis Soler who initiated this work.

References

A. Abdel-Ghali, P. Chan and B. Littlewood, "Evaluation of competing software reliability predictions," *IEEE Transactions on Software Engineering*, vol. SE-12, 9, pp. 950-967, 1986.

M. Aitkin, "Likelihood and Bayesian analysis of mixtures," *Statistical Modelling*, vol. 1, pp. 287-304, 2001.

M. Basseville and I.V. Nikiforov, *Detection of Abrupt Changes, Theory and Application*, Prentice-Hall, Englewood Cliffs, 1997.

A. Cutler and M.P. Windham, "Information-Based Validity Functionals for Mixture Analysis," *Proceedings of the first US-Japan Conference on the Frontiers of Statistical Modelling*, Amsterdam, Kluwer (Academic Publishers), pp. 149-170, 1993.

P. A. Devijver, "Baum's forward-backward algorithm revisited," *Pattern Recognition Letters*, vol. 3, pp. 369-373, 1985.

J. T. Duane, "Learning curve approach to reliability monitoring," *IEEE Transactions on Aerospace*, vol. AS-2, 2, pp. 563-566, 1964.

J.-B. Durand and O. Gaudoin, "Software reliability modelling and prediction with hidden Markov chains," *INRIA Research Report*, RR-4747, 2003.

Y. Ephraim and N. Merhav "Hidden Markov Processes," *IEEE Transactions on Information Theory*, vol. 48, pp. 1518-1569, 2002.

E. Gassiat, "Likelihood ratio inequalities with application to various mixtures," *Annales de l'Institut Henri Poincaré*, vol. 38, 6, pp. 268-278, 2002.

O. Gaudoin, *Outils statistiques pour l'évaluation de la fiabilité des logiciels*, PhD, Joseph Fourier University, Grenoble (in French), 1990.

O. Gaudoin, C. Lavergne and J. L. Soler, "A generalized geometric de-eutrophication software-reliability model," *IEEE Transactions on Reliability*, vol. R-44, 4, pp. 536-541, 1994.

O. Gaudoin, "Software reliability models with two debugging rates," *International Journal of Reliability, Quality and Safety Engineering*, vol. 6, 1, pp. 31-42, 1999.

A. L. Goel and K. Okumoto, "Time dependent error detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. R-28, 1, pp. 206-211, 1979.

Z. Jelinski and P. B. Moranda, "Statistical computer performance evaluation," in *Software reliability research*, W. Freiberger ed, Academic press, New-York, pp. 465-497, 1972.

R. E. Kass and A. E. Raftery, "Bayes factors," *Journal of American Statistical Association*, vol. 90, pp. 773-795, 1995.

P. A. Keiller, B. Littlewood, D. R. Miller and A. Sofer, "Comparison of software reliability predictions," *Proceedings 13th IEEE International Symposium on Fault Tolerant Computing*, Milano, IEEE Computer society Press, pp. 128-134, 1983.

B. Littlewood, "Predicting software reliability," *Philosophical Transactions of the Royal Statistical Society, Series A*, vol. 327, pp. 513-527, 1989.

B. Littlewood and J. Verral, "A Bayesian reliability growth model for computer software," *Journal of the Royal Statistical Society, Series C*, vol. 22, pp. 332-336, 1973.

M. R. Lyu ed., *Handbook of software reliability engineering*, IEEE Computer Society Press and Mc Graw-Hill Book Company, 1996.

G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, J. Wiley and sons, 1997.

P. B. Moranda, "Event altered rate models for general reliability analysis," *IEEE Transactions on Reliability*, vol. R-28, 5, pp. 376-381, 1979.

J. D. Musa, "Software reliability data," Technical Report, Rome Air Development center, 1979.

H. Pham, *Software reliability*, Springer, 2000.

C.P. Robert, T. Rydèn and D.M. Titterington, "Bayesian inference in hidden Markov models through the reversible jump Markov chain Monte Carlo method," *Journal of the Royal Statistical Society, Series B*, vol. 62, pp. 57-75, 2000.

J. L. Soler, "Modélisation des processus de risque, de défaillance et de correction des systèmes présentant des fautes de conception - Application à la fiabilité des logiciels," *Proceedings of 6th Nat. Conf. On Reliability and Maintainability*, Strasbourg, pp. 647-650 (in French), 1988.

S. Yamada, M. Ohba and S. Osaki, "S-shaped reliability growth modelling for software error detection," *IEEE Transactions on Reliability*, vol. R-35, 5, pp. 475-478, 1983.

M. Zhao, "Change-point problems in software and hardware reliability," *Communications in Statistics, Theory and Methods*, vol. 22, pp. 757-768, 1993.