Software Reliability Modelling and Prediction with Hidden Markov Chain

Jean-Baptiste Durand, Olivier Gaudoin

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Software Reliability Modelling and Prediction with Hidden Markov Chain

Jean-Baptiste Durand — Olivier Gaudoin

N° 4747

Février 2003

THÈME 4

ISSN 0249-6399 ISRN INRIA/RR--4747--FR+ENG

INRIA
RHÔNE-ALPES

Software Reliability Modelling and Prediction with Hidden Markov Chain

Jean-Baptiste Durand * , Olivier Gaudoin†

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet IS2

Rapport de recherche n° 4747 — Février 2003 — 25 pages

Abstract: The purpose of this report 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 in average with respect to usual models, especially when the reliability is not regularly growing.

Key-words: software reliability, reliability growth models, debugging, hidden Markov chains, EM, BIC

* Projet IS2, INRIA Rhône-Alpes, Jean-Baptiste.Durand@inrialpes.fr
† INP Grenoble, Laboratoire LMC, BP 53, 38041 Grenoble Cedex 9, France, Olivier.Gaudoin@imag.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Modélisation et prévision de la fiabilité des logiciels par chaînes de Markov cachées

Résumé : L'objet de ce rapport est l'utilisation du cadre des chaînes de Markov cachées pour la modélisation du processus des défaillances et des corrections des logiciels, ainsi que la prévision de leur fiabilité. Les paramètres du modèle sont estimés en utilisant l'algorithme EM *avant-arrière* et la sélection de modèles est réalisée à l'aide du critère BIC. Les avantages et les inconvénients de cette approche par rapport aux modèles usuels sont analysés. La comparaison est également effectuée sur la base de données réelles. Le principal apport de la modélisation par chaînes de Markov cachées est de mettre en évidence l'existence de périodes homogènes dans le processus des corrections, qui permettent d'identifier les corrections majeures ou les mises à jour du logiciel. En termes de prévision de la fiabilité, le modèle de chaîne de Markov cachée a de bonnes performances en moyenne par rapport aux modèles usuels, surtout quand la fiabilité ne croît pas de manière régulière.

Mots-clés : fiabilité des logiciels, modèles de croissance de fiabilité, corrections des logiciels, chaînes de Markov cachées, EM, BIC

1 Software reliability modelling

Studies on software reliability modelling began 30 years ago. Today, more than 50 stochastic models have been proposed for the software failure and debugging process. 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 [19] and Pham [22].

A general framework for stochastic software reliability models is that of self-exciting random point processes, proposed in Gaudoin [9] and Chen-Singpurwalla [3]. Let $T_i, i \geq 1$, be the successive software failure times, starting from $T_0 = 0$. After each failure, the software is corrected or not, and restarted. It is usual to consider that debugging times are negligible or not taken into account. Then, let $X_i = T_i - T_{i-1}, i \geq 1$, be the successive times between failures and N_t be the number of failures occurred between 0 and t. The failure process is equivalently one of the random processes $\{T_i\}_{i\geq 1}$, $\{X_i\}_{i\geq 1}$ or $\{N_t\}_{t\geq 0}$. It is completely defined by the failure intensity:

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

where $\mathcal{F}_t = \sigma\left(\{N_s\}_{0\leq s\leq t}\right)$ is the internal filtration of the failure process.

Software reliability at time t expresses the probability that no software failure will occur during a time of any length τ after t, conditionally to the past of the failure process:

$$R_t(\tau) = P\left(N_{t+\tau} - N_t = 0 | \mathcal{F}_t\right) = P\left(T_{N_t+1} - t > \tau | \mathcal{F}_t\right) = \exp\left(-\int_t^{t+\tau} \lambda_s ds\right). \tag{2}$$

Most software reliability models assume that $\{N_t\}_{t\geq 0}$ is a non homogeneous Poisson process (NHPP), for which failure intensity is a deterministic and continuous function of time: $\lambda_t = \lambda(t)$. Among them, the most usual are:

- The Duane model or Power-Law Process [6] (PLP): $\lambda(t) = \alpha\beta t^{\beta-1}$, $\alpha \in \mathbb{R}^+, \beta \in \mathbb{R}^+$.
- The Goel-Okumoto model [12] (GO): $\lambda(t) = \lambda e^{-\phi t}$, $\lambda \in \mathbb{R}^+, \phi \in \mathbb{R}$.
- The S-shaped model [25] (S): $\lambda(t) = \alpha\beta^2 t e^{-\beta t}$, $\alpha \in \mathbb{R}^+, \beta \in \mathbb{R}^+$.

The main reason for the wide use of NHPP in software reliability is the simplicity of their use. But the main drawback of these models is the assumption that failure intensity is a continuous function of time: it is more realistic to consider that debugging induces a discontinuity in failure intensity.

Moreover, an important feature of software, and a major difference with hardware, is that 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. Then, another class of models is those for which the times between failures X_i are independent are exponentially distributed. Here, the failure intensity is a step function.

More sophisticated models are built following Littlewood's [17] statement, according to whom 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. Note that there is a strong link between inputs and failures: there is a failure if, for a given input, the software output is not that provided by the specifications.

Then, the concept of total fault has been defined in [9]: the total fault at time t, F_{N_t}, is the set of all inputs which can provoke a failure at time t. The effect of debugging is a transformation of F_{N_t} into F_{N_t+1}. It is logical to assume that the debugging of a fault at a given time depends only of the state of this fault at this time, and not of its past states. So the fault process $\{F_{N_t}\}_{t\geq 0}$ can be assumed Markovian. It is proved in [9] that, if the inputs occur in time according to a homogeneous Poisson process with parameter μ, if they are independent, independent of input times, and have the same distribution Q on the input space, then the failure intensity is:

$$\lambda_t = \mu Q(F_{N_t}). \tag{3}$$

The uncertainty on inputs is expressed by μ and Q, and the uncertainty on debugging by the fault process $\{F_{N_t}\}_{t\geq 0}$.

If we set $\Lambda_i = \mu Q(F_{i-1})$, we obtain, as stated yet by Soler [24], that there exists a Markov process $\mathbf{\Lambda} = \{\Lambda_i\}_{i\geq 1}$ such that, conditionally to $\{\Lambda_i = \lambda_i\}_{i\geq 1}$, the times between failures X_i are independent and exponentially distributed with respective parameters λ_i. Λ_i can be understood as the software failure rate after $(i-1)^{\text{th}}$ debugging. A software reliability model in this framework is given by a model for process $\{\Lambda_i\}_{i\geq 1}$.

Several software reliability models belong to this class:

- When the Λ_i are deterministic, the times between failures X_i are independent and exponentially distributed. The most famous of these models are:
  - The Jelinski-Moranda model [13] (JM): $\Lambda_i = \phi(N-i+1)$, $\phi \in \mathbb{R}^+, N \in \mathbb{N}$.
  - The Moranda geometric model [20] (MG): $\Lambda_i = \lambda c^{i-1}$, $\lambda \in \mathbb{R}^+, c \in (0,1]$.
- The Littlewood-Verral model [18] (LV) can be understood as a model of this class for which the Λ_i are independent and have the gamma distribution with respective parameters $(\alpha, \beta_1 + i\beta_2)$, $(\alpha, \beta_1, \beta_2) \in \mathbb{R}^{+3}$.
- In [10], Gaudoin, Lavergne and Soler defined a class of models, called the proportional models, assuming that:

$$\forall i \geq 1,\ \Lambda_{i+1} = \Lambda_i e^{-\Theta_i},\ \text{where } \Lambda_i \text{ and } \Theta_i \text{ are independent.} \tag{4}$$

  The Θ_i represent the successive debugging effects: $\Theta_i = 0 \Longrightarrow \Lambda_{i+1} = \Lambda_i$ means that debugging has no effect, $\Theta_i > 0 \Longrightarrow \Lambda_{i+1} < \Lambda_i$ means that the correction reduces software failure rate, so debugging is of good quality, and $\Theta_i < 0 \Longrightarrow \Lambda_{i+1} > \Lambda_i$ means that debugging is of bad quality.

- The imperfect debugging model proposed by Gaudoin [11] assumes that:

$$\forall i \geq 1, \Lambda_i = (1 - \alpha_i - \beta_i)\,\Lambda_{i-1} + \mu\beta_i \tag{5}$$

  where the α_i are good debugging rates and the β_i are bad debugging rates. A simple model is obtained by letting $\alpha_i = \alpha$ and $\beta_i = \beta$ for all i.

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 doing any correction. Debugging happens when a sufficiently large amount 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 in 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. Then, 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 EM algorithm. Section 4 deals with the hidden states' 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 studied 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 $\mathbf{\Lambda}$ sum up as follows:

1. $\mathbf{\Lambda}$ is a discrete-valued Markov chain ;
2. conditionally to $\{\Lambda_i = \lambda_i\}_{i\geq 1}$, the times between failures $\{X_i\}_{i\geq 1}$ are independent ;

3. conditionally to $\{\Lambda_i = \lambda^{(j)}\}$, 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 $\Lambda_{i+1} = \lambda^{(l)}$.

In the software test period, the homogeneous zones can be interpreted as periods where no corrections have 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.

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 the class of models presented in section 1 is that there is not necessarily a correction after each failure, which leads to the existence of homogeneous periods.

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 confine to the matrix P defined by the p_{jl}.

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

Let η be the set of all parameters of a HMC model. In sections 3 and 4 as well as in the current section, the number K of possible failure rates is considered as known.

A consequence of the conditional independence hypothesis and the finite values of $\boldsymbol{\Lambda}$ is that the marginal distribution of X_i is a finite mixture, since its density f_{X_i} is given by

$$f_{X_i}(x) = \sum_j P(\Lambda_i = \lambda^{(j)}) f_{\lambda^{(j)}}(x) \tag{6}$$

where $f_{\lambda^{(j)}}(x)$ denotes the density of the exponential distribution with parameter $\lambda^{(j)}$, which is the distribution of X_i conditionally to $\{\Lambda_i = \lambda^{(j)}\}$. Hence the HMC model is a mixture model where a Markovian dependence assumption substitutes for the usual independence assumption of the $\{X_i\}_{i\geq 1}$. Though, the $\boldsymbol{X}$ process is not a Markov chain itself. In mixture models, an unknown discrete variable Z_i is associated to each observed variable X_i. Observed variables having the same value of Z_i can be grouped in a same cluster, which makes the

mixture models widely used in clustering. In the same way, the HMC models take advantage of the unknown discrete variables $\{\Lambda_i\}_{i\geq 1}$, also called (hidden) states, for the classification of the data $\{X_i\}_{i\geq 1}$, where in addition, the dependencies between variables are taken into account (for instance *via* the Markov assumption). The obtained clusters form, in the $\boldsymbol{X}$ process, the homogeneous zones mentioned above.

It can be noticed that Kimura and Yamada [16] used yet the HMC framework in software reliability, but with a completely different point of view.

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 η by likelihood maximization. Thus, we resort to the EM algorithm [4], dedicated to the likelihood maximization in the context of missing values. This iterative algorithm starts from an initial value $\eta^{(0)}$ of the parameters and creates a sequence $\{\eta^{(m)}\}_{m\geq 0}$ whose likelihood grows. At each iteration m, it proceeds as follows:

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

$$Q(\eta,\eta^{(m)})=E_{\eta^{(m)}}[\log P_\eta(\boldsymbol{\Lambda}_1^n,\boldsymbol{X}_1^n=\boldsymbol{x}_1^n)|\boldsymbol{X}_1^n=\boldsymbol{x}_1^n]; \tag{7}$$

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

$$\eta^{(m+1)}=\arg\max_\eta Q(\eta,\eta^{(m)}). \tag{8}$$

The EM algorithm has the following properties: the sequence $\{\eta^{(m)}\}_{m\geq 0}$ converges to the consistent solution of the likelihood equations when $\eta^{(0)}$ is close to the optimal solution. However, when the mixture components $\{\lambda^{(1)},\ldots,\lambda^{(K)}\}$ are poorly separated, the estimators obtained strongly depend on the initial value $\eta^{(0)}$ and the convergence rate can be crippling. To cope with the former 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 estimators. The one which has maximal likelihood is used as starting position for the EM algorithm. We stop the algorithm

when 1000 iterations have been done, or when the relative increase of the log-likelihood is below a threshold ε, for instance $\varepsilon = 10^{-6}$.

>From Baum *et al.* [2], the *M step* is given by the following reestimation formulae:

$$\hat{\pi}_j = \gamma_1(j) \tag{9}$$

$$\hat{p}_{jl} = \frac{\sum_i \xi_i(j,l)}{\sum_i \gamma_i(j)} \tag{10}$$

$$\hat{\lambda}^{(j)} = \left[\frac{\sum_i \gamma_i(j) x_i}{\sum_i \gamma_i(j)}\right]^{-1} \tag{11}$$

where

$$\gamma_i(j) = E_{\eta^{(m)}}[\mathbb{1}_{\{\Lambda_i=\lambda^{(j)}\}}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n] = P_{\eta^{(m)}}(\Lambda_i = \lambda^{(j)}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n) \tag{12}$$

$$\begin{aligned}\xi_i(j,l) &= E_{\eta^{(m)}}[\mathbb{1}_{\{\Lambda_i=\lambda^{(l)},\Lambda_{i-1}=\lambda^{(j)}\}}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n] \\ &= P_{\eta^{(m)}}(\Lambda_i = \lambda^{(l)}, \Lambda_{i-1} = \lambda^{(j)}|\boldsymbol{X}_1^n = \boldsymbol{x}_1^n).\end{aligned} \tag{13}$$

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, due to Baum *et al.* [2], to be called the *forward-backward algorithm.*

In what follows, the parameters $\eta^{(m)}$ are fixed and we denote $P = P_{\eta^{(m)}}$ 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

$$\alpha_{i+1}(l) = \sum_j p_{jl}\alpha_i(j) f_{\lambda^{(l)}}(x_{i+1}). \tag{14}$$

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

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

$$\beta_i(j) = \sum_l p_{jl}\beta_{i+1}(l) f_{\lambda^{(l)}}(x_{i+1}). \tag{15}$$

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

$$\gamma_i(j) = \frac{\alpha_i(j)\beta_i(j)}{\sum_l \alpha_n(l)} \tag{16}$$

$$\xi_i(j,l) = \frac{\beta_{i+1}(l)p_{jl}f_{\lambda^{(l)}}(x_{i+1})\alpha_i(j)}{\sum_{j'} \alpha_n(j')}. \tag{17}$$

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 [5], 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 $P_\eta(\boldsymbol{\Lambda}_1^n = \boldsymbol{\lambda}_1^n, \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$, are as follows:

$$\tilde{\pi}_j = \mathbb{1}_{\{\Lambda_1=\lambda^{(j)}\}} \tag{18}$$

$$\tilde{p}_{jl} = \frac{\sum_i \mathbb{1}_{\{\Lambda_i=\lambda^{(l)},\Lambda_{i-1}=\lambda^{(j)}\}}}{\sum_i \mathbb{1}_{\{\Lambda_{i-1}=\lambda^{(j)}\}}} \tag{19}$$

$$\tilde{\lambda}^{(j)} = \left[\frac{\sum_i \mathbb{1}_{\{\Lambda_i=\lambda^{(j)}\}} x_i}{\sum_i \mathbb{1}_{\{\Lambda_i=\lambda^{(j)}\}}}\right]^{-1}. \tag{20}$$

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 reestimation formulae (9) to (11) of the EM algorithm, the unknown random quantities involved through the indicator functions are replaced by their expectations, conditional to $\{\boldsymbol{X}_1^n = \boldsymbol{x}_1^n\}$, with respect to the distribution $P_{\eta^{(m)}}$.

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 $\mathbb{1}_{\{\lambda=\lambda^{(l)}\}}$ when only one sequence is used to estimate the parameters, which is the case in this paper. On the other hand, the distribution P_η remains invariant by any permutation of the parameters η - corresponding to a relabelling of the hidden states - as usual in mixture models. As a consequence, we can make the assumption that $\pi_j = \mathbb{1}_{\{\lambda=\lambda^{(1)}\}}(\lambda^{(j)})$. It means that the failure rate at $i = 1$ is $\Lambda_1 = \lambda^{(1)}$ with a 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 to interpret 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 η, 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, conditionally to $\boldsymbol{x}_1^n$. The resulting algorithm is called a MAP algorithm (maximum a posteriori). It consists in computing $\arg\max_{\boldsymbol{\lambda}_1^n} P_\eta(\boldsymbol{\Lambda}_1^n = \boldsymbol{\lambda}_1^n | \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$, or equivalently $\arg\max_{\boldsymbol{\lambda}_1^n} P_\eta(\boldsymbol{\Lambda}_1^n = \boldsymbol{\lambda}_1^n, \boldsymbol{X}_1^n = \boldsymbol{x}_1^n)$. This can be done by the so-called Viterbi algorithm [7], based on dynamic programming methods. 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 $\delta_i(j) = \max_{\boldsymbol{\lambda}_1^{i-1}} P_\eta(\boldsymbol{\Lambda}_1^{i-1} = \boldsymbol{\lambda}_1^{i-1}, \Lambda_i = \lambda^{(j)}, \boldsymbol{X}_1^i = \boldsymbol{x}_1^i)$, initialized at $i = 1$ by $\delta_1(j) = \pi_j f_{\lambda^{(j)}}(x_1)$ and computed inductively by

$$\delta_{i+1}(l) = \max_j p_{jl} \delta_i(j) f_{\lambda^{(l)}}(x_{i+1}). \tag{21}$$

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 $i-1$. This backtracking procedure consists in tracing backward along this pointers from the optimal final state to the initial state.

Another strategy for the hidden states' restoration consists in computing

$$\bar{\lambda}_i = \arg\max_j P(\Lambda_i = \lambda^{(j)} | \boldsymbol{X}_1^n = \boldsymbol{x}_1^n). \tag{22}$$

It leads to the maximization of the expected number of correct failure rates. Though, this local method 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 $1/\lambda^{(j)}$.

Figure 1a) shows the times between failures for a real software failure data set (C3 in [9]) superimposed on the optimal state sequence restored by the Viterbi algorithm, using a

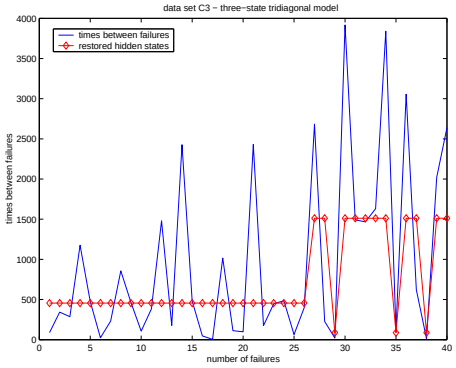

Fig. 1a) *The globally most likely sequence.*

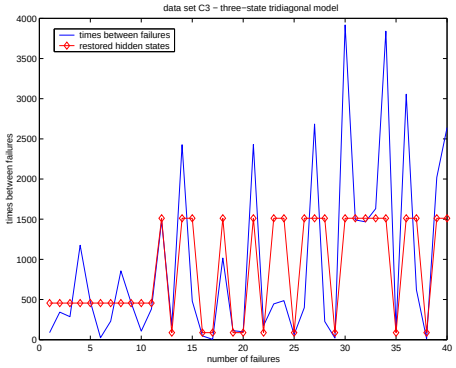

Fig. 1b) *The individually most likely states.*

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

three-state model with parameters $\hat{\eta}$ estimated by maximum likelihood. Figure 1b) 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 $\hat{\eta}$. 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.

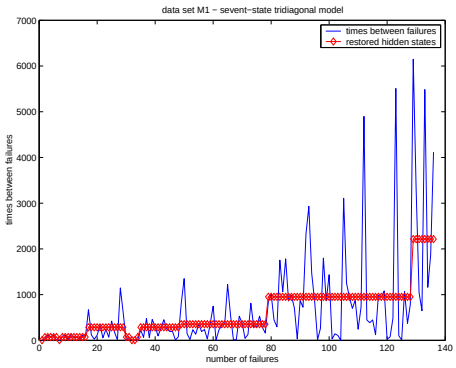

Fig. 2a) *The globally most likely sequence.*

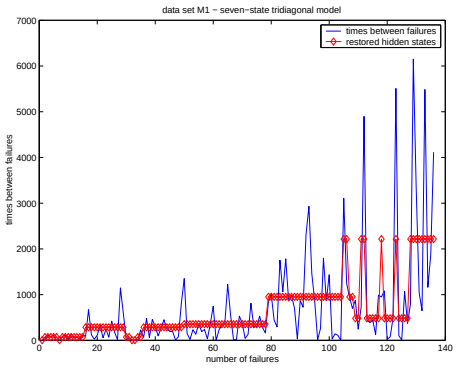

Fig. 2b) *The individually most likely states.*

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

Figure 2a) shows the times between failures for another real data set (M1 in [21]) 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 2b) 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 on 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.

5 Choice of the hidden states' 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 η depends on K. With each value of K, we can associate the set of hidden Markov chains having K hidden states. These sets can be called the competing hidden Markov chain models. Generally, the problem of choosing the number of hidden states requires that one can express (more or less formally) which use of the model is planned. For example, one may want to choose a model having a moderate complexity which reasonably fits the data. We will focus on this objective in what follows. Then, the definition of a criterion to assess the relevance of a given model, regarding our aim, can be envisaged.

We use a criterion based on a penalization of the log-likelihood, called the Bayesian Information Criterion (BIC) [14], also known as the *Schwarz criterion*. This criterion is defined by

$$BIC(K) = \log(P_{\hat{\eta}_K}(\boldsymbol{x}_1^n)) - \frac{\nu_K}{2} \log(n) \tag{23}$$

where $\log(P_{\hat{\eta}_K}(\boldsymbol{x}_1^n))$ 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 η_K. We recall that n is the length of the observed sequence, that is the number of failures. The number of independent parameters 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 has its origins in a Bayesian approach, where the probability of the sequence $\boldsymbol{x}_1^n$ conditionally to a model is assessed by integrating over the parameters. The integral is approximated using a Laplace expansion, valid when the maximum likelihood

estimator is asymptotically normally distributed, which is not always the case here if we assume that there exists a true value K_0 - the asymptotic normality not being satisfied for $K > K_0$. In any case, 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 [8], criteria based on the marginal log-likelihood with the same penalization as that of BIC are proved to be consistent.

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 on the transition probability matrix, which means that any improvement or deterioration of the software failure rate is possible.

It can be envisaged 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. Then, any transition from a hidden state to a previously visited hidden state must be forbidden. This assumption implies an upper diagonal transition probability matrix, of the following form (up to a relabelling of the states):

$$P = \begin{pmatrix} p_{1,1} & p_{1,2} & 0 & \dots & 0 \\ 0 & p_{2,2} & p_{2,3} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & p_{K-1,K-1} & p_{K-1,K} \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix} \tag{24}$$

In this matrix, each line sums to 1, so $\forall l < K$, $p_{l,l+1} = 1 - p_{l,l}$. Under this assumption, the fact that $\Lambda_1 = \lambda^{(1)}$ with probability one gives no choice for the meaning of the other $\lambda^{(l)}$, since $\lambda^{(2)}$ will necessarily correspond to the second visited state and so on. Thus, the failure rates are ordered in accordance to the transition probability matrix. It should be noted that when the states must be relabelled to ensure the condition $\pi_j = \mathbb{1}_{\{\lambda=\lambda^{(1)}\}}(\lambda^{(j)})$, the transition probability matrix is generally no more upper diagonal. This is not a problem since the unknown failure rates remain ordered in the sense above. This assumption is satisfied if some permutation of the rows and the columns of the transition probability matrix results in an upper diagonal matrix.

In practice, it is important to take into account the possibility of imperfect debugging. It 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 of the following form (up to a permutation):

$$P = \begin{pmatrix} p_{1,1} & p_{1,2} & 0 & \dots & 0 \\ p_{2,1} & p_{2,2} & p_{2,3} & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & p_{K-1,K} \\ 0 & \dots & 0 & p_{K,K-1} & p_{K,K} \end{pmatrix} \tag{25}$$

In this matrix, for each l such that $1 < l < K$, $p_{l-1,l} + p_{l,l} + p_{l,l+1} = 1$, $p_{1,1} + p_{1,2} = 1$ and $p_{K,K-1} + p_{K,K} = 1$. 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 above all, but they also translate into the restoration of the hidden sequence by the Viterbi algorithm. Figure 3a) shows the times between failures for a real data set (C1 in [9]) 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. On Figure 3b), a model with an upper diagonal transition matrix has been used, thus leading to new estimates for the parameters (still by maximum likelihood method) and a new restored sequence of failure rates. On Figure 3c), a model with a tridiagonal transition matrix has been used. Figure 3b) illustrates the fact that a model with an upper diagonal transition probability matrix forbids 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 3a) and c) that adding milder constraints than an upper diagonal transition probability matrix results in the suppression of some spurious transitions.

>From an estimation point of view, a change in the type of transition matrix is likely to affect the estimates of every parameter included in $\hat{\eta}$ - even the failure rates $\hat{\lambda}^{(j)}$. In fact, we do not know of any direct method to deduce the estimates of the parameters for a particular model from the estimates for the most general model. Thus, the EM algorithm has to be run for every model separately. We can see from the reestimation formulae of the EM algorithm (see section 3) that if p_{jl} is equal to zero in the initial parameter $\eta^{(0)}$ then at each EM iteration, the estimate $\hat{p}_{jl}$ remains equal to zero and the constraints on the transition probability matrix are automatically satisfied.

In conclusion, we are now facing a new model selection problem where the competing models differ from their type of 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. It is worth noting that in the context of a fixed value for K,

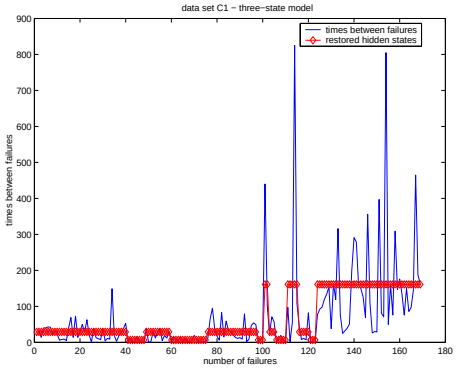

Fig. 3a) *Model with no constraint on the transition probability matrix.*

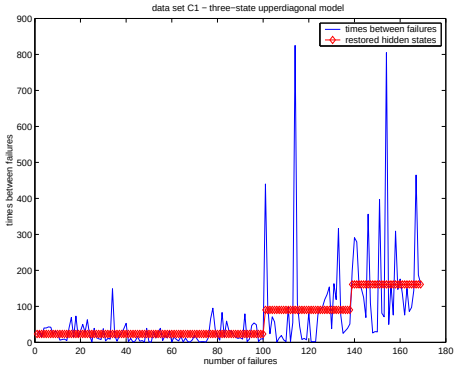

Fig. 3b) *Model with an upper diagonal transition probability matrix.*

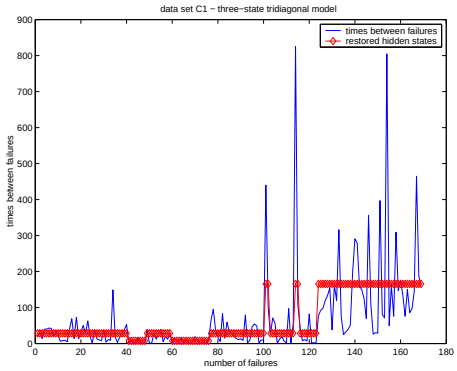

Fig. 3c) *Model with a tridiagonal transition probability matrix.*

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

assumed inferior to a conceptual true value K_0, the maximum likelihood estimator has an asymptotically normal distribution, which makes the use of BIC more justified than for the choice of K. In any case though, K remains to be chosen.

In practice, 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 [15, 1].

For any model with parameter θ, let $P_{\hat{\theta}_i}(X_i \leq x | \boldsymbol{X}_1^{i-1} = \boldsymbol{x}_1^{i-1})$ be the predicted CDF of the next time to failure X_i given the first $i-1$ interfailure times, where $\hat{\theta}_i$ is an estimator of θ based on $\boldsymbol{x}_1^{i-1}$. The idea of the method is to compute the $u_i = P_{\hat{\theta}_i}(X_i \leq x_i | \boldsymbol{X}_1^{i-1} = \boldsymbol{x}_1^{i-1})$. 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:

$$P(X_i \leq x | \boldsymbol{X}_1^{i-1} = \boldsymbol{x}_1^{i-1}) = \frac{\sum_l F_{\lambda^{(l)}}(x) \sum_k \alpha_{i-1}(k) p_{kl}}{P_{\hat{\eta}_i}(\boldsymbol{x}_1^{i-1})} \tag{26}$$

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

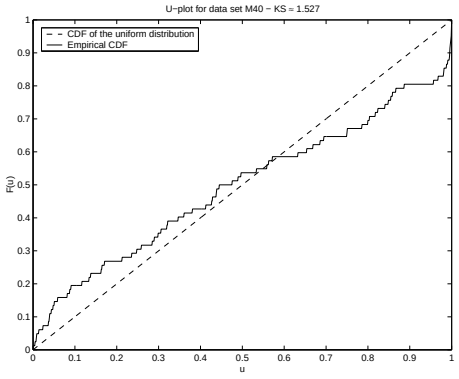

Figure 4: U-plot for the HMC model

Figure 4 shows the U-plot for data set M40 in [21], using a three-state model with upper diagonal transition probability matrix. On 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 are times between failures of nine American control-command software (denoted M1, M2, M3, M4, M6, M14C, M17, M27 and M40) in the test period and operational life [21]. The second group are times between failures of four complex French software (denoted C1, C2, C3, C4) in the test period [9]. Time unit is running clock time in both cases.

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 $K_{max} = 7$.

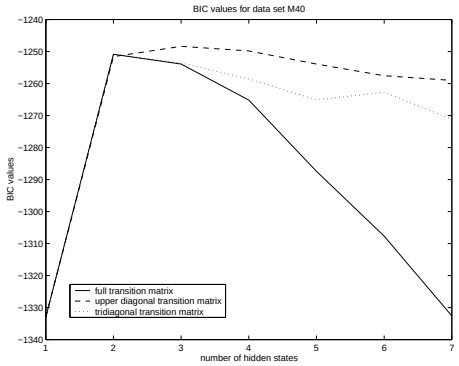

Figure 5: Model selection with BIC for M40

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 estimators are:

$$\left[\begin{array}{ccc} \hat{\lambda}^{(1)} & \hat{\lambda}^{(2)} & \hat{\lambda}^{(3)} \end{array}\right] = 10^{-4} * \left[\begin{array}{ccc} 0.5035 & 0.0908 & 0.0175 \end{array}\right] \quad (27)$$

$$\hat{P} = \left[\begin{array}{ccc} 0.9809 & 0.0191 & 0 \\ 0 & 0.9502 & 0.0498 \\ 0 & 0 & 1.0000 \end{array}\right] \quad (28)$$

Then, it is easy to restore the hidden states, as shown in figure 6. 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. Table 1 gives the KS distances in the U-plots of several

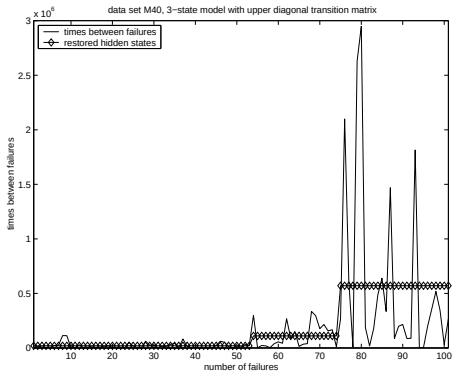

Figure 6: Restoration of the hidden states for M40.

models. The MG model has the best predictive validity, followed by PLP and the hidden Markov chain model. The HMC model performs much better than GO, S and JM models.

Table 1: Model comparison for M40 - KS distances in U-plots

HMC	PLP	GO	S	JM	MG
1.375	1.310	2.417	3.342	4.783	1.223

A similar study has been done on the thirteen data sets. Table 2 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 ranks of 6 models according to the U-plot method (rank 1 is for the model with the smallest KS distance). The table also gives the mean rank of each model for all data sets. The complete table of KS distances is given in appendix A. The restoration of the hidden states in HMC models for all data sets is given in appendix B.

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 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 and spurious transitions appear, so a model with less hidden states is preferable. This means that for these software, very few major corrections seem to have occurred during the debugging process.

Table 2: Summary of model application on 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	4	4	4	2	1	1	2	2	3	1	3	5	5	2.85
PLP	2	3	3	1	2	5	5	4	2	4	1	4	3	2.77
GO	3	2	2	4	4	6	3	3	4	2	4	1	2	3.08
S	5	5	5	5	5	3	6	5	5	5	6	6	4	5.00
JM	6	6	6	6	6	2	1	6	6	6	5	2	6	4.92
MG	1	1	1	3	3	4	4	1	1	3	2	3	1	2.15

- 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 neither the KS distance nor 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 10 others, the HMC performs well in 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 2 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 few 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), 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 of the hidden states changes, which makes prediction of future time to failure much more difficult than it seems. Then, 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. Then, it leads to mixtures of usual models with Markovian transitions. For example, a "hidden Markov Moranda geometric model" corresponds to the assumption that, conditionally to $\{\Lambda_i = \lambda^{(j)}\}$, 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

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

[2] Baum L.E., Petrie T., Soules G. and Weiss N., A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, *Annals of Mathematical Statistics*, **41**, 164-171, 1970.

[3] Chen Y. and Singpurwalla N., Unification of software reliability models by self-exciting point processes, *Advances in Applied Probability*, **29**, 337-352, 1997.

[4] Dempster A., Laird N. and Rubin D., Maximum likelihood from incomplete data via the EM algorithm (with discussion), *Journal of Royal Statistical Society Series B*, **39**, 1-38, 1977.

[5] Devijver P.A., Baum's forward-backward algorithm revisited, *Pattern Recognition Letters*, **3**, 369-373, 1985.

[6] Duane J.T., Learning curve approach to reliability monitoring, *IEEE Transactions on Aerospace*, **AS-2**, 2, 563-566, 1964.

[7] Forney G.D. Jr., The Viterbi Algorithm, *Proceedings of the IEEE*, **61**, 268-278, March 1973.

[8] Gassiat E., Likelihood ratio inequalities with application to various mixtures, *Annales de l'Institut Henri Poincaré*, 2002 (to appear).

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

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

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

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

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

[14] Kass R.E. and Raftery A.E., Bayes factors, *Journal of American Statistical Association*, **90**, 773-795, 1995.

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

[16] Kimura M. and Yamada S., Statistical estimation of imperfect debugging rate based on hidden Markov software reliability modelling, *Proceedings 5th ISSAT Int. Conf. on Reliability and Quality in Design*, Las Vegas, 331-335, 1999.

[17] LITTLEWOOD B., Predicting software reliability, *Philosophical Transactions of the Royal Statistical Society, London, Series A*, **327**, 513-527, 1989.

[18] LITTLEWOOD B. AND VERRAL J., A Bayesian reliability growth model for computer software, *Journal of the Royal Statistical Society - Series C*, **22**, 332-336, 1973.

[19] LYU M.R. ED., *Handbook of software reliability engineering*, IEEE Computer Society Press and Mc Graw-Hill Book Company, 1996.

[20] MORANDA P.B., Event altered rate models for general reliability analysis, *IEEE Transactions on Reliability*, **R-28**, 5, 376-381, 1979.

[21] MUSA J.D., *Software reliability data*, Technical Report, Rome Air Development center, 1979.

[22] PHAM H., *Software reliability*, Springer, 2000.

[23] QIAN W. AND TITTERINGTON D.M., Estimation of parameters in hidden Markov models, *Philosophical Transactions of the Royal Statistical Society, London, Series A*, **337**, 407-428, 1991.

[24] SOLER J.L., 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, 647-650, 1988 (in French).

[25] YAMADA S., OHBA M. AND OSAKI S., S-shaped reliability growth modelling for software error detection, *IEEE Transactions on Reliability*, **R-35**, 5, 475-478, 1983.

Appendices

A Table of KS distances for 6 models and 13 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
K	3	2	2	2	2	2	2	2	3	2	3	1	4
HMC	1.819	1.737	1.683	0.749	0.765	0.665	0.745	0.899	1.375	0.914	2.067	1.462	3.389
PLP	1.402	1.307	1.509	0.661	1.595	1.501	0.927	1.322	1.320	1.699	1.669	1.215	2.976
GO	1.589	1.161	1.147	0.850	1.668	1.702	0.746	1.001	2.417	1.114	4.284	0.637	2.162
S	4.433	2.487	2.604	1.962	2.563	0.989	0.993	2.432	3.342	2.930	7.635	1.578	3.283
JM	5.858	3.601	NC	NC	3.106	0.894	0.571	2.598	4.783	6.290	7.369	0.710	NC
MG	1.209	0.813	0.774	0.758	1.662	1.468	0.747	0.880	1.223	1.289	1.953	0.747	1.920

NC means that the algorithm which computes the KS distance did not converge. It happened three times, only for the JM model.

B Restoration of the hidden states for all data sets

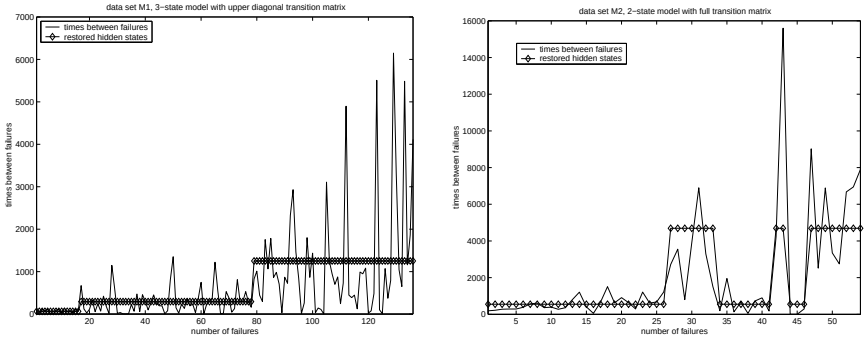

Figure 7: Restoration of the hidden states for M1 and M2

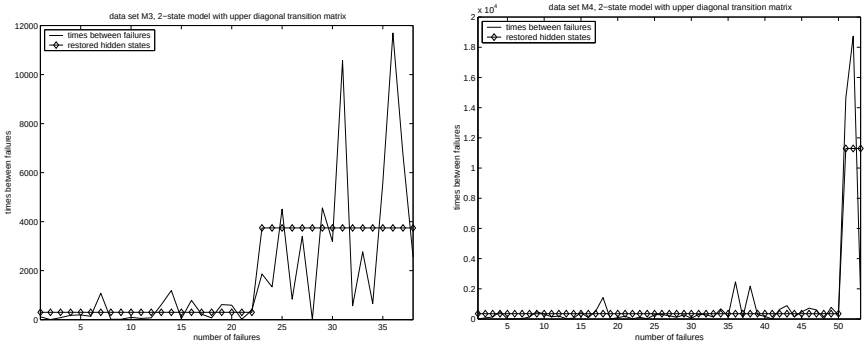

Figure 8: Restoration of the hidden states for M3 and M4

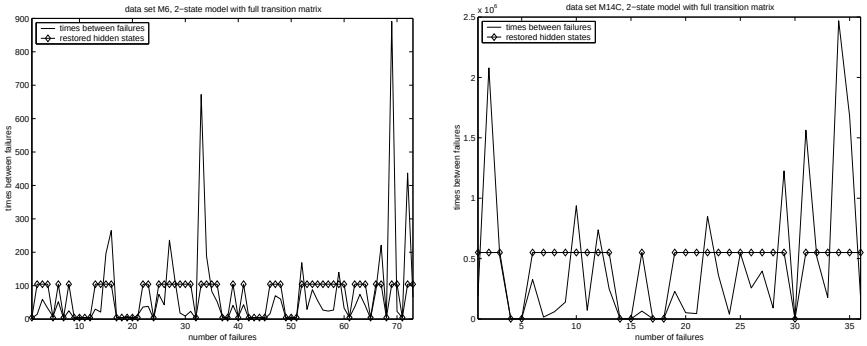

Figure 9: Restoration of the hidden states for M6 and M14

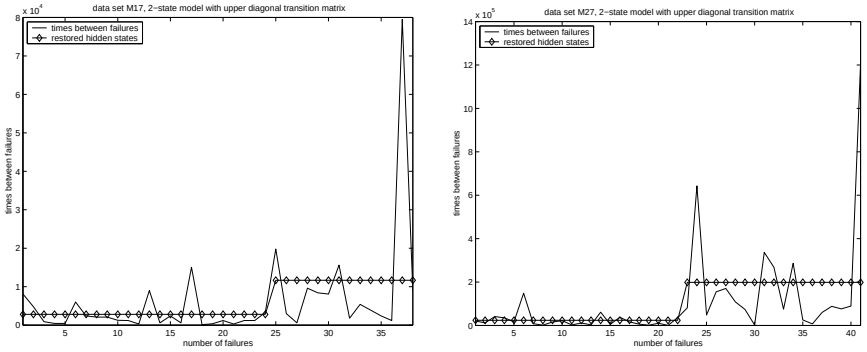

Figure 10: Restoration of the hidden states for M17 and M27

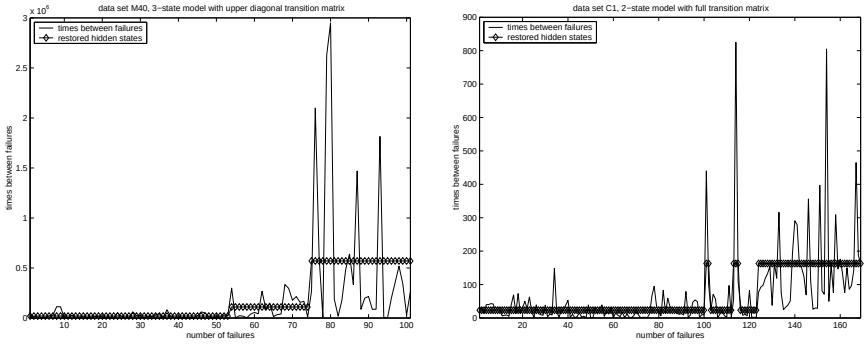

Figure 11: Restoration of the hidden states for M40 and C1

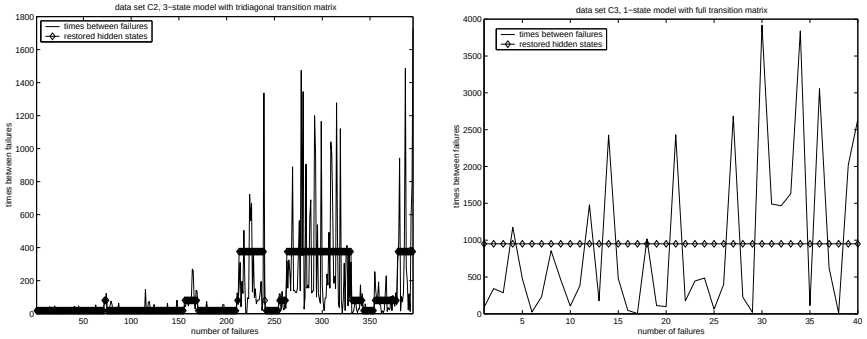

Figure 12: Restoration of the hidden states for C2 and C3

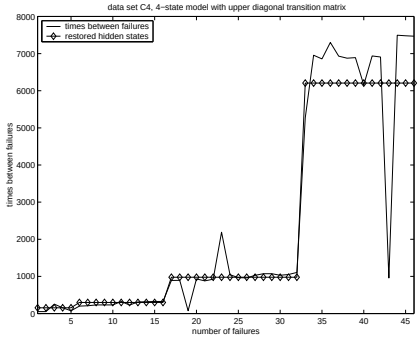

Figure 13: Restoration of the hidden states for C4

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399