FIVE PERFORMABILITY ALGORITHMS: A COMPARISON

LUCIA CLOTH AND BOUDEWIJN R. HAVERKORT*

Abstract. Since the introduction by John F. Meyer in 1980 [21], various algorithms have been proposed to evaluate the performability distribution. In this paper we describe and compare five algorithms that have been proposed recently to evaluate this distribution: Picard's method, a uniformisation-based method, a path-exploration method, a discretisation approach and a fully Markovian approximation.

As a result of our study, we recommend Picard's method not to be used (due to numerical stability problems). Furthermore, the path exploration method turns out to be heavily dependent on the branching structure of the Markov-reward model under study. For small models, the uniformisation method is preferable; however, its complexity is such that it is impractical for larger models. The discretisation method performs well, also for larger models; however, it does not easily apply in all cases. The recently proposed Markovian approximation works best, even for large models; however, error bounds cannot be given for it.

Key words. Performability evaluation, Markov-reward models, computational techniques

AMS subject classifications. 60J22, 60J27, 65C20, 90B25

1. Introduction. Over the last 25 years, many algorithms for the computation of the performability distribution, that is, the distribution of accumulated reward up to some time t in a Markov reward model (MRM), have been proposed; for overviews we refer to [16, 17, 26, 35]. Early work was restricted to *acyclic* MRMs [2, 9–11]. Some algorithms for possibly cyclic MRMs are based on Laplace transforms and are therefore only suitable for MRMs with relatively small state spaces [19, 32]. In this context, however, the application of newer algorithms for Laplace transform inversion (cf. Dingle et al. [7]) has not been investigated. Other authors have considered only availability models with two different reward classes [5, 29, 30].

The contribution of this paper is that we compare five algorithms that have been proposed recently to evaluate the performability distribution in general Markov reward models. As far as we can see, this is the first time that these five algorithms are described and compared, using the same notation and using the same cases. Thus, we are able to make comparative statements about complexity (space, time) and accuracy. The five considered algorithms are: Picard's method [24], Sericola's uniformisation-based method [31], the path exploration method [4, 26], a discretisation method [34], and a fully Markovian approximation. This fifth algorithm has been developed by us (and has been described concisely in [14, 15]).

As a result of the comparison, we conclude that for small models (less than a few dozens of states) the uniformisation-based method of Sericola is the best choice. For larger models, that may even include inhomogeneities, the Markovian approximation is preferred; for some models also the discretisation algorithm appears to be a good choice. Disadvantage of both these latter methods, though, is the lack of a clear accuracy statement. Picard's method should not be used as it neither provides accurate results nor is reasonably fast. Furthermore, the performance of the path exploration method heavily depends on the model under study, and therefore cannot always be recommended.

This paper is further organised as follows. In Section 2 we recapitulate the definitions of Markov reward models and the accumulated reward distribution. Section 3

*Design and Analysis of Communication Systems, P.O. Box 217, University of Twente, 7500 AE Enschede, The Netherlands, ([lucia,brh]@cs.utwente.nl).

describes the five algorithms for the computation of the performability distribution. Section 4 then compares the algorithms with respect to accuracy and scalability by means of a small example MRM, as well as a larger MRM representing a multiprocessor system (taken from the literature). We conclude the paper in Section 5.

2. Markov Reward Models. An MRM $\mathcal{M}$ consists of a continuous-time Markov chain (CTMC) $(X_t, t \geq 0)$ and a reward structure ρ. The CTMC is defined by its state space S and the generator matrix $\mathbf{Q} = (Q_{ss'})_{s,s' \in S}$. For a state $s \in S$, the value of ρ_s indicates the rate at which reward is accumulated in state s. We only consider the case where all reward rates are nonnegative. The initial distribution of the CTMC is given by the discrete probability distribution α over all states in S. While X_t is the random variable that describes the state of the CTMC at time t, Y_t is the accumulated reward up to time t. It depends on the reward rates of the states the CTMC has visited in its evolution until time t:

$$Y_t = \int_0^t \rho_{X_u} du. \tag{2.1}$$

Y_t is a random variable, and, hence, we are interested in the distribution $F_Y(t, y) = \Pr\{Y_t \leq y\}$. Meyer called this the performability distribution [21,22]. The algorithms presented in Section 3 compute the joint distribution $\Upsilon_{ss'}(t, y)$ of the state of the CTMC X_t and the accumulated reward Y_t, given the starting state X_0, that is,

$$\Upsilon_{ss'}(t, y) = \Pr\{X_t = s', Y_t \leq y \mid X_0 = s\}$$

The distribution of the accumulated reward at time t is then given as

$$\Pr\{Y_t \leq y\} = \sum_{s \in S} \alpha_s \cdot \sum_{s' \in S} \Upsilon_{ss'}(t, y). \tag{2.2}$$

The joint distributions for state pairs $s, s' \in S$, $\Upsilon_{ss'}(t, y)$, are characterised by a set of partial differential equations [24, 31]

$$\frac{\partial \Upsilon_{ss'}(t, y)}{\partial t} + \rho_s \cdot \frac{\partial \Upsilon_{ss'}(t, y)}{\partial y} = \sum_{z \in S} Q_{sz} \cdot \Upsilon_{zs'}(t, y), \tag{2.3}$$

with initial values

$$\Upsilon_{ss'}(0, y) = \begin{cases} 1, & s = s' \text{ and } y \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

Applying the method of characteristics [24] leads to the set of integral equations

$$\Upsilon_{ss'}(t, y) = e^{Q_{ss}t} \Upsilon_{ss'}(0, y - \rho_s t) + \int_0^t \sum_{z \neq s} e^{Q_{ss}x} Q_{sz} \Upsilon_{zs'}(t - x, y - \rho_s x) dx \tag{2.5}$$

with the same initial values.

3. Five algorithms. In this section we discuss five algorithms for the computation of $\Upsilon_{ss'}(t, y)$. We start with a straightforward solution of the integral equation (2.5) using successive approximations (Picard's method), as proposed by Pattipati et al. [24]. The second algorithm is based on uniformisation and was developed by Sericola [31]. The third algorithm explicitely explores the possible realisations (paths) of the MRM. It was first presented by Qureshi and Sanders [25, 26] and later also used in the CSRL model checking context [4, 18]. We then describe the discretisation algorithm by Tijms and Veldman [34]. The last algorithm presented is the Markovian approximation first presented in [14, 15].

3.1. Picard's method. The set of integral equations (2.5) has to be evaluated by a fixed point computation because terms involving $\Upsilon_{ss'}(t,y)$ appear on both sides. One numerical algorithm for fixed point computations is known as "Picard's method:" it generates a sequence of approximations $\Upsilon^{(n)}_{ss'}(t,y)$ that converges to the correct solution, that is,

$$\lim_{n\to\infty} \Upsilon^{(n)}_{ss'}(t,y) = \Upsilon_{ss'}(t,y).$$

The first approximation is given by

$$\Upsilon^{(0)}_{ss'}(t,y) = e^{Q_{ss}t}\Upsilon_{ss'}(0, y-\rho_s t),$$

where $\Upsilon_{ss'}(0, y-\rho_s t)$ is known from (2.4). The subsequent approximations are computed using the integral equation:

$$\Upsilon^{(n+1)}_{ss'}(t,y) = e^{Q_{ss}\tau}\Upsilon^{(n)}_{ss'}(0, y-\rho_s t) + \int_0^t \sum_{z\neq s} e^{Q_{ss}x} Q_{sz} \Upsilon^{(n)}_{zs'}(t-x, y-\rho_s x)dx.$$

The iteration is terminated if the absolute value of the difference between two subsequent iterations drops below a given accuracy threshold.

Each iteration step involves the evaluation of $|S|-1$ integrals over the previous approximation of the joint distribution. The integration can only be performed numerically. Different integration schemes are possible; for the sake of simplicity we restrict the description to the trapezoidal rule. The integration interval $[0,t]$ is divided into subintervals of size Δt. The integrals are then approximated as follows:

$$\int_0^t e^{Q_{ss}x} Q_{sz} \Upsilon^{(n)}_{zs'}(t-x, y-\rho_s x)dx \approx$$

$$Q_{sz}\cdot\left(\frac{1}{2}\Upsilon^{(n)}_{zs'}(t,y) + \sum_{i=1}^{\frac{t}{\Delta t}-1} e^{Q_{ss}i\Delta t}\Upsilon^{(n)}_{zs'}(t-i\Delta t, y-\rho_s i\Delta t) + \frac{1}{2}e^{Q_{ss}t}\Upsilon^{(n)}_{zs'}(0, y-\rho_s t)\right).$$

The integration scheme shows that it does not just suffice to compute $\Upsilon_{ss'}(t,y)$ in each iteration step. Instead we need sample points $\Upsilon_{ss'}(i\Delta t, j\Delta t)$, for all $i = 0,\cdots,\frac{t}{\Delta t}$, and $j = 0,\cdots,\frac{y}{\Delta t}$. Actually, we might also need sample points for negative values of j, but then the distribution is zero anyway and it is not necessary to compute/store these values.

The multitude of numerical integrations plus the approximate nature of the outer iteration make it impossible to indicate an estimate of the resulting numerical error for this method.

3.2. Analytical solution using uniformisation. Sericola [31] derives a uniformisation-based [12,13] solution for the system of partial differential equations that describes the complementary joint distribution of state and accumulated reward $\overline{\Upsilon}_{ss'}(t,y) = \Pr\{X_t = s', Y_t > y \mid X_0 = s\}$ for an MRM. The joint distribution $\overline{\Upsilon}_{ss'}(t,y)$ is conditioned on the number of steps n in the uniformised MRM and on the number k of transitions that happen before a certain threshold y_h (see below) and the reward bound y, as follows:

$$\overline{\Upsilon}_{ss'}(t,y) = \sum_{n=0}^{\infty} PP(\lambda t, n) \sum_{k=0}^{n} \binom{n}{k} y_h^k (1-y_h)^{n-k} C^{(h)}_{ss'}(n,k), \tag{3.1}$$

where $y_h = \frac{y - r_{h-1}t}{r_h t - r_{h-1}t}$, for $y \in [r_{h-1}t, r_h t)$, and $\binom{n}{k} y_h^k (1-y_h)^{n-k}$ is the probability that exactly k of the n transitions have happened by time $\frac{y - r_{h-1}t}{r_h}$.

The value of $C_{ss'}^{(h)}(n,k)$ is then the complementary distribution $\overline{\Upsilon}_{ss'}(t,y)$ conditioned on n and k. The $C_{ss'}^{(h)}(n,k)$-values are computed recursively. For details of this recursion we refer to [3, 31].

Using uniformisation, it is not possible to evaluate the complete infinite sum (3.1). It has to be truncated at some $N \in \mathbb{N}$. The error induced by this truncation is bounded by $\varepsilon = 1 - \sum_{n=0}^{N} PP(\lambda t, n)$, exactly as it is the case with traditional uniformisation for the computation of $\Pi_{ss'}(t) = \Pr\{X_t = s' \mid X_0 = s\}$. From the complementary probability $\overline{\Upsilon}_{ss'}(t,y)$ we compute $\Upsilon_{ss'}(t,y)$ by

$$\Upsilon_{ss'}(t,y) = \Pi_{ss'}(t) - \overline{\Upsilon}_{ss'}(t,y).$$

3.3. Path exploration. We now present a uniformisation-based method, where the $\Upsilon_{ss'}(t,y)$ are not conditioned on the number of steps taken until time t, but on the precise path taken up to this time. Let $\sigma = (s_0, \cdots, s_n) \in S^n$ be a so-called uniformised path of length $|\sigma| = n$. Its probability of occurrence is $P(\sigma) = U_{s_0 s_1} \cdot \cdots \cdot U_{s_{n-1} s_n}$, where $\mathbf{U} = (U_{ss'})_{s,s' \in S}$ is the generator matrix uniformised with uniformisation parameter λ, that is, $\mathbf{U} = \mathbf{I} + \mathbf{Q}/\lambda$. The set of all uniformised paths is denoted *uPath*, $\mathit{first}(\sigma)$ and $\mathit{last}(\sigma)$ denote the first and last state of a uniformised path σ. Then $\Upsilon_{ss'}(t,y)$ conditioned on uniformised paths is given by

$$\Upsilon_{ss'}(t,y) = \sum_{n=0}^{\infty} PP(\lambda t, n) \cdot \sum_{\substack{\sigma \in \mathit{uPath} \\ |\sigma| = n \\ \mathit{first}(\sigma) = s \\ \mathit{last}(\sigma) = s'}} P(\sigma) \cdot \Pr\{Y_t \le y \mid \sigma\}. \tag{3.2}$$

Following this expression, we consider the uniformised paths that start in s and end in s', calculate the reward distribution conditioned on each of these paths and compute the weighted sum of all conditioned probabilities. Two questions arise with this approach:

i) How do we calculate the conditional reward distribution?
ii) Is it possible to consider all relevant uniformised paths?

These two issues are addressed in the following.

The conditional reward distribution. The state space of the MRM $\mathcal{M}$ can be divided into $K+1$ distinct reward classes. States with identical reward rate constitute one such reward class. Without loss of generality, the reward classes are ordered such that

$$r_0 > r_2 > \ldots > r_K \ge 0.$$

For the computation of $\Pr\{Y_t \le y \mid \sigma\}$ it is not necessary to consider the complete information contained in the path σ but one only has to know how many epochs of the uniformised path have been spent in each of the reward classes. A vector $\mathbf{k} = (k_1, \ldots, k_K)$ recording these visit counts is called a *colouring*. The value k_i indicates the number of epochs the MRM has spent in states of reward class i. The term colouring stems from the idea of assigning the same colour to states with identical reward rate [6]. The computation of the distribution of Y_t given a colouring $\mathbf{k}$ boils down to the computation of the distribution of a linear combination of uniform order

statistics [31]. We are aware of 3 methods for the calculation of this type of distribution. The approaches of Weisberg [36] and Matsunawa [20] use involved computations and tend to be numerically unstable. The recently proposed method of Diniz et al. [8] is based on a very simple recursion scheme and is numerically stable. It is therefore the one we use in this context.

Which paths to explore? The set of all uniformised paths starting in s and ending in s' in (3.2) is partitioned according to the number of steps (the length) within a path. For a fixed starting state s there is exactly one uniformised path with 0 steps, namely s itself. The starting state has up to $|S|$ successor states, so there are $\mathcal{O}(|S|)$ uniformised paths of length 1. Repeating this argument, there are $\mathcal{O}(|S|^n)$ uniformised paths of length n. The total number of paths is of course infinite, but even if we only take into account paths up to a given length N, the number grows exponentially with N. Hence, the consideration of all paths in (3.2) is infeasible.

The probability $P(\sigma)$ of a uniformised path is used in (3.2) as a weight for the conditional reward distribution. Qureshi and Sanders [26] introduce a threshold $w \in (0,1)$ for $P(\sigma)$: only if $P(\sigma) > w$, the path σ is included in the summation. Additionally, a maximum length N is fixed for the uniformised paths. Define the set of uniformised paths of length n that are actually considered for the computation as

$$\mathit{Considered}(s,s',w,n) = \{\sigma \in \mathit{uPath} \mid \mathit{first}(\sigma)=s, \mathit{last}(\sigma)=s', P(\sigma) > w \text{ and } |\sigma| = n\}.$$

This leads to the following approximation for the reward distribution:

$$\Upsilon_{ss'}(t,y) \approx \sum_{n=0}^{N} PP(\lambda t, n) \cdot \sum_{\sigma \in \mathit{Considered}(s,s',w,n)} P(\sigma) \cdot \Pr\{Y_t \le y \mid \mathbf{k}(\sigma)\}, \tag{3.3}$$

where $\mathbf{k}(\sigma)$ is the colouring arising from path σ. For the calculation of (3.3), all paths contained in one of the sets $\mathit{Considered}(s,s',w,n)$ for $n = 0,\ldots,N$, have to be generated one by one. An error bound for the approximation can be determined in the course of the path exploration. For details on the exploration algorithm and the error bound we refer to [3, 4, 26].

3.4. Discretisation. A wide variety of general purpose numerical solution methods for ODEs and PDEs are based on the idea of *discretising* the continuous parameters. Tijms and Veldman published an approximate discretisation algorithm for the computation of $\Upsilon_{ss'}(t,y)$ that uses the same step size Δ for both time and accumulated reward [34].

Like any distribution, $\Upsilon_{ss'}(t,y)$ is a definite integral over the corresponding density:

$$\Upsilon_{ss'}(t,y) = \int_0^y \upsilon_{ss'}(t,x)dx. \tag{3.4}$$

For fixed $t > 0$ and step size Δ we can use the rectangular approximation:

$$\Upsilon_{ss'}(t,y) \approx \sum_{j=1}^{\frac{y}{\Delta}} \upsilon_{ss'}(t, j\cdot\Delta)\Delta. \tag{3.5}$$

For $\Delta \to 0$ we obtain again (3.4). Other approximation schemes, e.g., trapezoid, are possible.

We discretise the time up to t and the accumulated reward up to y in steps of size Δ and consider the density at times $0, \Delta, \cdots, \frac{t}{\Delta}\Delta$, and for accumulated rewards $0, \Delta, \cdots, \frac{y}{\Delta}\Delta$. The densities $\upsilon_{ss'}(\tau, x)$ are also not determined exactly but approximated by $\upsilon^{\Delta}_{ss'}(\tau, x)$ assuming that at most one transition has occurred in a time interval of length Δ. The possibility that two or more transitions occur is neglected. This is a reasonable assumption if Δ is small. The initial values for $\tau = 0$ are given by

$$\upsilon^{\Delta}_{ss'}(0, x) = \begin{cases} \frac{1}{\Delta}, & s = s' \text{ and } x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Only if $s = s'$ and $x = 0$ the approximate density can be positive at time 0. Since it is a derivative we have to fit it to the step size Δ.

By assuming that either no transition or exactly one transition has occurred in the time interval $[\tau, \tau + \Delta)$, the quantity $\upsilon^{\Delta}_{ss'}(\tau + \Delta, x)$ can recursively be calculated as follows:

$$\begin{aligned} \upsilon^{\Delta}_{ss'}(\tau + \Delta, x) &= \upsilon^{\Delta}_{ss'}(\tau, x - \rho_{s'}\Delta) \cdot \ (1 + Q_{s's'} \cdot \Delta) \\ &\quad + \sum_{z \neq s'} \upsilon^{\Delta}_{sz}(\tau, x - \rho_{s'}\Delta) \cdot \ Q_{zs'} \cdot \Delta. \end{aligned} \tag{3.6}$$

The above recursion only operates correctly if $(1 + Q_{ss}\Delta)$ and $(Q_{sz}\Delta)$ are indeed probabilities, that is, if Δ is small enough. This is the case for any state s' if $\Delta \leq -\frac{1}{Q_{ss}}$ for all $s \in S$ and all τ and x [33]. No error bound is known for this method.

3.5. Markovian approximation. The last algorithm we present is an approximation for the joint distribution of state and accumulated reward that is based on the transient solution of a derived CTMC, that is, no rewards are involved. It was described in [14, 15].

The joint distribution of state and accumulated reward, can be rewritten by summing over evenly-sized subintervals of the reward interval $[0, y]$:

$$\begin{aligned} \Upsilon_{ss'}(t, y) &= \Pr\{X_t = s', Y_t \in [0, \Delta y] \mid X_0 = s\} \\ &\quad + \sum_{j=1}^{\frac{y}{\Delta y}-1} \Pr\{X_t = s', Y_t \in (j\Delta y, (j+1)\Delta y] \mid X_0 = s\}. \end{aligned}$$

We want to approximate the terms $\Pr\{X_t = s', Y_t \in (j\Delta y, (j+1)\Delta y] \mid X_0 = s\}$ in such a way that the computation is done for a pure CTMC (without rewards). An MRM $\mathcal{M}$ can be seen as having an infinite and uncountable state space $S \times \mathbb{R}_{\geq 0}$. A joint state (s, y) indicates that CTMC underlying the MRM is in state s and that the accumulated reward is y. For our approximation, we break down the uncountable state space to an infinite but countable one. Define a CTMC C^{∞} with infinite state space $S \times \mathbb{N}$. The probability of being in state (s', j) is then the desired approximation:

$$\Pr\{X_t = s', Y_t \in (j\Delta y, (j+1)\Delta y] \mid X_0 = s\} \approx \Pi^{\mathcal{C}^{\infty}}_{(s,0)(s',j)},$$

where $\Pi^{\mathcal{C}^{\infty}}_{(s,0)(s',j)}$ is the transient probability of CTMC C^{∞} to reside in state (s', j) at time t, having started in state $(s, 0)$. The generator matrix $\mathbf{Q}^{\infty}$ must contain transitions from one reward level $j\Delta y$ to the next reward level $(j+1)\Delta y$. The rate at which reward is accumulated in a state s in the original MRM is ρ_s. The natural

choice for the rate of accumulating Δy reward, that is, reaching the next reward level, is $\frac{\rho_s}{\Delta y}$. Transitions between states at the same reward levels are transfered from the original MRM, leading to the following generator matrix:

$$Q^{\infty}_{(s,i)(s',j)} = \begin{cases} Q_{ss'}, & s \neq s' \text{ and } i = j, \\ Q_{ss'} - \frac{\rho_s}{\Delta y}, & s = s' \text{ and } i = j, \\ \frac{\rho_s}{\Delta y}, & s = s' \text{ and } j = i+1, \\ 0, & \text{otherwise.} \end{cases}$$

The generator matrix $\mathbf{Q}^{\infty}$ has a block diagonal structure, with the original generator matrix $\mathbf{Q}$ (with adapted diagonal entries) appearing on the diagonal and the matrix $\mathbf{D}/\Delta y$ (where $\mathbf{D}$ is the diagonal matrix arising from the reward structure ρ) appearing as upper off-diagonal. The infinite CTMC $\mathcal{C}^{\infty}$ is a quasi-birth process (a subclass of quasi-birth-death processes [23]). Transitions are only possible within a level (matrix $\mathbf{Q}$) or to the next higher level (matrix $\mathbf{D}/\Delta y$).

The transient probabilities $\Pi^{\mathcal{C}^{\infty}}_{(s,0)(s',j)}(t)$ needed for the approximation can efficiently be computed using uniformisation even though the CTMC $\mathcal{C}^{\infty}$ has infinite state space [27, 28].

We are not able to indicate the error introduced by this approximation. Of course, the approximation will get more accurate for smaller Δy. The accuracy is also influenced by the error bound used for computing the transient probabilities using uniformisation.

3.6. Complexity of the algorithms. Table 3.1 indicates the space and time complexity of Picard's method, uniformisation, discretisation and the Markovian approximation. For all four algorithms the complexity of computing $\Upsilon_{ss'}(t, y)$ for *all* pairs $s, s' \in S$ is given. For the path exploration algorithm it is not reasonable to indicate the complexity. The number of paths explored is potentially exponential, but the exploration is restricted by the weight w and the maximal path length N.

	space	time
Picard's method	$\mathcal{O}\left(\frac{\lvert S\rvert^2 ty}{(\Delta t)^2}\right)$	$\mathcal{O}\left(\frac{\lvert S\rvert^3 Nt^2 y}{(\Delta t)^3}\right)$
Uniformisation	$\mathcal{O}\big(\lvert S\rvert^2 \lambda t K\big)$	$\mathcal{O}\big(\lvert S\rvert^3 (\lambda t)^2\big)$
Discretisation	$\mathcal{O}\left(\frac{\lvert S\rvert^2 y}{\Delta}\right)$	$\mathcal{O}\left(\frac{\lvert S\rvert^3 ty}{\Delta^2}\right)$
Markovian approx.	$\mathcal{O}\left(\frac{\lvert S\rvert^2 y}{\Delta y}\right)$	$\mathcal{O}\left(\frac{\lvert S\rvert^3 ty}{(\Delta y)^2}\right)$

TABLE 3.1
Time and space complexity

The four algorithms included in the table have a complexity that is cubic in the number of states. If only a single starting state s is considered, the complexity w.r.t. the number of states is only quadratic for Picard's method, Discretisation and the Markovian approximation. If one uses a sparse representation of the generator matrix $\mathbf{Q}$, one factor $|S|$ can be replaced by ν, the average number of transitions originating from a state.

The uniformisation approach suffers from stiffness problems as normal uniformisation does, if the rates in the generator differ several orders of magnitude. In the discretisation approach, the step size Δ has to be smaller than $1/Q_{ss}$ for any state s, which may lead to an excessively high number of steps. The Markovian approximation relies again on transient solution via uniformisation and therefore also exhibits bad performance when dealing with stiff models.

4. Comparison. In this section we compare and discuss the five algorithms described in the previous section. For the comparison of the accuracy and execution times we use the small four-state illustrating example of [3]. The behaviour of the algorithms for larger state spaces is evaluated using a scalable model of a multiprocessor computer system [32]. On the basis of the numerical results we make a recommendation for the use of the different algorithms.

Experiments in the CSRL setting [3] have shown that Picard's method performs poorly with respect to both accuracy and execution time. Additionally, we have frequently encountered numerical problems with this algorithm. This algorithm is therefore not further considered.

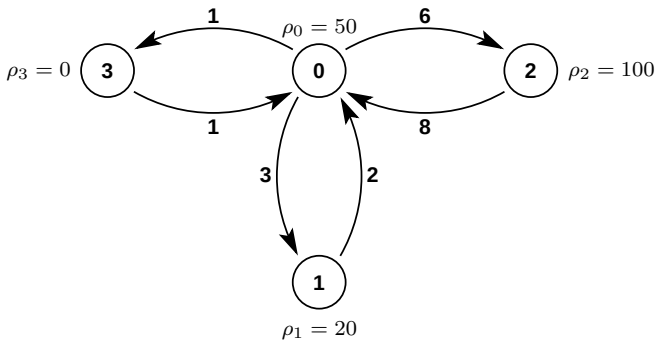

FIG. 4.1. *Four-state MRM*

4.1. Accuracy and execution times. Consider the MRM of Figure 4.1. Its four states have reward rates $\rho_0 = 50$, $\rho_1 = 20$, $\rho_2 = 100$ and $\rho_3 = 0$. In Figure 4.2 the performability distribution for this model is depicted for $t \in [0, 5]$ and $y \in [0, 200]$ and the initial distribution $\alpha = (1, 0, 0, 0)$. For $t = 0$, the performability measure is one, since $Y_0 = 0 \leq y$ for any y. With growing t and constant y, the performability measure decreases because more reward is accumulated. For constant t and growing y the performability increases because it is more likely to stay below the reward bound.

In the following we apply the algorithms to compute the performability measure for $t = 0.2$ and $y = 5$, that is, $\Pr\{Y_{0.2} \leq 5\}$.

Uniformisation. We start the discussion with the uniformisation algorithm because it is the only one that provides us with an *a priori* error bound for the results. The results are then used to compute the relative error for the numerical results of the other algorithms. Table 4.1 shows the numbers of steps, the resulting probability and the run time (user time) in seconds. The number of considered steps increases with the required accuracy, and so does the required time. Further experiments (see also [3]) have shown that also the time t has influence on the number of steps, as expected for a uniformisation-based algorithm. In contrast, the value of y has no impact on the execution time. The execution time for this small example is reasonable, staying far below one second.

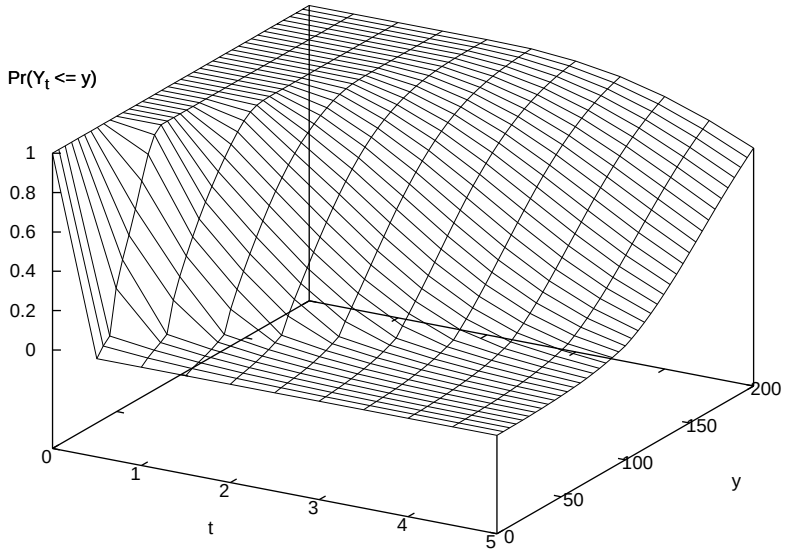

FIG. 4.2. *Performability distribution for the four-state MRM*

ε	N	$\Pr\{Y_{0.2} \leq 5\}$	time
10^{-2}	6	0.1319195485	0.005
10^{-4}	9	0.1330107485	0.010
10^{-8}	14	0.1330224790	0.022
10^{-16}	22	0.1330224800	0.051

TABLE 4.1
Results for the uniformisation algorithm

Path exploration. With path exploration, an error bound can only be determined a posteriori. In Table 4.2 we list the results of this algorithm. Each row shows the weight for the paths, the number c of colourings considered for this weight, the probability, the a posteriori error bound, the relative error with respect to the results of the uniformisation algorithm and the execution time in seconds. We have chosen the maximum number of steps such that it does not influence the path generation; hence, paths are only discarded because of their weight. Clearly, with decreasing path weight w, the error bounds decrease and the execution time increases. Other experiments [3] have shown that the error bound and the execution time increase with growing t. As it was the case for the uniformisation algorithm, y does not have any influence on the execution time, nor does it affect the error bound obtained. The execution times are reasonable for this small example but for comparable accuracies, uniformisation is faster. Uniformisation also has the advantage of providing an a priori error bound.

w	c	$\Pr\{Y_{0.2} \leq 5\}$	absolute error	relative error	time
10^{-1}	24	0.0844766620	0.2845779895	0.3649444668	0.003
10^{-2}	173	0.1286911957	0.0483092418	0.0325605437	0.020
10^{-3}	597	0.1327120417	0.0037866433	0.0023337283	0.086
10^{-4}	1709	0.1330023912	0.0002745972	0.0001510179	0.421
10^{-5}	3357	0.1330217567	0.0000122965	0.0000054377	2.000
10^{-6}	4758	0.1330224461	0.0000005541	0.0000002551	9.758
10^{-7}	5870	0.1330224791	0.0000000176	0.0000000063	43.69

TABLE 4.2
Results for the path exploration

Discretisation. Table 4.3 presents the resulting probability, the relative error and the execution time of the discretisation algorithm subject to the step size Δ. With decreasing Δ, the relative error decreases and the execution time increases. Further experiments [3] have shown that both t and y have an influence on the execution time. Dividing the step size Δ by 10 increases the execution time by a factor 100 which confirms that the algorithm has a time complexity in Δ^{-2}. The values show that the discretisation algorithm provides usable results in tolerable time but is much slower than uniformisation and path exploration.

Δ	$\Pr\{Y_{0.2} \leq 5\}$	relative error	time
10^{-2}	0.0773048006	0.4188591236	0.006
10^{-3}	0.1270651788	0.0447841688	0.543
10^{-4}	0.1324221136	0.0045132699	57.18

TABLE 4.3
Results for discretisation

Markovian approximation. The last algorithm to evaluate is the Markovian approximation. Table 4.4 shows the probability, the relative error and the execution time depending on the step size Δy for the accumulated reward. For the calculation of the transient probabilities via (ordinary) uniformisation we have chosen the error bound $\varepsilon = 10^{-16}$. The execution time grows with Δy^{-2}: if we divide Δy by 10, the run time is multiplied with 100. Further experiments [3] have shown that the execution time also depends linearly on t and y. The Markovian approximation is quite fast while at the same time providing adequately accurate results.

Δy	$\Pr\{Y_{0.2} \leq 5\}$	relative error	time
10^{-1}	0.1294067747	0.0271811597	0.003
10^{-2}	0.1324884190	0.0040148179	0.232
10^{-3}	0.1329690459	0.0004016921	20.56

TABLE 4.4
Results for the Markovian approximation

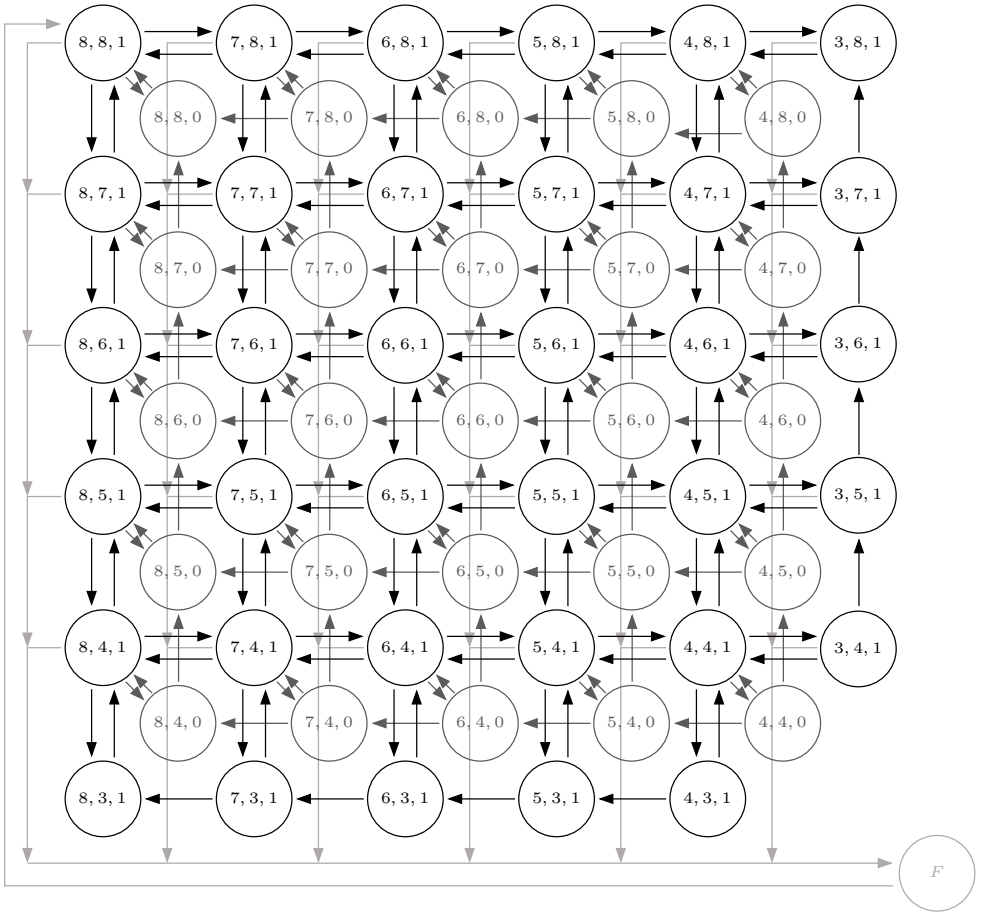

FIG. 4.3. *State-transition diagram for the multiprocessor model with* $M = 8$

4.2. Scaling the state space. The results shown in the previous section suggest that the uniformisation algorithm is fast and the most accurate of the four algorithms, However, we have not shown the behaviour of the algorithms for different sizes of the state space. In this section we compare the execution times for growing state spaces. For this purpose we adapt the multiprocessor model from [32, Section III].

The multiprocessor model. We address a multiprocessor system with M processors and M memory modules and an interconnecting network (crossbar switch). The states of the MRM are triples (i, j, k), where i is the number of operating processors, j is the number of operating memories, and $k = 1$ if the network is operational and $k = 0$ if the network is broken. Each of the components can fail. After the failure of a processor or a memory a reconfiguration takes place leading to a state where one of the component indices is decreased. The probability $c = 0.9$ for this reconfiguration to be successful is known as coverage. The component is then locally repaired by a single repair unit specific to each component type. In case the reconfiguration does not complete successfully, a global failure state F is entered. A global repair takes longer than a local component repair and always restores the system in fully operational state.

To be operational, a minimum of four operating processors and four operating

memories is required, and the switch has to be operational as well. We assume that there are no further failures once the system is non-operational. With this restriction, the MRM for a system with M processors and memories has $(M-2)^2 + (M-3)^2$ states. The state-transition diagram for this model with $M = 8$ can be found in Figure 4.3.

We consider a single initial state (instead of an initial distribution), namely the state where all components are non-failed. The parameters of the system are given in Table 4.5; the repair rates are taken from the original model in [32]. The original failure rates are scaled up to get more meaningful results for the numerical comparison.

transition		rate
single processor failure rate	λ	0.0689
single memory failure rate	γ	0.2241
switch failure rate	δ	0.2024
processor repair rate	ν	2.0
memory repair rate	η	1.0
switch repair rate	ϵ	0.5
global repair rate	μ	0.2

TABLE 4.5
Rates for the multiprocessor model

The reward rate of an operational state is defined to be the average number of busy memories. Following [1], this equals

$$\rho_{(i,j,1)} = m\left(1 - \left(1 - \frac{1}{m}\right)^{l}\right),$$

where $l = \min(i,j)$ and $m = \max(i,j)$. The reward rate of non-operational states is zero. The accumulated reward Y_t can then be interpreted as the amount of work that has been completed up to time t.

By varying the number of processors and memory modules M between 4 and 240 we obtain MRMs with 5 to 112813 states. For these models the performability measure $\Pr\{Y_{10} \leq 10\}$, that is, the probability that the amount of performed work at time instant 10 is at most 10, should be computed. As basis for a comparison we take the smallest MRM with 5 states and compute the performability measure using the uniformisation algorithm with error bound $\varepsilon = 10^{-5}$. For the other algorithms we choose the parameters in such a way that the relative error is below 3% and keep these algorithm settings as such.

Figure 4.4 shows the execution time in seconds (y-axis) for the computation of $\Pr\{Y_{10} \leq 10\}$ in the size-varying MRMs (x-axis in logarithmic scale) for the different algorithms, as long as they stay below 5 minutes. For path exploration, discretisation and the Markovian approximation there are two implementations each, one using a dense and one using a sparse representation of the generator matrix. Uniformisation (which has cubic time complexity in the number of states) is only possible for small state spaces; already for $M = 8$ (61 states), the execution time exceeds 5 minutes.

Discretisation is only directly suited for MRMs with integer reward rates. The reward rates of the multiprocessor model are non-integers. To overcome this problem we can scale the reward rates *and* the reward bound y by a factor that makes the rates integers. Unfortunately, this factor also impacts the execution time of the algorithm, since it increases the number of reward steps to be taken, cf. Section 3.4. We choose to scale the rates and y by a factor 10 only and take the integer part of that. In order to have an relative error w.r.t. uniformisation for $M = 4$ below 3%, we set $\Delta = 0.02$. As can be seen from the figure, even with this small factor, discretisation with a dense representation of $\mathbf{Q}$ is slower than uniformisation, while the sparse version performs slightly better.

For the Markovian approximation, the allowed relative error of 3% leads to a step size $\Delta y = 1$. The dense implementation of the algorithm manages up to 1741 states ($M = 32$) in less than 5 minutes. The quadratic dependency on the number of states in clearly visible. The implementation using a sparse representation has an execution time of about 5 minutes for a model with more than 112000 states ($M = 240$).

Finally, both implementations of the path exploration algorithm are not able to deliver a result for $M = 4$ with a relative error below 3% in less than 5 minutes. This algorithm is therefore not included in Figure 4.4.

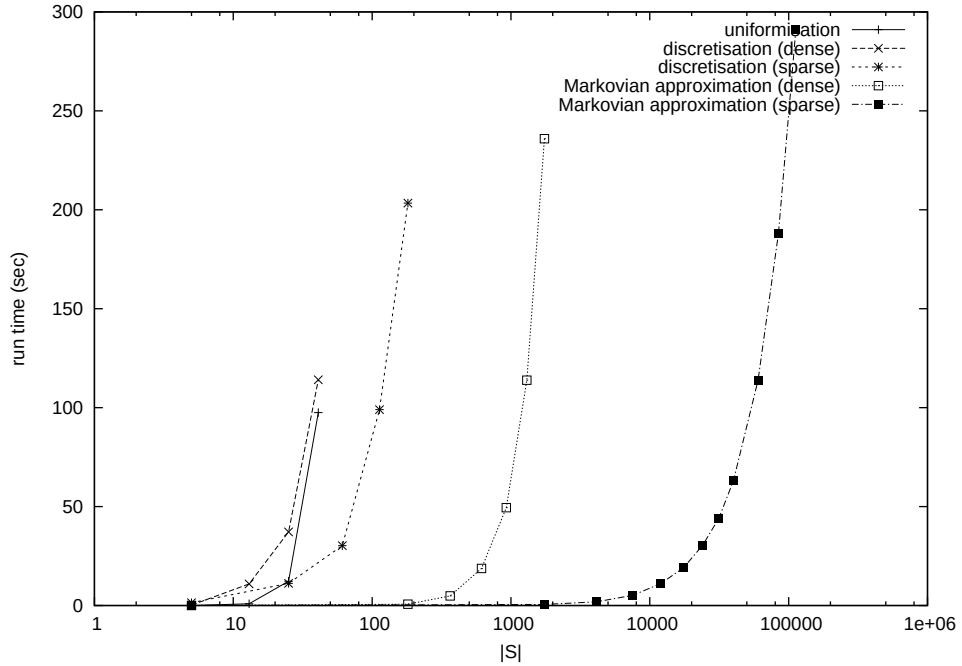

FIG. 4.4. *Execution times for uniformisation, discretisation and Markovian approximation when scaling the size of the state space*

Table 4.6 shows the values for accomplishing an amount of work less than 10 by time 10, i.e., $\Pr\{Y_{10} \leq 10\}$, computed by uniformisation, the Markovian approximation and discretisation. This probability is quite high for the model with only four processors and memory modules (43.4%). When increasing the number of processors and memories M, the performability measure decreases, as the maximum bandwidth increases (from less than 3 for $M = 4$ to approximately 20 for $M = 32$) and it is

less likely to reside in the failure state, having started in the state where all components are operational. The relative error for the Markovian approximation and the discretisation behaves in a non-predictable way.

	uniformisation	Markovian approximation		discretisation	
M	$\Pr\{Y_{10} \leq 10\}$	$\Pr\{Y_{10} \leq 10\}$	rel. error	$\Pr\{Y_{10} \leq 10\}$	rel. error
4	0.43406876	0.42224274	0.02724458	0.42504043	0.02079930
5	0.21462346	0.22717613	0.05848695	0.20817395	0.03005031
6	0.16067183	0.17011596	0.05877896	0.15614911	0.02814883
7	0.13737533	0.14419586	0.04964886	0.12886252	0.06196754
8	-	0.12984323	-	0.11656597	-
10	-	0.11441596	-	0.10151163	-
12	-	0.10601227	-	0.09367540	-
32	-	0.08532634	-	-	-
64	-	0.08038803	-	-	-
128	-	0.07809976	-	-	-
240	-	0.07706989	-	-	-

TABLE 4.6
Performability measure $\Pr\{Y_{10} \leq 10\}$ *for the multiprocessor model*

5. Conclusions. In this paper we discussed five algorithms for the computation of the performability distribution of MRMs with rate rewards. All of them are applicable to MRMs with arbitrary structure.

Based on the numerical results, we recommend the uniformisation algorithm whenever the model is small (a few dozens of states). It is the only algorithm for which an error bound can be determined a priori. For larger models, the Markovian approximation seems to be the only applicable algorithm. For a larger model with integer reward rates, the discretisation algorithm could also be employed. The results of both algorithms have to be handled with care because no error bound is provided. The path exploration algorithm performed badly when applied to the multiprocessor model. For other models, where less paths through the state space are possible or the probability mass is concentrated on a few path only, it might be competitive. Picard's method has only been studied for completeness, it is inferior to the other algorithms in accuracy and execution time.

REFERENCES

[1] D. BHANDARKAR, *Analysis of memory interference in multiprocessors*, IEEE Transactions on Computers, C-24 (1975), pp. 897–908.
[2] B. CICIANI AND V. GRASSI, *Performability evaluation of fault-tolerant satellite systems*, IEEE Transactions on Communications, 35 (1987), pp. 403–409.
[3] L. CLOTH, *Model Checking Algorithms for Markov Reward Models*, PhD thesis, University of Twente, 2006.
[4] L. CLOTH, J.-P. KATOEN, M. KHATTRI, AND R. PULUNGAN, *Model checking Markov reward models with impulse rewards*, in International Conference on Dependable Systems and Networks (DSN'05), IEEE Press, 2005, pp. 722–731.

[5] E. de Souza e Silva and H. R. Gail, *Calculating cumulative operational time distributions of repairable computer systems*, IEEE Transactions on Computers, C-35 (1986), pp. 322–332.

[6] E. de Souza e Silva, H. R. Gail, and R. Vallejos Campos, *Calculating transient distributions of cumulative reward*, ACM SIGMETRICS Performance Evaluation Review, 23 (1995), pp. 231–240.

[7] N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt, *Response time densities in generalised stochastic Petri net models*, in Proceedings of the 3rd international workshop on Software and performance (WOSP'02), ACM Press, 2002, pp. 46–54.

[8] M. C. Diniz, E. de Souza e Silva, and H. R. Gail, *Calculating the distribution of a linear combination of uniform order statistics*, INFORMS Journal on Computing, 14 (2002), pp. 124–131.

[9] L. Donatiello and B. R. Iyer, *Analysis of a composite performance reliability measure for fault-tolerant systems*, Journal of the ACM, 34 (1987), pp. 179–199.

[10] D. Furchtgott and J. F. Meyer, *Performability solution method for degradable nonrepairable systems*, IEEE Transactions on Computers, 33 (1984), pp. 550–553.

[11] A. Goyal and A. Tantawi, *Evaluation of performability for degradable computer systems*, IEEE Transactions on Computers, 36 (1987), pp. 738–744.

[12] W. K. Grassmann, *Finding transient solutions in Markovian event systems through randomization*, in Numerical Solution of Markov Chains, W. Stewart, ed., Marcel Dekker Inc., 1991, pp. 357–371.

[13] D. Gross and D. R. Miller, *The randomization technique as a modeling tool and solution procedure for transient Markov processes*, Operations Research, 32 (1984), pp. 343–361.

[14] B. R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier, *Model checking performability properties*, in Proceedings of the International Conference on Dependable Systems and Networks (DSN'02), IEEE Press, 2002, pp. 102–112.

[15] B. R. Haverkort, H. Hermanns, J.-P. Katoen, and C. Baier, *Model checking CSRL-specified performability properties*, in Proceedings of the 5th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS'01), 2001, pp. 105–109.

[16] B. R. Haverkort, R. Marie, G. Rubino, and K. S. Trivedi, eds., *Performability Modelling*, John Wiley & Sons, 2001.

[17] B. R. Haverkort and I. G. Niemegeers, *Performability modelling tools and techniques*, Performance Evaluation, 25 (1996), pp. 17–40.

[18] M. Khattri and R. M. Pulungan, *Model checking Markov reward models with impulse rewards*, Master's thesis, University of Twente, Department of Computer Science, 2004.

[19] V. G. Kulkarni, V. F. Nicola, R. M. Smith, and K. S. Trivedi, *Numerical evaluation of performability and job completion time in repairable fault-tolerant systems*, in Proceedings of the 16th International Symposium on Fault-Tolerant Computing (FTCS'85), 1985, pp. 252–257.

[20] T. Matsunawa, *The exact and approximate distributions of linear combinations of selected order statistics from uniform distributions*, The Annals of the Institute of Statistical Mathematics, 37 (1985), pp. 1–16.

[21] J. F. Meyer, *On evaluating the performability of degradable computing systems*, IEEE Transactions on Computers, 29 (1980), pp. 720–731.

[22] ———, *Performability: a retrospective and some pointers to the future*, Performance Evaluation, 14 (1992), pp. 139–156.

[23] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach*, Dover Publications, 1994.

[24] K. R. Pattipati, R. Mallubhatla, V. Gopalakrishna, and N. Viswanatham, *Markov-reward models and hyperbolic systems*, in Performability Modelling, John Wiley & Sons, 2001, pp. 83–106.

[25] M. A. Qureshi, *Reward model solution methods with impulse and rate rewards: An algorithm and numerical results*, Master's thesis, University of Arizona, 1992.

[26] M. A. Qureshi and W. H. Sanders, *Reward model solution methods with impulse and rate rewards: an algorithm and numerical results*, Performance Evaluation, 20 (1994), pp. 413–436.

[27] A. Remke, L. Cloth, and B. R. Haverkort, *Uniformization with representatives: Transient analysis of infinite-state Quasi-Birth-Death processes. Submitted for publication*, 2006.

[28] A. Remke, B. R. Haverkort, and L. Cloth, *Model checking infinite-state Markov chains*, in Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05), Lecture Notes in Computer Science 3440, Springer-Verlag, 2005, pp. 237–252.

[29] G. Rubino and B. Sericola, *Interval availability distribution computation*, in Proceedings of the 23rd IEEE International Symposium on Fault Tolerant Computing (FTCS'93), IEEE Press, 1993, pp. 48–55.

[30] B. Sericola, *Closed-form solution for the distribution of the total time spent in a subset of states of a homogeneous Markov process during a finite observation period*, Journal of Applied Probability, 27 (1990), pp. 713–719.

[31] ———, *Occupation times in Markov processes*, Communications in Statistics — Stochastic Models, 16 (2000), pp. 479–510.

[32] R. Smith, K. S. Trivedi, and A. Ramesh, *Performability analysis: Measures, an algorithm and a case study*, IEEE Transactions on Computers, 37 (1988), pp. 406–417.

[33] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, 1994.

[34] H. Tijms and R. Veldman, *A fast algorithm for the transient reward distribution in continuous-time Markov chains*, Operations Research Letters, 26 (2000), pp. 155–158.

[35] K. S. Trivedi, J. K. Muppala, S. Woolet, and B. R. Haverkort, *Composite performance and dependability analysis*, Performance Evaluation, 14 (1992), pp. 197–215.

[36] H. Weisberg, *The distribution of linear combinations of order statistics from the uniform distribution*, Annals of the Institute of Statistics, 42 (1971), pp. 704–709.