NUMERICAL SOLUTION OF NON-HOMOGENEOUS MARKOV PROCESSES THROUGH UNIFORMIZATION[1]

Aad P. A. van Moorsel† and Katinka Wolter‡

† Distributed Software Research Department
Bell Laboratories Research, Lucent Technologies
600 Mountain Ave., Murray Hill, NJ 07974, USA
aad@bell-labs.com

‡ Technische Universität Berlin
Institut für Technische Informatik
Franklinstr. 28–29, 10587 Berlin, Germany
katinka@cs.tu-berlin.de

ABSTRACT

Numerical algorithms based on uniformization have been proven to be numerically stable and computationally attractive to compute transient state distributions in homogeneous continuous-time Markov chains. Recently, Van Dijk [van Dijk 1992] formulated uniformization for *non*-homogeneous Markov processes, and it is of interest to investigate numerical algorithms based on uniformization for non-homogeneous models as well. We will introduce three different uniformization-based algorithms for non-homogeneous Markov processes. We implement all three algorithms, and use an example of a duplex safety model to compare the algorithms with respect to accuracy, memory use and speed. Based on these results, we provide guidelines for the implementation and application of uniformization-based algorithms for non-homogeneous Markov processes.

1 INTRODUCTION

Computing and communication systems often exhibit time-dependent behavior that can be modeled naturally by non-homogeneous continuous-time Markov processes [Platis et al. 1997, Rindos et al. 1995]. In such Markov processes, the intensity of state transitions varies with time, thus making it possible to represent time-dependent system load or time-dependent component failure rates. In this paper, we investigate the numerical solution of the transient state probability vector of non-homogeneous Markov processes by algorithms that are based on uniformization [Grassmann 1991, Jensen 1953].

Uniformization transforms a continuous-time Markov chain into a discrete-time Markov chain, sub-ordinated to a Poisson process (see Section 2 for further explanation). Numerical algorithms based on uniformization have been very successfully applied in homogeneous Markov models [Grassmann 1991, van Moorsel 1993]. The algorithms are stable, provide proven accuracy [van Moorsel and Sanders 1997], and are typically fast irrespective of the desired accuracy. Given these properties in the homogeneous case, it is of obvious interest to investigate the use of uniformization-based algorithms for non-homogeneous processes as well.

We will present, implement and compare three uniformization-based algorithms. All three algorithms numerically solve the integral equations that constitute the uniformization equation as derived by Van Dijk [van Dijk 1992]. The algorithms are: 1) base algorithm, which is a direct implementation of the discretization approach proposed in [van Dijk 1992], 2) correction algorithm, which corrects for the assumption in the base algorithm that all discretization terms can be weighed equally, and 3) interval splitting, which first splits up the time interval, and then applies either of the first two algorithms to the individual intervals.

The algorithms all have the same benefit as uniformization algorithms in the homogeneous case, in that numerical stability is guaranteed because all operations are on stochastic matrices, constituting all positive elements between 0 and 1. The achievable accuracy of uniformization for non-homogeneous processes is, however, less predictable than in the homogeneous case, since time-discretization is required to deal with the time-dependent parameters. In developing and applying the algorithms, we therefore pay considerable attention to improving accuracy. In addition, the algorithms aim at high efficiency with regard to computation time.

The algorithms have all three been implemented in Mathematica, and we apply them in this paper to a simple duplex safety model. Based on the obtained results, we give suggestions when to use each method, and how to implement them so they exhibit optimal characteristics with respect to speed and accuracy. In particular, we will find that one should apply the interval splitting algorithm up to a certain threshold, and apply the base or correction algorithm to the respective intervals. Other issues, such as choosing a matrix that is representative

[1] This work was initiated while Aad van Moorsel was a visiting researcher at TU Berlin, October 1997, supported by German Research Council under grant number GRK 86/2.

for a discretization interval, and the discretization step size, are also studied, and we will give heuristic guidelines for tuning these parameters.

2 UNIFORMIZATION

A non-homogeneous Markov process on the state space S, is defined by the initial state distribution π_0, and the generator matrix $Q(t), t \geq 0$, with elements that are functions of time. The generator matrix is defined as usual: the (i,j)-th element $Q_{i,j}(t), i \neq j$, corresponds to the transition rate from state i to state j at time t, and the diagonal elements $Q_{i,i}(t)$ represent the negative sum of transition rates out of state i:

$$Q_{i,i}(t) = -\sum_{j \neq i} Q_{i,j}(t).$$

The dynamics of a non-homogeneous Markov process are given by Kolmogorov's differential equations:

$$\frac{d\pi(t)}{dt} = \pi(t)Q(t), \quad t \geq 0, \tag{1}$$

where $\pi(t)$ is the vector of state probabilities at time t. The initial value for $t = 0$ is given by the initial distribution π_0. Obtaining the transient state vector $\pi(t)$ thus involves solving eqn. (1). We will pursue the solution by means of uniformization, and will now show how the uniformization equation can be obtained by purely algebraic means.

The closed-form solution of (1) is given by the Peano-Baker series [Antsaklis and Michel 1997]:[1]

$$\pi(T) = \pi_0 \sum_{k=0}^{\infty} \int_{s_1=0}^{T} \int_{s_2=s_1}^{T} \cdots \int_{s_k=s_{k-1}}^{T} Q(s_1)Q(s_2)\ldots Q(s_k)ds_1 ds_2 \ldots ds_k. \tag{2}$$

For $k = 0$ the integral is defined as the identity matrix I, and we set $s_0 = 0$. Let the probability matrices $P(t), t \geq 0$, be defined as

$$P(t) = I + \frac{Q(t)}{\lambda}, \tag{3}$$

where, for the uniformization rate λ:

$$\lambda \geq inf_{t \in [0,T], i \in S} |Q_{i,i}(t)|. \tag{4}$$

It is now a matter of straightforward manipulation of the Peano-Baker series in eqn. (2) to derive the uniformization equation. First, from (3) we have $Q(t) = \lambda(P(t) - I)$, and thus we can substitute $Q(\cdot)$ in (2) by $\lambda(P(\cdot) - I)$:

$$\pi(T) = \pi_0 \sum_{k=0}^{\infty} \int_{s_1=0}^{T} \int_{s_2=s_1}^{T} \cdots \int_{s_k=s_{k-1}}^{T} \lambda^k (P(s_1) - I) \cdots (P(s_k) - I) ds_1 \cdots ds_k. \tag{5}$$

[1] Note that (2) actually is a 'transpose' of the Peano-Baker series, since equation (1) is the transposed formulation of the usual problem statement $\dot{x} = A(t)x$ [Antsaklis and Michel 1997]. This also might explain the mistaken expression (8.4) in [Rindos et al. 1995].

When rewriting the above expression we use that for any $m \geq 1$ it holds that:

$$\int_{s_1=0}^{T} \int_{s_2=s_1}^{T} \cdots \int_{s_m=s_{m-1}}^{T} ds_1 \cdots ds_m = \frac{T^m}{m!}, \tag{6}$$

and we then obtain:

$$\pi(T) = \pi_0 \sum_{k=0}^{\infty} \sum_{n=0}^{k} \lambda^k \frac{(-1)^{k-n} T^{k-n}}{(k-n)!} \int_{s_1=0}^{T} \int_{s_2=s_1}^{T} \cdots \int_{s_n=s_{n-1}}^{T} P(s_1) \cdots P(s_n) ds_1 \cdots ds_n. \tag{7}$$

Swapping the sums, using that by definition $e^{-\lambda T} = \sum_{k=n}^{\infty} \frac{\lambda^{k-n}(-T)^{k-n}}{(k-n)!}$, and introducing terms $\frac{n!}{T^n}$ in the integral, (7) can then be equated to the *uniformization equation* [van Dijk 1992]:

$$\pi(T) = \pi_0 \sum_{n=0}^{\infty} \frac{(\lambda T)^n}{n!} e^{-\lambda T} \int_{s_1=0}^{T} \int_{s_2=s_1}^{T} \cdots \int_{s_n=s_{n-1}}^{T} P(s_1) \cdots P(s_n) \frac{n!}{T^n} ds_1 \cdots ds_n. \tag{8}$$

The uniformization equation (8) has a useful intuitive meaning. The terms $\frac{(\lambda T)^n}{n!} e^{-\lambda T}$ are Poisson probabilities, and the matrices $P(\cdot)$ are stochastic matrices, defining the jump probabilities of a discrete-time Markov chain. The matrix products $P(s_1) \cdots P(s_n)$ give the transition matrix after n events, where the events are generated by the Poisson process and occur at times $s_1, \ldots, s_n$. The integration in eqn. (8) takes care of all possible sets of time points $\{s_1, \ldots, s_n\}$ at which events in the Poisson process can take place in the interval $[0, T]$. Note that the terms $\frac{n!}{T^n}$ inside the integrals correspond to the density of the order statistics of an n-dimensional uniform distribution over $[0, T]^n$ [van Dijk 1992]. This density is introduced by the Poisson process, for which it is known that when n events occur in the interval $[0, T]$, these events are uniformly distributed over $[0, T]$ [Çinlar 1975].

The above algebraic derivation is by no means the only way to derive the uniformization equation. In fact, this paper is motivated by a recent paper by Van Dijk [van Dijk 1992], in which the uniformization equation is derived using stochastic arguments. Furthermore, using probabilistic arguments, uniformization for non-homogeneous and non-Markov systems has also been developed for discrete-event simulation purposes by Lewis and Shedler [Lewis and Shedler 1979], and has been deployed by, e.g., Nicola *et al.* [Nicola et al. 1992]. It should be noted that, although we concentrate on non-homogeneous Markov processes in this paper, the uniformization method can also be used for general distributions, using hazard rates of general distributions as the transition rates [Nicola et al. 1992].

3 ALGORITHMS

We present three uniformization-based algorithms, which we name 1) base algorithm, 2) correction algorithm, and 3) interval splitting algorithm. The base algorithm and correction algorithm are direct solutions of the uniformization equation. Both algorithms are based on discretization of time, and truncation of the infinite sum in (8). They differ, however, in the weights that are assigned to parts of the discretization space. The interval splitting algorithm follows a different approach: the interval $[0, T]$ is first split in subintervals, after which either the base or correction algorithm is applied to each subinterval. The interval splitting algorithm aims at speed up and reduced memory usage, while the correction algorithm aims at improved accuracy, compared to the base algorithm.

The truncation and discretization approach we use is as follows. Let N be the truncation point of the infinite sum in (8), and let Δ be the length of the discretization interval. Then the total number of discretization intervals in $[0, T]$ equals $K = \frac{T}{\Delta}$. Using the superscript c to denote the *computed* value of a variable, the transient state probabilities are computed as:

$$\pi^c(T) = \pi_0 \sum_{n=0}^{N} \frac{(\lambda T)^n}{n!} e^{-\lambda T}$$
$$\sum_{k_1=1}^{K} \sum_{k_2=k_1}^{K} \cdots \sum_{k_n=k_{n-1}}^{K} P_{k_1} P_{k_2} \cdots P_{k_n} p_K(k_1, k_2, \ldots, k_n), \tag{9}$$

where the elements $p_K(k_1, k_2, \ldots, k_n)$ function as weights for the terms in the summation (more about this later), and $P_{k_j}, j = 1, \ldots, n$, are probability matrices representing the k_j-th discretization interval. Note, that the subscript k_j in P_{k_j} denotes an index, not a time point.

3.1 Base Algorithm

In the base algorithm, which follows the discretization in Van Dijk [van Dijk 1992], all weights $p_K(k_1, k_2, \ldots, k_n)$ are equal, for any given $n, n = 0, \ldots, N$. Let $p_{K,n}$ be that weight for n, $n = 0, \ldots, N$. Since (9) must result in a probability vector, $p_{K,n}$ must equal[2]

$$p_{K,n} = 1 / \sum_{k_1=1}^{K} \sum_{k_2=k_1}^{K} \cdots \sum_{k_n=k_{n-1}}^{K} 1. \tag{10}$$

We develop an algorithm that obtains results for all $n = 0, \ldots, N$, in recursive fashion. Let the vectors $A(n, k)$ be

[2] [van Dijk 1992] uses $p_{K,n} = \frac{n!}{K^n}$, but this does not define a probability distribution over the summation space.

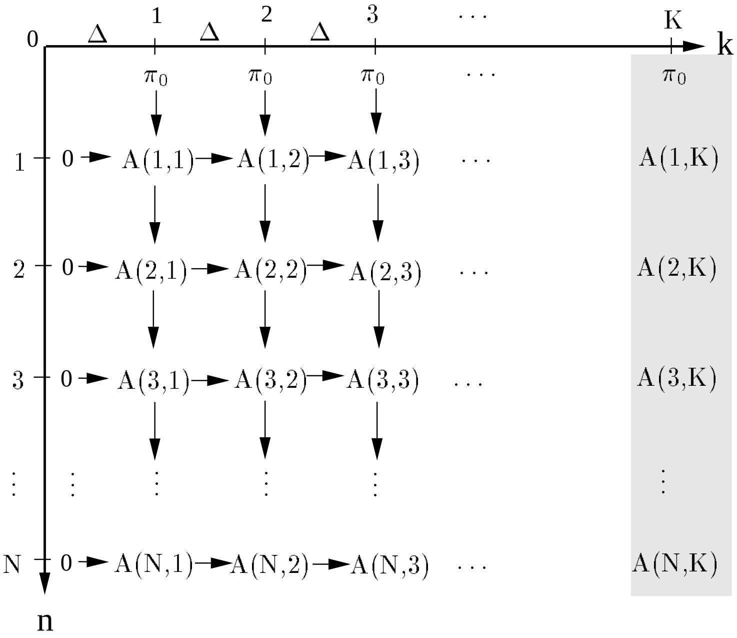

Figure 1: The base algorithm

defined as:

$$A(n, k) = \pi_0 \sum_{k_1=1}^{k} \sum_{k_2=k_1}^{k} \cdots \sum_{k_n=k_{n-1}}^{k} P_{k_1} P_{k_2} \cdots P_{k_n}, \tag{11}$$

for $n = 0, \ldots, N$, $k = 1, \ldots, K$. $A(n, k)$ represent the (not necessarily normalized) state "probability" vector after exactly n jumps in the Poisson process with rate λ, and exactly k discretization steps. To compute terms $A(n, k)$ the following recursive relation can be obtained from (11):

$$A(n, k) = A(n, k-1) + A(n-1, k) P_k. \tag{12}$$

Eqn. (12) gives rise to a recursive algorithm, for which the following starting conditions are needed:

$$\begin{aligned} A(0, k) &= \pi_0, \quad k \geq 1, \\ A(n, 0) &= 0, \quad\ \ n \geq 1. \end{aligned}$$

The numerical scheme that results is depicted in Figure 1. The arrows denote the recursive structure of the computation, following expression (12).

The terms $A(n, k)$ have values that grow rapidly, and in the implementation we therefore scale continuously. The scaling is based on Pascal's triangle, whose elements correspond to the sum of the terms in the vectors $A(n, k)$. Instead of (12), we compute:

$$A(n, k) = \frac{k-1}{k+n-2} A(n, k-1) + \frac{n-1}{k+n-2} A(n-1, k) P_k. \tag{13}$$

This way of scaling assures that the terms $A(n, k)$ are probability vectors for all n and k. Implicitly, (13) in fact includes the weights $p_{k,n}$ in the computation, recursively for all values of $n = 0, \ldots, N$, and $k = 1, \ldots, K$.

Once the terms $A(n, K)$, for $n = 0, \ldots, N$, have been computed, multiplication with Poisson probabilities must be carried out. Hence, the base algorithm completes by computing:

$$\pi^c(T) = \sum_{n=0}^{N} \frac{(\lambda T)^n}{n!} e^{-\lambda T} A(n, K). \tag{14}$$

Note that normalization is required if (12) is used, but is automatically taken care of if (13) is used.

Important from an implementation perspective is that not all terms $A(n, k), n = 0, \ldots, N$, $k = 1, \ldots, K$, need to be kept in memory indefinitely. From (13), as well as from Figure 1, it can be seen that the term $A(n, k)$ is no longer needed if $A(n + 1, k)$ as well as $A(n, k + 1)$ have been computed. Different algorithms can be thought of, but with regard to memory usage, it is best to store P_k for only one value of k at a time. Therefore, we iterate in k: we use P_k to compute $A(n, k)$ for all n and fixed k, and then increase k by one, compute $A(n, k + 1)$, etc. In terms of Figure 1 we go from top to bottom, column after column. At the end of the recursion, multiplication with Poisson probabilities is carried out for the elements in the shaded column.

3.2 Correction Algorithm

The weights $p_K(k_1, k_2, \ldots, k_n)$ in (9) provide a means to express the relative importance of the matrix products $P_{k_1} P_{k_2} \cdots P_{k_n}$. In the base algorithm all weights were taken equal, but this is not correct because different terms $P_{k_1} P_{k_2} \ldots P_{k_n}$ represent different parts of the integration space in (8). The correction algorithm therefore determines values for the weights corresponding to the relative contribution of the terms $P_{k_1} P_{k_2} \ldots P_{k_n}$ to the integral in (8). (We use the term correction algorithm for this variation of the base algorithm, because it corrects for the assumption that all weights can be taken equal.)

We will show that when m of the n Poisson events happen in the same discretization interval, the weights should be decreased by a factor $(m!)$. Define the intervals $D_j, j = 1, \ldots, n$, as $D_j = [(k_j - 1)\Delta, k_j \Delta]$. The discretization in (9) evaluates the normalization equation fully correct if, for all combinations $\{k_1, k_2, \ldots, k_n\}$,

$$P_{k_1} \cdots P_{k_n} p_K(k_1, k_2, \ldots, k_n) = \int_{s_1 \in D_1} \int_{s_2 \in D_2, s_2 \geq s_1} \cdots \int_{s_n \in D_n, s_n \geq s_{n-1}} P(s_1) \cdots P(s_n) \frac{n!}{T^n} ds_1 \cdots ds_n, \tag{15}$$

since then the multiple sum in (9) equates to the multiple integral in (8). Note that the conditions $s_j \geq s_{j-1}, j = 2, 3, \ldots, n$, are required since ordering of events is otherwise not preserved if two or more intervals $D_j, j = 1, \ldots, n$, are identical. As in (9), let $P_{k_j}, j = 1, \ldots, n$, represent $P(s_j)$, for $s_j \in D_j$. Then (15) becomes:

$$P_{k_1} \cdots P_{k_n} p_K(k_1, k_2, \ldots, k_n) = P_{k_1} \cdots P_{k_n} \frac{n!}{T^n} \int_{s_1 \in D_1} \int_{s_2 \in D_2, s_2 \geq s_1} \cdots \int_{s_n \in D_n, s_n \geq s_{n-1}} ds_1 \cdots ds_n.$$

In other words, we obtain the following expression for the weights:

$$p_K(k_1, k_2, \ldots, k_n) = \frac{n!}{T^n} \int_{s_1 \in D_1} \int_{s_2 \in D_2, s_2 \geq s_1} \cdots \int_{s_n \in D_n, s_n \geq s_{n-1}} ds_1 \ldots ds_n. \tag{16}$$

From eqn. (16) we see that the weights are determined by the outcome of the integral, which in turn depends on whether multiple events occur in the same interval (according to eqn. (6)). In general, we may consider $n\prime \leq n$ disjunct intervals, with m_j events occurring in the j-th interval D_j ($\sum_{j=1}^{n\prime} m_j = n$). The weight for the term $P_{k_1}^{m_1} P_{k_2}^{m_2} \ldots P_{k_{n\prime}}^{m_{n\prime}}$ then is:

$$p_K(k_1, k_2, \ldots, k_{n\prime}) = \frac{1}{m_1! m_2! \ldots m_{n\prime}!} \left(\frac{n!}{T^n} \Delta^n\right). \tag{17}$$

In the implementation of the correction algorithm, we keep track of the number of events occurring within an interval ($m_j, j = 1, 2, \ldots, n\prime$, in (17)). As a consequence, if we allow up to M events per discretization interval, $M + 1$ vectors are needed for storage. We obtain vectors $A(n, k, m)$, which stand for the unnormalized state "probability" vector after n jumps in the Poisson process, k discretization steps, and m events in the k-th discretization step. The iteration becomes:

$$\begin{array}{rcl} A(n,k,0) & = & \sum_{m=0}^{M} A(n,k-1,m), \\ A(n,k,m) & = & \frac{1}{m} A(n-1,k,m-1) P_k, \;\; m = 1, \ldots, M, \end{array} \tag{18}$$

with starting conditions:

$$\begin{array}{rcll} A(0,k,0) & = & \pi_0, & k \geq 1, \\ A(0,k,m) & = & 0, & k \geq 0, m = 1, \ldots, M, \\ A(n,0,m) & = & 0, & n \geq 1, m = 0, \ldots, M. \end{array}$$

We have not found a way to normalize the terms $A(n, k, m)$ along the way as in (13) for the base algorithm. Instead, we normalize once at the end and then multiply with Poisson probabilities:

$$\pi^c(T) = \sum_{n=0}^{N} \frac{(\lambda T)^n}{n!} e^{-\lambda T} \frac{\sum_{m=0}^{M} A(n, K, m)}{\sum_{n=0}^{N} \sum_{m=0}^{M} A(n, K, m)}. \tag{19}$$

3.3 Interval Splitting Algorithm

In interval splitting, the original interval $[0, T]$ is divided up in I intervals $[0, T_1], [T_1, T_2], \dots, [T_{I-1}, T]$, and transient probability vectors are computed for all subintervals subsequently. That is, first $\pi(T_1)$ is computed starting from π_0, then $\pi(T_2)$ is computed starting from $\pi(T_1)$, all the way until $\pi(T)$ is solved. To obtain the intermediate results $\pi(T_1), \pi(T_2), \dots$, uniformization is used (either the base algorithm or the correction algorithm).

The potential benefit of splitting the time interval in multiple parts can be observed from estimating the computational complexity of the uniformization algorithms, which is dominated by the number of matrix-vector multiplications (MVMs) needed (iterative scheme eqn. (12) and eqn. (18), respectively). To compute the number of MVMs, we need to know the number of Poisson jumps. In [van Moorsel and Sanders 1997] it was shown that the truncation errors approximately sum when doing interval splitting, and, hence, the error tolerance should be set to $\frac{\epsilon}{I}$ for all intervals, where ϵ is a pre-specified bound on the truncation error. The number of Poisson jumps in the respective intervals then is $N(\frac{T}{I}, \frac{\epsilon}{I})$, such that:

$$\sum_{n=0}^{N(\frac{T}{I},\frac{\epsilon}{I})} \frac{(\lambda \frac{T}{I})^n}{n!} e^{-\lambda \frac{T}{I}} > 1 - \frac{\epsilon}{I}. \tag{20}$$

It can now be shown that the base algorithm requires the following number of MVMs:

$$MVM_{base} = N(\frac{T}{I}, \frac{\epsilon}{I}) \cdot I \cdot K_i, \tag{21}$$

where $K_i = \lceil \frac{T/I}{\Delta} \rceil$ is the number of discretization steps in the ith interval. The optimum choice with respect to computational requirements is thus to choose I such that the above expression is minimized.[3]

For the correction algorithm with M correction events, it can be shown that the following number of MVMs is required to achieve the desired accuracy:

$$MVM_{corr} = N(\frac{T}{I}, \frac{\epsilon}{I}) \cdot I \cdot Max(K_i, \lceil \frac{N(\frac{T}{I}, \frac{\epsilon}{I})}{M} \rceil) \cdot Min(M, N(\frac{T}{I}, \frac{\epsilon}{I})). \tag{22}$$

Table 2 in Section 4 shows for an example model the trend of the number of MVMs required for different values of M and I.

4 RESULTS

We now apply the algorithms from Section 3 to a simple non-homogeneous Markov model and study their accuracy, speed and memory demand. As an example we use a safety model of a duplex system, that has been analyzed in [Rindos et al. 1995]. All computations are carried out by running Mathematica programs on an Intel 100 MHz PC. Figure 2 shows the Markov process of

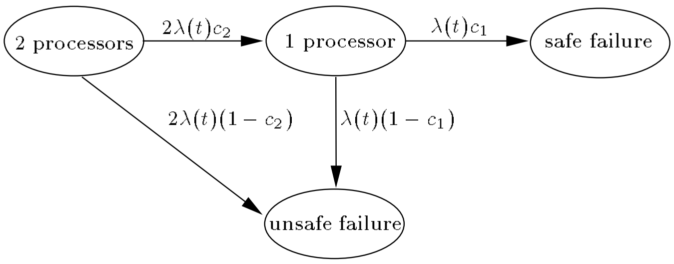

Figure 2: Example

the considered system. There are two processors, both with a time-dependent failure rate $\lambda(t) = \lambda_0 \alpha t^{\alpha-1}$. Failure of each processor may lead to a system crash or a safe shut-down of the one processor. Probability of a safe shut-down is c_2 for the first processor and c_1 for the second. The parameters are chosen to be $\lambda_0 = 0.1$, $\alpha = 2.0, c_1 = c_2 = 0.6$. These parameter values differ from [Rindos et al. 1995] to assure that the obtained results are not too small and sensitive to round-off errors. The generator matrix $Q(t)$ can then be written as

$$Q(t) = \lambda(t) \cdot \begin{pmatrix} -2 & 2c_2 & 0 & 2(1-c_2) \\ 0 & -1 & c_1 & 1-c_1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{23}$$

We use this model to determine the accuracy of the uniformization algorithms because the transient state probability vector $\pi(T)$ can be computed from a CTMC with generator matrix R_T defined as:

$$R_T = \frac{1}{T} \int_{t=0}^{T} \lambda(t)\ dt\ Q. \tag{24}$$

The conversion is possible because $Q(t)$ depends on t only through one parameter $\lambda(t)$ (see [Rindos et al. 1995] for more general conditions).[4] To obtain the results in Table 1 we used SHARPE [Sahner et al. 1996].

Time	Prob(Safe Failure)	Prob(Unsafe Failure)
1	$3.26013009 * 10^{-3}$	0.0746811188
2	$3.91279939 * 10^{-2}$	0.246353744
3	$1.26777444 * 10^{-1}$	0.418398740
4	0.229308899	0.536567717
5	0.303324461	0.599521128

Table 1: Results for safety model, obtained from transformation to a CTMC

[3] The approach taken in [Gokhale et al. 1996] to solve reliability growth models, boils down to interval splitting with $K_i = 1$. This will give accurate results if I is chosen large enough, but may not necessarily be computationally best.

[4] The particular examples of reliability growth models mentioned in [Gokhale et al. 1996] can be solved by using the conversion approach as well.

4.1 Accuracy

In the discretization in (11), P_{k_i} is a probability matrix that must be representative for the whole k_i-th interval. We will take $P_{k_i} = P(s)$ for some $s \in [(k_i - 1)\Delta, k_i\Delta]$. In our experiments we consider five different values for the point s: $s = (k_i - 1)\Delta + Ptime\ \Delta$, for $Ptime =$ 0.0, 0.25, 0.5, 0.75, 1.0.

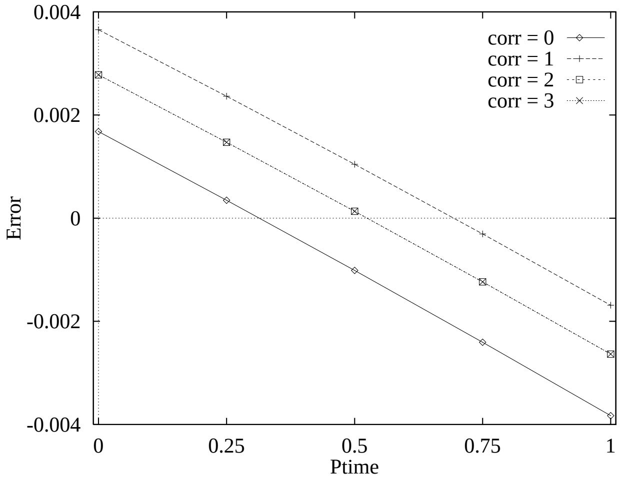

Figure 3: Error for different values of $Ptime$

Figure 3 shows how important the choice of P_{k_i} is for the accuracy. Accuracy of the algorithms is measured by the deviation from the results in Table 1. Throughout this section we present results for Prob(Safe Failure) at time = 5, but the results for Prob(Unsafe Failure) are in principle the same. We do runs for $M = 1, 2, 3, 4$ correction events, and plot a separate curve for each value of M (that is called 'corr' in the plots). Note, that 'corr = 0' refers to results for the base algorithm.

Assuming that the most accurate results are obtained when the number of correction events is highest, Figure 5 shows that the best value of $Ptime$ is close to 0.5. Based on probabilistic arguments, we formulate a heuristic for choosing P_{k_i}. Since Poisson events form a uniform distribution over a time interval, one can take the 'expected' transition rates over the interval by equating P_{k_i} to

$$P_{k_i} = \int_{s_i=(k_i-1)\Delta}^{k_i\Delta} P(s_i) \cdot \frac{1}{\Delta}\, ds_i.$$

If one evaluates P_{k_i} in this way for our example, it gives $P_{k_i} = P((k_i - 0.5)\Delta)$.

When solving integral equations by discretization one expects the error to decrease with the step size Δ. Figure 4 shows that this holds true for the uniformization algorithms as well. Note, that both axes in Figure 4 have a logarithmic scale, plotting the absolute error versus the step size.

Figure 4 also shows the impact of the number of correction events on the error. Including three events for

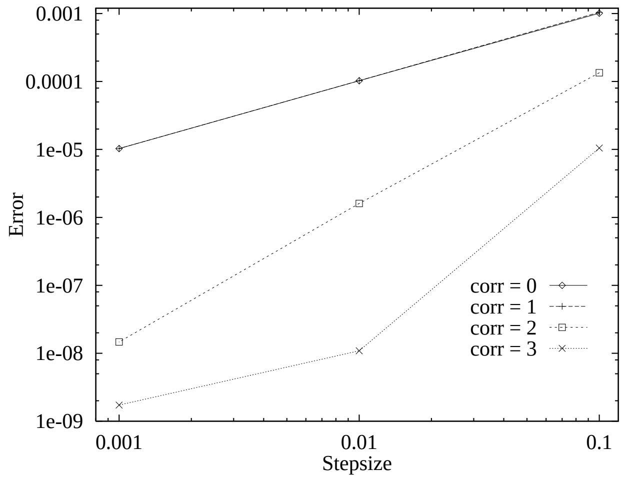

Figure 4: Absolute error for step size 0.1, 0.01 and 0.001

correction for step size 0.1 reduces the error as much as taking two orders of magnitude smaller step size. So it certainly is the cheapest way to increase accuracy when regarding runtime and memory requirements (see below).

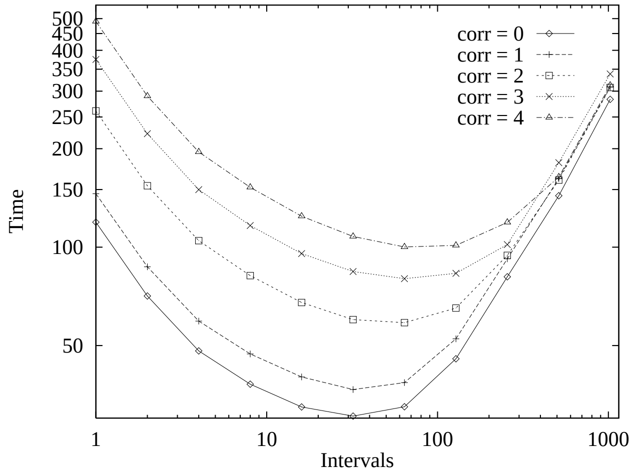

Figure 5: Duration (in seconds) of interval splitting algorithm for 2^i intervals, $i = 0, \ldots, 10$

4.2 Computation Time

We now investigate the computation time of the algorithms. Especially the interval splitting algorithm is of interest, since it aims at improving the efficiency of the algorithms, hopefully without loosing accuracy. In the experiments that follow the error bound ϵ in (20) for the summation of Poisson probabilities has been set to 10^{-8}, small enough so that it does not obscure other error causes. Step size is fixed at $\Delta = 0.01$.

Figure 5 shows the duration of the program runs when splitting up the time period into $2^i, i = 0, \ldots, 10$, intervals. There is an optimum around 32 or 64 intervals, depending on the number of correction events used. The

Interv. I	Poiss. jumps $N(\frac{T}{I}, \frac{\epsilon}{I})$	base algorithm MVMs	K	corr = 1 MVMs	K	corr = 2 MVMs	K	corr = 3 MVMs	K	Poiss. comp. $N(\frac{T}{I}, \frac{\epsilon}{I}) \cdot I$
1	32	16 000	500	16 000	500	32 000	500	48 000	500	32
2	22	11 000	250	11 000	250	22 000	250	33 000	250	44
4	16	8 000	125	8 000	125	16 000	125	24 000	125	64
8	12	6 048	63	6 048	63	12 096	63	18 144	63	96
16	10	5 120	32	5 120	32	10 240	32	15 360	32	160
32	8	4 096	16	4 096	16	8 192	16	12 288	16	256
64	7	3 584	8	3 584	8	7 168	8	10 752	8	448
128	6	3 072	4	4 608	6	6 144	4	9 216	4	768
256	5	2 560	2	6 400	5	7 680	3	7 680	2	1280
512	5	2 560	1	12 800	5	15 360	3	15 360	2	2560
1024	4	4 096	1	16 384	4	16 384	2	24 576	2	4096
2048	4	8 192	1	32 768	4	32 768	2	49 152	2	8192
4096	3	12 288	1	36 864	3	49 152	2	36 864	1	12288

Table 2: Number of Poisson jumps, MVMs, discretization steps and multiplications with Poisson probabilities, for the interval splitting algorithm with different numbers of intervals.

difference in computing time can be quite substantial. For instance, for $M = 4$ correction events the computing time for 64 intervals is reduced 75 percent compared to that of 1 interval, and for the base algorithm the reduction at 32 intervals is about 95 percent.

To better explain the shape of the curves in Figure 5, we use the results in (21) and (22) to approximate the number of MVMs required, since this number is expected to dominate the computational effort. Table 2 gives the number of MVMs for the base algorithm and correction algorithm, with the number of intervals increasing from 1 to 4096. It can be seen that the shape of the curves in Figure 5 indeed seems to find its explanation in the trends of Table 2 for the number of MVMs. Note, that generating a table like Table 2 can be done without running either of the algorithms, and can be a helpful tool is determining which uniformization algorithm to apply.

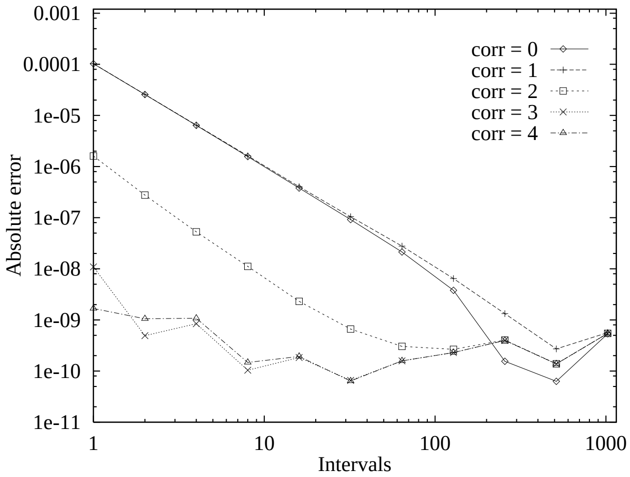

Figure 6: Absolute error for the interval splitting algorithm with 2^i intervals, $i = 0, \ldots, 10$

Interval splitting helps speeding up the computation (up to some threshold), but it is not clear beforehand whether it also gives satisfactory accuracy. Figure 6 therefore shows the absolute error versus the number of intervals. We see that accuracy actually improves when more intervals are used. We also see that the base algorithm and the correction algorithm with one or two correction events do not always reach the accuracy level of $\epsilon = 10^{-8}$ we specified. With three or four correction events, accuracy is always within the desired bounds.

4.3 Memory Requirements

We want to spend a few words on the memory requirements of the respective algorithms. The memory requirements of the algorithms are dominated by the number and size of the vectors $A(n, k)$ in (12) and $A(n, k, m)$ in (18), which can be measured as:

$$\text{size}(A) = N\left(\frac{T}{I}, \frac{\epsilon}{I}\right) \cdot \text{length}(\pi) \cdot Max(M, 1).$$

So, the memory requirement of the correction algorithm linearly increases in the number of correction events, and from a memory perspective the base algorithm is thus preferred. In addition, the more intervals are used, the smaller $N\left(\frac{T}{I}, \frac{\epsilon}{I}\right)$ gets, so interval splitting will also be helpful to reduce memory demands.

4.4 User Guidelines

We conclude this section with guidelines for the application of the uniformization algorithms. First of all, interval splitting is favorable for memory use and accuracy, and up to some point for speed as well. The base algorithm outperforms the correction algorithm with respect to memory and speed, but lacks in accuracy. Only when interval splitting is applied with enough intervals, the accuracy of the base algorithm is satisfactory. Interval splitting should thus be applied up to some number of intervals, and it depends on the achieved accuracy whether the base algorithm or the correction algorithm should be applied to the subintervals. A sensible approach is to create a table like Table 2 to estimate the optimal setting with respect to speed, and then test the accuracy by comparing this setting with a setting with higher number of correction events.

In addition to the above trade-offs, decisions must be made regarding the time $Ptime$ for which to compute the matrix P and the discretization step size Δ. To determine a good value of Δ, we have no better recipe than to try out different values, the smaller the better. For $Ptime$, however, we outlined a heuristic procedure based on stochastic argument that gave the optimal value for the presented example.

5 CONCLUSION

We have presented three uniformization-based algorithms for the numerical solution of transient state probabilities in non-homogeneous continuous-time Markov processes. To study performance and accuracy we applied the algorithms to a duplex safety model. Compared to the homogeneous case, the algorithms for non-homogeneous processes possess more intricate error behavior because of the time-discretization. Nevertheless, for our example, a high accuracy is obtained when parameters of the algorithm are tuned correctly.

Important in the application of the algorithms is the choice of the representative matrix for each discretization interval, and we provided a heuristic procedure that gave the optimal matrix for the example. A trade-off was also found between time and accuracy. As a rule of thumb, as long as the computation does not turn slow, interval splitting should be applied with as many subintervals as desired, with the base algorithm as underlying method. In cases when this is not accurate enough, the correction algorithm can provide good accuracy irrespective of the number of subintervals in interval splitting.

The presented uniformization algorithms form the first step in evaluating the potential of uniformization for numerical solution of non-homogeneous Markov processes. With the gained understanding of how to apply uniformization best, the next step is to compare it to rival methods (typically differential equation solvers [Heindl and German 1997, Stoer and Bulirsch 1980]), with regard to speed, accuracy and memory use. This comparison needs not be limited to non-homogeneous Markov chains, but may also include some types of non-Markovian models, using the hazard rate of delay distributions as time-dependent transition rate [Nicola et al. 1992].

REFERENCES

[Antsaklis and Michel 1997] Antsaklis, P. J. and Michel, A. N. (1997). *Linear Systems*. Series in Electrical and Computer Engineering, Control Theory. McGraw-Hill, New York.

[Çinlar 1975] Çinlar, E. (1975). *Introduction to Stochastic Processes*. Prentice-Hall, Englewood Cliffs, New Jersey.

[Gokhale et al. 1996] Gokhale, S. S., Philip, T., and Marinos, P. N. (1996). A non-homogeneous Markov software reliability model with imperfect repair. In *2nd Annual IEEE International Computer Performance & Dependability Symposium*, pages 262–270, Urbana-Champaign, IL. IEEE, IEEE Computer Press.

[Grassmann 1991] Grassmann, W. K. (1991). Finding transient solutions in Markovian event systems through randomization. In Stewart, W., editor, *Numerical Solution of Markov Chains*, chapter 18, pages 357–371. Marcel Dekker, New York.

[Heindl and German 1997] Heindl, A. and German, R. (1997). A fourth-order algorithm with automatic stepsize control for the transient analysis of DSPNs. In *7th International Workshop on Petri Nets and Performance Models*, pages 60–69, Saint Malo, France. IEEE, IEEE Computer Society Press.

[Jensen 1953] Jensen, A. (1953). Markoff chains as an aid in the study of Markoff processes. *Skandinavisk Aktuarietidskrift*, 36:87–91.

[Lewis and Shedler 1979] Lewis, P. A. W. and Shedler, G. S. (1979). Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413.

[Nicola et al. 1992] Nicola, V. F., Heidelberger, P., and Shahabuddin, P. (1992). Uniformization and exponential transformation: Techniques for fast simulation of highly-dependable non-Markovian systems. In *22th International Symposium on Fault-Tolerant Computing*, pages 130–139. IEEE, IEEE Computer Society Press.

[Platis et al. 1997] Platis, A., Limnios, N., and Du, M. L. (1997). Asymptotic availability of systems modeled by cyclic non-homogeneous Markov chains. In *Annual Reliability and Maintainability Symposium*, pages 293–297, Philadelphia, PA. IEEE.

[Rindos et al. 1995] Rindos, A., Woolet, S., Viniotis, I., and Trivedi, K. (1995). Exact methods for the transient analysis of nonhomogeneous continuous time Markov chains. In Stewart, W. J., editor, *Computations with Markov Chains*, chapter 8, pages 121–133. Kluwer Academic Publishers, Boston.

[Sahner et al. 1996] Sahner, R. A., Trivedi, K. S., and Puliafito, A. (1996). *Performance and Reliability Analysis of Computer Systems, An Example-Based Approach Using the SHARPE Software Package*. Kluwer, Boston, MA.

[Stoer and Bulirsch 1980] Stoer, J. and Bulirsch, R. (1980). *Introduction to Numerical Analysis*. Springer Verlag, New York.

[van Dijk 1992] van Dijk, N. M. (1992). Uniformization for nonhomogeneous Markov chains. *Operations Research Letters*, 12:283–291.

[van Moorsel 1993] van Moorsel, A. P. A. (1993). *Performability Evaluation Concepts and Techniques*. PhD thesis, University of Twente, The Netherlands.

[van Moorsel and Sanders 1997] van Moorsel, A. P. A. and Sanders, W. H. (1997). Transient solution of Markov models by combining adaptive and standard uniformization. *IEEE Transactions on Reliability*, 46(3):430–440.