Probabilistic Evaluation for the Analytical Solution of Large Markov Models: Algorithms and Tool Support

Aad P.A. van Moorsel
University of Illinois at Urbana-Champaign
Center for Reliable and High-Performance Computing
1308 W. Main St. Urbana, IL 61801-228
moorsel@crhc.uiuc.edu

Boudewijn R. Haverkort
Rheinisch-Westfälische Technische Hochschule Aachen
LFG Informatik IV—Distributed Systems
Ahornstrasse 55, D-52056 Aachen, Germany
haverkort@informatik.rwth-aachen.de

September 6, 1995
Abstract

Over the past decade constantly increasing computer power has made analytic solution of Markovian performance and dependability models more attractive. However, its application for practical systems still needs improvement since detailed and realistic models typically result in Markov (reward) models that are too large to completely generate and store in memory. In this paper we discuss this problem of largeness of Markov reward models and propose solutions if transient measures are considered.

We will introduce a novel approach, called probabilistic evaluation, based on the probabilistic verification and validation methods known in the area of communication protocol analysis. In order to apply these ideas to solve for transient reward measures, we develop algorithms that do not rely on \textit{a priori} generation of the whole state space. Instead, only parts of the state space will be considered at a time, and states that are not needed to get accurate results will not be generated.

We will not only deal with acyclic models but allow for general Markov reward models. The algorithms we develop are based on uniformization. For acyclic models we introduce \textit{orthogonal uniformization} and for the analysis of non-acyclic models we introduce \textit{partial uniformization}.

For the utility of the above new uniformization methods the order in which the states are generated from the high-level description will turn out to be of great importance. We propose the use of a simple heuristic method for state selection, which, after comparison with two other methods, appears to be relatively cheap and close to optimal when considering transient dependability measures.

The uniformization and state space exploration methods we develop are implemented in a prototype probabilistic evaluation tool, which is especially useful for experimenting with new algorithms.
1 Introduction

Markov modeling is a potentially very powerful and generic approach for evaluating the performance and dependability of many different systems, such as computer systems, communication networks, manufacturing systems, etc. Many performance and dependability measures of interest can be specified by reward variables on top of the Markov model, and largely independent of the actually constructed model standard solution methods can be applied to analytically solve these reward measures. As a consequence of their general applicability, Markov reward models are used in several software packages that have been developed for the analytic solution of performance and dependability models, e.g., [1, 19]. In these tools, high-level modeling constructs such as stochastic Petri-nets are used to specify complicated models, and an automatic mapping is carried out at a Markov reward model. Subsequently this Markov model is solved by standard solution algorithms, appropriately selected for the precise measure of interest.

One of the major aspects that hampers the use of Markov reward models for the analytic solution of performance and dependability measures is the size of the model. With the use of the mentioned high-level modeling tools it is a simple task to specify complicated and large underlying Markov models, and although typically models of up to hundreds of thousand states can conveniently be solved on modern-day work-stations, often even larger models are needed to represent the necessary detail of the system or to represent systems with realistic dimensions. As a result, the underlying state space and generator matrix of the model often are too big to generate and store in memory. In this paper we discuss this problem of largeness, and propose solutions for it when the desired performance and dependability measures are transient measures (i.e., measures defined for finite points in time).

We base our approach in this paper on what is known in the area of communication protocol design as probabilistic verification [14] and probabilistic validation [4]. In probabilistic verification a communication protocol that is specified in the form of a finite state machine is verified for correctness. To cope with the typically large size of the state space, verification is carried out by selecting a subset of the possible state trajectories. This leads to a lower bound for the correctness of the protocol, i.e., with at least a certain probability a protocol will behave correct. In probabilistic validation this approach has been taken over, but now for protocols with a temporal specification. In probabilistic validation a lower bound for the system reliability (i.e., the probability the system is in a failure state at the end of the mission time) is determined by considering only a subset of state trajectories. Two restrictions apply to the proposed probabilistic validation approach in [4]: only acyclic Markov models can analytically be solved, and the considered mission time is exponentially distributed.

In this paper we build forward on the probabilistic verification/validation approaches. We will see that the approach of probabilistic validation can be generalized in a straightforward manner to take other measures besides reliability into account. We therefore propose to speak
of probabilistic evaluation, as we do not deal with pure protocol verification or validation. We will study the probabilistic evaluation of transient measures for which the time instant of interest is deterministic instead of exponentially distributed (as was considered in [4]). A deterministic time instant appears usually more naturally in transient performance evaluation, however, the analysis is typically more complicated than for exponential mission times.

To perform the analysis for deterministic time points we develop algorithms that are based on uniformization, a method considered by different authors to be the method of choice for transient analysis [8]. We will enhance it in different ways to be able to evaluate models with a very large state space. For acyclic models orthogonal uniformization (OU) will be introduced. In OU the memory occupancy is no longer determined by the size of the state space, which makes it possible to solve very large and even infinitely large acyclic Markov models. To deal with non-acyclic models a method called partial uniformization (PU) will be introduced. PU iteratively traverses through the state space, using the fact that for transient measures bounds on the induced error can be straightforwardly obtained along the way. It will be seen that PU is particularly promising as an approach to dealing with large state spaces in generic Markov reward models.

The uniformization methods we introduce are based in part on the idea that not the complete state space has to be available in memory at all time. This philosophy has been taken in other work as well, as we mentioned above. Especially noteworthy in this regard is the work in [18], since it addresses the computation of transient measures in (semi-)Markov models. In [18] the model is evaluated by considering, one by one, possible evaluations in time for the model. No generation of the state space takes place in this case, and memory usage is traded for computation time. In our approach we will not trade off computation and memory use, but will temporarily store intermediate results when necessary.

The utility of both OU and PU as probabilistic evaluation approaches is influenced considerably by the order in which the state space is traversed. Therefore, we consider different state space exploration approaches in this paper. Besides known methods we introduce a simple, and therefore cheap, heuristic approach which intuitively seems most attractive when dealing with transient measures. Comparing the different state space exploration approaches will show that in many cases the heuristic approach explores the state space in a satisfactory order without much overhead (see also [11]).

The combination of uniformization and state space exploration forms a promising approach for probabilistic evaluation, but its practical applicability is very much determined by implementation decisions and by the actual model under consideration. It is therefore essential to experiment with prototype implementations of the algorithms, for realistic systems. We therefore developed the probabilistic evaluation tool (PET), and will discuss the use of the tool by applying the proposed methods to example models.

The organization of the paper is as follows. First we extend in Section 2 the probabilistic validation approach in [4] to general transient performance and dependability measures and
to deterministic mission times. Then we introduce in Section 3 orthogonal uniformization to evaluate acyclic Markov models, and introduce in Section 4 partial uniformization for general, i.e., non-acyclic, Markov models. State space exploration approaches are then discussed in Section 5. Finally, Section 6 discusses the design of the PET software, while we report on experiments using PET in Section 7.

2 Probabilistic Evaluation for Deterministic Mission Times

In this section we formally define probabilistic evaluation. To set the stage for the introduction of new probabilistic evaluation algorithms, we give in Section 2.2 the basic equation for the transient solution of Markov chains at deterministic points in time, and review in Section 2.3 the standard form of uniformization which underlies the newly developed algorithms in this paper. It will also be shown how standard uniformization can be viewed as a probabilistic evaluation technique itself.

2.1 Probabilistic Evaluation

Probabilistic protocol validation in [4] aims at showing that the probability that a pre-defined set of states (the failure states) is reached before time \( t \) is less than \( \epsilon \). So, if \( S \) is the model state space, \( F \subseteq S \) is the set of failure states and \( \pi_i(t), i \in S \), is the probability the model resides in state \( i \in S \) at moment \( t \), then the goal of probabilistic validation is to show that \( \sum_{i \in F} \pi_i(t) < \epsilon \) (or, in other words, that \( \sum_{i \in S \setminus F} \pi_i(t) \geq 1 - \epsilon \)). In this way, the reliability of a system (which is commonly defined as the probability the system is up over the whole interval \([0, t]\)) can be bounded.

In probabilistic evaluation we generalize this notion, as we do not concentrate on protocol validation or verification but on system performance evaluation. The measure we will consider is the expected transient reward \( \mathbf{E} R(t) \), which is defined as

\[
\mathbf{E} R(t) = \sum_{i \in S} \pi_i(t) r_i,
\]

where \( r_i \) denotes the reward collected when the system is in state \( i \in S \). Probabilistic evaluation then aims at determining a value \( \phi \) such that for given accuracy \( \epsilon \),

\[
\phi \leq \mathbf{E} R(t) \leq \phi + \epsilon.
\]

We will say that the tuple \((\phi, \epsilon)\) probabilistically evaluates \( \mathbf{E} R(t) \). With the above definition of probabilistic evaluation, more intricate measures than the reliability can be investigated, such as the buffer occupancy of a switch in a communication network, or the performance/dependability of a fault-tolerant computer system as in [21]. Note that probabilistic validation is a special case of probabilistic evaluation. In particular, choose \( r_i = 1 \) if \( i \in S - F \).
and \( r_i = 0 \) if \( i \in F \). A protocol is then correct under probabilistic validation if one can find \( \phi \) such that \( \phi \geq 1 - \epsilon \), and hence \( 1 - \epsilon \leq \phi \leq ER(t) \leq 1 \).

Besides this straightforward generalization of probabilistic validation to generic measures we want to deal with two other restrictions Florin et al. [4] introduce in their development of a solution approach for probabilistic validation. First, only acyclic Markov models are addressed, a major limitation which we will examine in Section 4. Secondly, the instant of time \( t \) is not taken to be deterministic, but exponentially distributed with average \( \mu \). This assumption simplifies the analysis considerably and makes it possible to analyze very large models. However, often one is interested in a deterministic point in time, and therefore we will take that as the focus in this paper. Before discussing probabilistic evaluation methods, let us first give the general equation for transient solutions when \( t \) is deterministic, and then give the basic uniformization solution algorithm.

### 2.2 Transient Solutions

We consider the continuous-time Markov chain \( Y = \{Y(s); s \geq 0\} \), defined on the countable, possibly infinite state space \( S \). So, transitions between states \( i \in S \) and \( j \in S \) are exponentially distributed, having rate \( q(i, j) \), and the infinitesimal generator is given by \( Q = (q(i, j)) \), \( i, j \in S \), with \( q(i, i) = -\sum_{j \in S} q(i, j) \). Then \( \pi(t) \), the state occupancy distribution at instant of time \( t \), is the solution of Kolmogorov’s system of differential equations [3]:

\[
\frac{d\pi(t)}{dt} = \pi(t)Q, \quad \text{given } \pi(0).
\]  

The number of differential equations in this solution equals the state space size \(|S|\). The solution of (3) is given by \( \pi(t) = \pi(0)e^{Qt} \), but direct computation of this matrix exponential is not feasible because of severe round-off errors that usually will occur [15]. Therefore, other algorithms have been developed of which uniformization is currently the most popular iterative algorithm.

### 2.3 Standard Uniformization

The basic algorithm behind all the probabilistic evaluation algorithms that follow is standard uniformization (SU). SU is based on the more general concept of uniformization, e.g., [12, 20], and is also known as Jensen’s method [8] or randomization [9]. To introduce SU, define

\[
P = I + \frac{1}{\lambda}Q.
\]  

Then \( \pi(t) = \pi(0)e^{Qt} = \pi(0)e^{\lambda tp-I}p \), i.e.,

\[
\pi(t) = \pi(0)\sum_{n=0}^{\infty} e^{-\lambda t}\left(\frac{\lambda t}{n!}\right)^n P^n.
\]
In this equation
\[ PP(n) = e^{-\lambda t} \frac{\lambda^t}{n!}, \quad n = 0, 1, \ldots, \] (6)
are Poisson probabilities, i.e., \( PP(n) \) is the probability of \( n \) events occurring in \([0, t]\) in a Poisson process with rate \( \lambda \) (\( \lambda \) denotes “is defined as”). If now \( \lambda \) is chosen such that \( \lambda \geq \sup_{i \in S} \{-q(i, i)\} \), then the entries in \( P \) are all between 0 and 1, while the rows of \( P \) sum to 1. In other words, \( P \) is a stochastic matrix and describes a discrete-time Markov chain. Note that the value of \( \lambda \) (the uniformization rate) can be derived from inspection of the \( Q \)-matrix, provided the complete matrix is available.

Uniformization allows for an iterative solution algorithm in which no matrix-matrix multiplications take place, and thus no matrix fill-in occurs. Instead of directly computing (5) one considers the following sum of vectors:
\[ \pi(t) = \sum_{n=0}^{\infty} PP(n)\pi(n), \] (7)
where \( \pi(n) \), being the state probability distribution vector after \( n \) epochs in the uniformized process with transition matrix \( P \), is derived recursively as
\[ \pi(n = 0) = \pi(t = 0) \quad \text{and} \quad \pi(n) = \pi(n - 1)P, \quad n = 1, 2, \ldots. \] (8)
Clearly, the infinite summation in (7) has to be truncated, say after \( N \) iterations or epochs in the discrete-time Markov chain. Then the actually computed state occupancy probability vector \( \pi^c(t) \) in SU is (the superscript \( c \) is used in this paper to distinguish the “computed” result from the actual result)
\[ \pi^c(t) = \sum_{n=0}^{N} PP(n)\pi(n). \] (9)
So, the computed result \( ER^c(t) \) for the expected reward at moment \( t \) is then
\[ ER^c(t) = \sum_{i \in S} \pi^c_i(t)r_i. \] (10)
Finally, we note that the Poisson probabilities \( PP(n), n = 0, 1, \ldots, N \), can be computed in a stable manner as formalized by Fox and Glynn [5], see also [7].

We will now illustrate how the computational SU scheme can be formulated as a probabilistic evaluation method. To this end, we discuss SU for the example Markov chain given in Figure 1. The Markov model represents an extended machine repairman model consisting of two components. Components can fail with rate \( f \), and failures are covered with a coverage probability \( c \). A covered failure is called a soft failure, an uncovered failure is called a hard failure. The state \((i, j, k)\) in the model represents the situation in which there are \( i \) components “up,” \( j \) soft failed components, and \( k \) hard failed components. Soft and hard failed components have different repair rates, \( s \) and \( r \) respectively. Larger versions of this model have been used in [2, 17] as typical examples of dependability models.
For the Markov model in Figure 1 the iterative scheme of SU (given by Equation (8)) is illustrated in Figure 2. In this figure, the \((i,n)\)-th circle in the \(i\)-th row and the \(n\)-th column denotes the element \(\pi_{i}(n)\). So, the columns \(k, k = 0, 1, \ldots N\), represent the elements computed at iteration \(k\), according to (8). In Figure 2 iteration \(n\) has just been completed while in the next iteration \(\pi(n+1)\) will be computed from \(\pi(n)\). A black circle denotes that the element \(\pi_{i}(n)\) is currently stored in memory. A grey circle \((i,n)\) denotes that the element \(\pi_{i}(n)\) has been computed earlier in the uniformization procedure, but is no longer stored in memory. A white circle denotes the elements that have to be computed yet, while an asterisk in a white circle denotes the elements that will be computed in the next iteration. So, the elements \(\pi_{i}(n)\) are kept in memory when \(\pi_{i}(n+1)\) is computed. Furthermore, the arrows from \(\pi_{i}(k)\) to \(\pi_{j}(k+1), k = 0, 1, \ldots, N - 1\), denote the matrix elements \(P(i,j)\) which are greater than zero. To compute \(\pi_{j}(k+1)\) only these elements \(\pi_{i}(k)\) for which \(P(i,j) > 0\) have to be stored in memory, an observation which will turn out to be important in what follows.

**Probabilistic evaluation by SU.** SU can be formulated as a probabilistic evaluation method for the measure \(E_{R}(t)\) in the following way. From (7) we have that

\[
E_{R}(t) = \sum_{i \in S} \sum_{n=0}^{\infty} r_{i} PP(n)\pi_{i}(n),
\]

but we see from (9) that we actually compute

\[
E_{R'}(t) = \sum_{i \in S} \sum_{n=0}^{N} PP(n)\pi_{i}(n).
\]

Now we make the following observation:

\[
\sum_{i \in S} \sum_{n=0}^{\infty} PP(n)\pi_{i}(n) = 1.
\]

So, the probability mass not covered in the computation of \(E_{R'}(t)\) equals

\[
\sum_{i \in S} \sum_{n=N+1}^{\infty} PP(n)\pi_{i}(n) = 1 - \sum_{i \in S} \sum_{n=0}^{N} PP(n)\pi_{i}(n) = 1 - \sum_{i \in S} \pi_{i}^{\ast}(t).
\]

Taking \(r_{max} = \max_{i \in S}\{r_{i}\}\), the bound \(\epsilon\) for the error introduced by considering \(E_{R'}(t)\) instead of \(E_{R}(t)\) is

\[
\epsilon = r_{max}(1 - \sum_{i \in S} \pi_{i}^{\ast}(t)).
\]

Take now \(\phi = E_{R'}(t)\), then,

\[
\phi = E_{R'}(t) \leq E_{R}(t) \leq E_{R'}(t) + \epsilon = \phi + \epsilon,
\]

and, hence, the pair \((\phi, \epsilon)\) probabilistically evaluates \(E_{R}(t)\).
Note that in (15) we implicitly assume all rewards \( r_i, i \in S \), to be greater or equal to 0. This does, however, not lead to loss of generality since it is always possible (for the expected reward) to increase the rewards to positive values and later compensate for this. Furthermore, note that because

\[
1 - \sum_{i \in S} \pi_i^c(t) = 1 - \sum_{i \in S} \sum_{n=0}^{N} PP(n)\pi_i(n) = \sum_{n=N+1}^{\infty} PP(n) = \epsilon_N,
\]

it is possible to reach any desired accuracy \( \epsilon \) in SU by appropriately increasing the number of iterations \( N \) until \( \epsilon_N \) is small enough.

Different authors (see [8]) have called SU the “method of choice” when computing transient measures. Two major problems might decrease the applicability of SU however: (i) the time it takes to perform the computation, which is determined by the necessary number of iterations \( N \), and (ii) the difficulty to handle a very large state space, because of the amount of used memory. The time problem is extensively discussed by many authors, see for instance [2, 17]. The size problem is the subject of this paper.

3 Orthogonal Uniformization for Acyclic Markov Models

In this section we consider the probabilistic evaluation of large acyclic Markov models, for which SU is no longer feasible because the complete state space is too large to be completely generated. We will present an enhancement of SU, to be called orthogonal uniformization (OU), which shifts the memory bottleneck from the size of the state space to the number of iterations \( N \) in uniformization. In some cases OU can solve for very large and even infinite Markov models, and we will see in Section 3.2 that its efficiency can even be improved by considering hybrid combinations of SU and OU.

3.1 Orthogonal Uniformization

To introduce OU, define \( A_i \), the set of so-called ascenders of a state \( i \in S \), as

\[
A_i = \{ j \in S; p(j,i) > 0 \}.
\]

From Equation (8) it follows that, for \( n \geq 1 \),

\[
\pi_i(n) = \sum_{j \in S} \pi_j(n-1)p(j,i) = \sum_{j \in A_i} \pi_j(n-1)p(j,i).
\]

In other words, to compute \( \pi_i(n) \) it is not necessary to have knowledge about all the state occupancy probabilities at step \( n-1 \), but just about the set of ascenders of \( i \). Now, for an acyclic model one can order the states in such a way that all ascenders of a state \( i \in S \) are listed in the state space before the state \( i \). In other words, the state space can be labeled such that for all states \( i \in S \) it holds that if \( j \in A_i \), then \( j < i \). This leads to the basic idea behind
OU: traverse through the state space, which is ordered in the above described way, and keep only information about earlier visited states if they are still ascender of not yet visited states.

Let us describe OU by looking at the simple Markov model in Figure 1, however, without repairs (i.e., \( r = s = 0 \)). The model is then acyclic, with strictly positive elements in the matrix \( P \), as illustrated by the arcs in Figure 3. In Figure 3 one sees that instead of the column-wise computational scheme in SU, OU computes the probabilities \( \pi_i(n) \) in an order orthogonal to this, namely row after row. In Figure 3, row 3, which corresponds to state \( i = (1,0,1) \), has been computed in the last iteration. To compute row 4, corresponding to state \( i = (0,2,0) \), row 2 and 3 have to be kept in memory because \( A_i = \{(1,1,0),(1,0,1)\} \).

However, elements \( \pi_j(n) \) for state \( j = (2,0,0) \) do not have to be kept in memory because this state is no ascender of any of the not yet considered states. Then, after state \( (0,2,0) \), the state \( (0,1,1) \) will be considered, followed by state \( (0,0,2) \), keeping only those intermediate results in memory that are necessary for future computation. In this way the whole state space can be considered.

**Probabilistic evaluation by OU.** In probabilistic evaluation terms, OU gives the tuple \((\phi, \epsilon)\) as follows. Let \( S_1 \subset S \) be the subset of states for which \( \pi_i(t) \) is computed (in Figure 3, \( S_1 = \{(2,0,0),(1,1,0),(1,0,1)\} \)). Then, similar as in (15),

\[
\phi = \sum_{i \in S_1} \pi_i(t)r_i, \quad \text{and} \quad \epsilon = \max(1 - \sum_{i \in S_1} \pi_i(t)).
\]

As a consequence, if \( \epsilon \) is determined to be small enough for the subset \( S_1 \), no states outside the set \( S_1 \) have to be considered anymore. OU therefore may be used very efficiently for probabilistic validation as it automatically detects whether the considered state space is large enough for accurate results.

To reach the desired accuracy as fast as possible it is of great importance in which order the states are traversed. Optimally, the probability mass should be concentrated in the first states to be visited. Therefore, we study in Section 5 different state space exploration methods. Another important aspect in the application of OU is that a uniformization rate \( \lambda \) has to be chosen without a completely generated generator matrix. In the tool implementation to be discussed in Section 6 the user has to give a value for \( \lambda \) as input before starting the OU algorithm.

**Memory usage.** In SU, for every state \( j \in S \) the probability \( \pi_j(n) \) has to be kept in memory at epoch \( n \). In OU, the computer memory occupancy is determined by the number of states \( j \in S \) that are ascender of some other state yet to be considered, and by the number of epochs \( N \) necessary to reach the desired accuracy. In other words, the memory bottleneck shifts from the state space to the number of epochs and the number of states necessary for later computation. One sees that it depends on the specific structure of the model how many states have to be kept in memory. Moreover, the amount of memory use depends on the
specific order in which the state space is explored. For instance, if state \((1,0,1)\) would have been considered after state \((0,2,0)\) the element \(\pi_{(2,0,0)}(n)\) in the first row in Figure 3 would have to be kept in storage longer.

We thus see that OU potentially can solve for large Markov models, but that the specific model under consideration determines to a large extend its usability. An instance in which OU has been proven to be very effective is in the computation of transient probabilities of a pure birth processes [17]. Clearly, a birth process is a very specific process, and in order to further enlarge the applicability of uniformization for large acyclic Markov models, we introduce in subsection 3.2 hybrid combinations of SU and OU. The hybrid OU/SU algorithm can be more efficient than OU, and potentially can solve for models that are “partly acyclic”, as will become clear in the following section.

### 3.2 Hybrid Forms of Standard and Orthogonal Uniformization

Hybrid forms of SU and OU allow for schemes that sometimes perform uniformization “state after state” (as in OU) and sometimes “epoch after epoch” (as in SU). It can deal with models that are partly acyclic by performing SU on a (possibly non-acyclic) subset of the state space, storing only the yet to be used probabilities \(\pi_i(k)\) in memory. Iteratively, new subsets are considered, as was done in OU for individual states.

This procedure is best illustrated by an example, see Figure 4. This time, repair is only possible when soft failures occur, i.e., in the states \((1,1,0),(0,2,0)\) and \((0,1,1)\). That makes that the state space can be divided in subsets which, considered as lumped states, form an acyclic Markov chain. In Figure 4 the subsets have been chosen to be \(S_1 = \{(2,0,0),(1,1,0),(0,2,0)\}\), \(S_2 = \{(1,0,0),(0,1,1)\}\) and \(S_3 = \{(0,0,2)\}\). Hybrid SU/OU then goes as follows. First, SU is carried out over the subset \(S_1\). When carrying out SU all elements \(\pi_i(n)\) are stored for states \(i\) which are ascender of states not belonging to \(S_1\) (i.e., states \((2,0,0)\) and \((1,1,0)\)). Using these stored probabilities, SU is then carried out over the subset \(S_2\), etc. In Figure 4 the algorithm is at the iteration where SU is carried out over subset \(S_2\), just having completed the computation for epoch \(n\). The black circles denote the elements needed for future computation, and are thus stored in memory.

The optimal use of this hybrid form of uniformization depends on the structure of the Markov model. Often, there exist different ways of choosing the subsets \(S_i\), and depending on this choice more or less memory will be used.

We like to note that in the case of a fully acyclic model, hybrid SU/OU can also be applied, and can even be more efficient in memory use than OU. Consider for example Figure 3 when the subsets are \(S_1 = \{(2,0,0),(1,1,0),(1,0,1),(0,2,0),(0,1,1)\}\) and \(S_2 = \{(0,0,2)\}\). If SU is carried out on the subset \(S_1\) only one row has to be stored, namely the row corresponding to state \((1,0,1)\) as only this state is ascender of a state in \(S_2\).

When we deal with a model that is non-acyclic, we can still apply OU, however, this will
then yield approximate results. What basically happens is that probability mass that “flows back” to states already handled is simply lost. In the computational procedures as we have implemented them in PET (see Section 6) only those rows of probabilities are stored that have outgoing rates to still unexplored new states. Depending on the actual model characteristics and parameters, applying OU to non-acyclic models can yield quite accurate results, however, the obtained accuracy can not easily be determined.

Clearly, for the very small examples considered in the figures OU and hybrid SU/OU are not necessary. However, for large and even infinitely large models the method might solve models that cannot be solved otherwise. Interesting is that in principle it is always possible to convert a non-acyclic Markov model into an acyclic Markov model, with an infinite number of states (basically, information is included in the state how many times the state has been visited). Following this approach OU and hybrid OU/SU can be used to probabilistically evaluate non-acyclic models. In practice, however, this idea seems not generally applicable because of the increased memory usage. We therefore now turn our attention to more practical, and more intricate, methods for probabilistic evaluation of non-acyclic models.

4 Partial Uniformization for Non-Ayclic Models

To probabilistically evaluate non-acyclic Markov models we discuss two different methods in this section. In Section 4.1 dynamic uniformization (DU), known from [8], is discussed, and in Section 4.2 the newly developed partial uniformization technique (PU) is introduced. We will see in Section 4.3 that applying PU in an iterative fashion is a promising approach to solve large non-acyclic Markov models.

4.1 Dynamic Uniformization

Dynamic uniformization has been introduced by Grassmann [8] to deal with models that have an infinite state space. The basic idea is not to generate the state space of the Markov model beforehand, but only introduce new states in the Markov chain when the states come up in the uniformization algorithm. To make this more precise, introduce the set of active states $A(n)$ at epoch $n$ of the (standard) uniformization algorithm, as follows:

$$A(n) = \{i \in S|\pi_i(n) > 0\}.$$  \hfill (21)

Then define the transition matrix $P(n)$ as

$$P(n) = I + \frac{Q(n)}{\lambda},$$  \hfill (22)

where $Q(n)$ equals $Q$, however restricted to rows corresponding to the active states $A(n)$. Then uniformization is applied with as only difference from SU (Equation (8)) that $\pi(n) = \pi(n-1)P(n-1)$, $n = 1, 2, \ldots, N$, i.e., $P$ varies with $n$.  

10
As a consequence, if the number of iterations necessary in SU equals $N$, DU only generates these states that are reached in $N$ iterations in the uniformization scheme. This is illustrated in Figure 5 for the example model. The circles with the cross-bar pattern denote elements $\pi_i(k)$ that do not show up at iteration $k$ in DU. Under certain restrictions (the number of active states should always remain finite), DU can solve with any desired accuracy for infinite state space models. Note, that with regard to probabilistic validation the tuple $(\phi, \epsilon)$ is identical to that in Section 2.3 for SU.

Unfortunately, in general DU still might lead to rapid growth of the generated state space. We will discuss in Section 7 examples in which DU can be helpful by itself, but for general models a more explicit form of state space truncation is inevitable. This will be achieved by partial uniformization.

### 4.2 Partial Uniformization

If the state space of a Markov model is truncated, a very convenient way to bound the error introduced by the truncation is available when transient measures are considered. In PU we use this fact to obtain the tuple $(\phi, \epsilon)$ which probabilistically evaluates the model under consideration.

For PU, introduce the truncated state space $S' \subset S$, and define a new (absorbing) state $a$. Delete all states $i \in S - S'$, as well as the transitions among these states. Add, however, the transition $i \rightarrow a$ with intensity $q(i, a)$, such that, for all $i \in S'$,

$$q(i, a) = \sum_{j \in S - S'} q(i, j).$$

(23)

Let $Y'$ denote the resulting Markov process defined on the state space $S'$, and let $\pi'_i(t)$ denote the transient occupancy probability distribution. It then can easily be shown that the probability mass $\pi'_a(t)$ of the absorbing state gives a bound for the induced truncation error as follows:

$$\pi'_a(t) \leq \pi_i(t) \leq \pi'_i(t) + \pi'_a(t),$$

(24)

and

$$\sum_{i \in S'} \pi'_i(t) \leq \sum_{i \in S} \pi_i(t) \leq \sum_{i \in S'} \pi'_i(t) + \pi'_a(t) = 1.$$  

(25)

If one carries out uniformization for the Markov model $Y'$ on the truncated state space $S'$, one obtains results for $\pi'_i(t)$, as well as for $\pi'_a(t)$. We have illustrated this method in Figure 6, which depicts the Markov chain $Y'$. The circles marked 'A' denote the states which are replaced in the new Markov model $Y'$ by a single absorbing state.

**Probabilistic evaluation by PU.** In terms of probabilistic evaluation, $\phi$ is given in PU in the form of results for the truncated model $Y'$. Specifically, if we use the same notation as for the original Markov model with an additional prime symbol when appropriate, we get
that $\phi = \sum_{i \in S} r_i \pi_i^\epsilon(t)$, while $\epsilon = r_{\max}(1 - \sum_{i \in S} \pi_i^\epsilon(t))$. If $\epsilon_N = \sum_{n=N+1}^{\infty} PP(n)$ (as in (17)), it follows from (25) that

$$\epsilon = r_{\max}(\epsilon_N + \pi_a^\epsilon(t)).$$

So, the probability mass in the absorbing state together with the truncation error for the Poisson probabilities gives the value of $\epsilon$ for probabilistic evaluation.

Clearly, the practical use of this basic form of PU is limited by the possibly large value of $\pi_a^\epsilon(t)$. For instance, if $t$ increases, more probability mass will leak away into the absorbing state in the interval $[0, t]$, and consequently $\epsilon$ will increase. Therefore, in the following section we extend the method to an iterative scheme with a changing set of states on which uniformization is applied. During the iterations the obtained accuracy will be increased. We will now discuss this iterative PU scheme.

### 4.3 Iterative Application of Partial Uniformization

We will explain the iterative application of PU by Figure 6 and Figure 7. In Figure 6 PU is carried out on the truncated state space $S_1 = \{(2, 0, 0), (1, 1, 0), (1, 0, 1)\}$, and a subset of the computed elements $\pi_i(n)$ is kept in memory (black circles). More precisely, the element which have outgoing arcs to states outside $S_1$ are stored. Once $\pi_i(t)$ is computed for all elements $i \in S_1$ the stored elements are used for a second application of PU. In this case a new subset $S_2$ is chosen (in Figure 7 $S_2 = \{(0, 2, 0), (0, 1, 1), (0, 0, 2)\}$ while states outside $S_2$ are substituted by a single absorbing state. When doing PU for the subset $S_2$ the results for $\pi_i(n)$ in the 'black circles’ in Figure 6 are taken into consideration (denoted by the dotted arcs in Figure 7). As a consequence, the probability mass that leaked away in Figure 6 to the absorbing state, is now correctly distributed over the states in $S_2$. In Figure 7 again probability mass leaks away after the second application of PU, but less than in the first iteration. By iteratively applying PU to subsets of the state space, more and more probability mass gets distributed correctly over the states.

**Probabilistic evaluation by iterative PU.** The values for the tuple $(\phi, \epsilon)$ can be determined for iterative PU in the same way as for the earlier discussed uniformization methods. At the $k$-th iteration in iterative PU, the probability mass in the absorbing state, summed by the truncation error $\epsilon_N$, and multiplied by $r_{\max}$, bounds the induced error. The above described process can therefore be continued until the probability mass in the absorbing state is small enough. We note that the presented iterative PU scheme is only a first introduction of the algorithm. Many questions arise in the actual application of the technique. For instance, it might not always be possible to store all the necessary elements $\pi_i(k)$ as this number can increase in later iterations. Therefore, probability mass has to be sacrificed. In the particular tool implementation in Section 6 we have chosen to let the user input which sets of states will be considered in the subsequent iterations.
An important issue which also came forward in OU\(^1\) is the determination of a correct value for the uniformization rate \(\lambda\). If PU is applied only once for a truncated state space \(S_1\), the highest outgoing rate for these states is known. However, if PU is iteratively applied, the highest rate for the sets \(S_i, i \geq 2\), can be higher than the one for \(S_1\). In that case, a smooth continuation is not possible. One therefore must take care that the initial uniformization rate is high enough for the whole model. In some cases it will be straightforward to derive a suitable value for the uniformization rate, but it clearly is of interest to find efficient ways of determining the uniformization rate in general.

5 State Space Generation

In the previous sections we have developed uniformization algorithms which deal with large state spaces by addressing only subsets of the state space at the same time. The practical utility of these algorithms is influenced by the order in which these subsets of the state space can be considered. This, in turn, is determined by the state space generation algorithm used to derive the Markov model from the high-level specification. In this section we discuss various candidate state-space generation methods for probabilistic validation.

Assume that the model of interest is unambiguously described by a stochastic Petri net with deterministic initial marking. To construct the underlying Markov model from such an SPN model, a tree-search algorithm is normally employed. Such tree-search algorithms generally operate in a breadth-first (BF) fashion, although one could also imagine a depth-first fashion. A BF state-space exploration approach induces a partition of the state space in which partitions are characterized by the minimum number of steps, \(i.e.\), transition firings, that is required to reach representatives of that partition, starting from the initial marking. Within a partition, not all states have equal probability of occurrence, in steady-state nor at some time instance \(t\), in general. Also, states in blocks “far away” from the initial state might have higher probability of occurrence than states that are very close to the initial state. This depends on the model structure, the involved state transition rates, and the time instance one is interested in.

From the above, it follows that the order in which states are selected is of much importance if one wants to incorporate as much probability mass as possible in as few as possible states. The states to be selected should be those states that cover the largest probability mass at the time instance of interest. This is valid, both when applying OU and PU. We therefore distinguish three state space exploration approaches:

**Breadth-first.** The simplest way of generating states is using the earlier mentioned breadth-first method. This method does not use any special knowledge about the model, the time instance of interest, nor the rates. It just selects states “as they appear”.

\(^1\)Note that hybrid OU/SU as well as OU itself can be seen as specific instances of PU.
In the rest of the paper, the BF method used and compared with, is the one that has been implemented in the SPNP package [1]. We note that, since probability mass might be concentrated around the initial state, BF-based methods have their merits as a technique for probabilistic validation as well.

Maximize “mean time to unknown”. To cope with the problem of the most probable states in a more general way, de Souza e Silva and Ochoa developed a dynamic state space exploration technique based on the so-called mean time to unknown (MTTU) [21].

In this approach, three classes of states are distinguished: states already generated, states that can be reached from the yet generated states in one step (the candidate states), and all other states, represented as a single absorbing state, i.e. the “unknown state”. The next state to be included in the set of generated states, to be selected from the set of candidate states, is selected so as to maximize the mean time it takes to exit from the newly formed set of generates states, i.e. to maximize the mean time until the single “unknown” state is reached. To make such a selection, however, a discrete-time Markov model with size equal to the number of already generated states (plus 1) has to be solved for its steady-state probabilities. This has to be done repeatedly, i.e. for every new state to be selected, which is rather costly, especially when already many states have been generated.

By specifying a required MTTU, a state selection algorithm can be made such that it, using the above selection procedure, continues to select states until the computed MTTU is at least equal to the required one. Care should be taken not to specify a required MTTU that is too large, as that will result in a very large number of states to be included. Instead, one can also require a certain number of states to be used. The MTTU criterion can then be used to select the set of states, of the required size, that covers the largest probability mass.

For details of this method we refer to [21]. We note that this approach has been developed for the cheap derivation of bounds on steady-state measures. We therefore expect the method to be particularly successful when large time instances are of interest.

Heuristic approaches. In order to include more intelligence in the state space exploration than in the BF-case, but not to do this at the high cost of the MTTU-approach, heuristic methods might be a good solution. We propose the following heuristic.

Associate with every state \( i \) a so-called gravity \( \gamma_i \in (0, 1] \). The gravity \( \gamma_i \) expresses the relative importance of state \( i \) in the overall probability mass; the closer \( \gamma_i \) is to 1, the more important state \( i \) is. For the initial state, say state 0, we set \( \gamma_0 = 1 \).

Let \( i \) be a state that belongs to the yet to be generated set of states, and let \( j \) be a member of the set of candidate states. For all such candidate states \( j \), the new gravities
are computed as follows: \( \gamma_j = \gamma_i \times \frac{q_j}{q_i} \), i.e. the gravity of the originating state \( i \) is multiplied by the probability of going to the particular next state \( j \) given a transition out of state \( i \) occurs. The candidate state with the largest gravity is then selected and added to the set of generated states (and removed from the set of candidate states). This procedure is repeated every time a new state has to be added, and the gravities are stored as long as they are necessary for future computations.

The sketched heuristic can be improved in a number of ways. We report on these improvements in a forthcoming paper [11].

6 PET: A Tool for Probabilistic Evaluation

We have developed the probabilistic evaluation tool (PET) to assist us in probabilistically evaluating performance and dependability models. Aim of the tool is to experiment with the described uniformization methods in combination with the various state space exploration techniques. We first present the general operation of the tool in Section 6.1. Then we discuss the employed uniformization and state space generation techniques in Section 6.2 and 6.3 respectively. Some implementation issues are discussed in Section 6.4.

6.1 Basic Operation

A block diagram of the tool is given in Figure 8. From this figure, it becomes clear that the tool user has to input the following: (1) a dependability model as an SPN; (2) the desired time point of evaluation \( t \); and (3) the desired accuracy.

Depending on the particular uniformization technique to be used, the uniformization tool (UNIF) starts requesting the state space generator (SSG) for states, i.e. for lines of the (partial) \( Q \) matrix describing the underlying Markov chain. Memory is reserved dynamically by the memory management unit (MM). Weighting factors in the uniformization summation are provided by the Poisson probabilities calculator (PP); they are calculated only once. If the user requests so, or if needed to fulfill the accuracy requirements, the uniformization tool asks for more states to be generated.

6.2 Supported Uniformization Techniques

PET currently supports the following uniformization techniques:

Standard uniformization. With SU, the total CTMC is generated before the solution process starts. The user provides the accuracy \( \epsilon \) and the time point \( t \), but needs not to provide the uniformization rate \( \lambda \), since it can be computed from the completely generated model.
**Dynamic uniformization.** With DU, the size of the employed part of state space depends on the value of the uniformization rate $\lambda$, which is to be provided by the user, the accuracy $\epsilon$, and the time instance $t$. Knowing $\lambda, \epsilon$ and $t$, the number of required steps in the uniformization process can be calculated. The uniformization tool only asks for those partitions from the state space generator that can be reached in the computed number of steps.

In order to avoid memory problems, the user can also absolutely limit the number of states to be used, either explicitly, or implicitly by limiting the maximum number of steps that are taken into account in the computations. When doing so, the error in the computation is no longer controlled by $\epsilon$. In such cases, it might also happen that not a total partition is included but only a part of it.

**Orthogonal uniformization.** In OU, uniformization is applied state-by-state, either to all states, to a fixed number of states, or to as many states as required to reach the required accuracy. The uniformization rate $\lambda$ is to be provided by the user.

**Partial uniformization.** In PU, the uniformization method is applied on only a part of the state space, extended with a single absorbing state, representing all not-yet-generated states. Two variants exists:

- **without** the possibility to continue afterwards, thus implying that only the state probabilities in the final epochs need to be remembered, *i.e.* “standard” partial uniformization;
- **with** the possibility to continue afterwards, thus implying that the probabilities need to be kept in memory for those states that have outgoing rates to the absorbing state, *i.e.* the iterative partial uniformization is carried out.

An option to stepwise increase the state space in “chunks” of fixed size is also provided.

### 6.3 Supported State Space Generation Techniques

PET allows for the three state space generation techniques presented in Section 5, *i.e.* breath-first, the mean-time-to-unknown method, and the heuristic method. PET has been designed in such a way that new heuristic approaches can easily be included (see also subsection 6.4).

### 6.4 Implementation Issues

PET has been implemented in C for Sun Sparc 1 workstations and covers around 4000 lines of commented code. The SPN models that serve as input are described using the C-based SPN language of SPNP, *i.e.* CSPL [1].

The memory management module reserves and frees memory blocks for the uniformization process. This is done in a way that depends on the type of uniformization employed. For
the OU approach, the state probabilities are calculated “row-wise”, i.e. per state, and the data structure employed is therefore also ordered in rows. Only those rows remain in memory that are required for the computation of states that still need to be subjected to a row-wise uniformization. Since the length of the rows is known \textit{a priori}, but not the number of states, a linked list of rows (normal arrays) is used.

For SU, DU and PU, the state probabilities are computed “column-wise”, i.e. per epoch. The corresponding data structure is therefore also structured in a column-wise fashion. Since the number of states per time step is different in case of DU (it increases) the “height” of the column is made dynamically adjustable. In SU and DU only two columns are needed at any time, i.e. one for the previous epoch and one for the new epoch to be computed.

The data structure for \( P \) stores \( P \) column-wise, as this provides the fastest access during the step-wise computations. As the size of \( P \) can change during the computations, this data structure is made-up of dynamically adjustable linked lists as well.

The Poisson probabilities are calculated only once, and in a stable way, using the approach of Fox and Glynn [5].

The state space generator is currently only virtually present. What actually is implemented is that the overall state space is generated using the SPNP package [1], however, access to the state space and the generator matrix \( Q \) is done in such a way that, from the viewpoint of the uniformization tool, it is as if the state space is generated stepwise and on-demand. This feature of PET makes that new state space generation techniques can easily be incorporated. Once fully tested, they can be integrated in an SPN tool that really transforms the SPN to the Markov model.

7 Examples of Probabilistic Evaluation

In this section we discuss three examples of probabilistic evaluation. These examples are of moderate size but give us a first inside into the potential of the discussed methods. Furthermore, for these models analytic results are available which enables us to compare results with exact results. In Section 7.1 we present a dependability model of a multi-class system with repairs and in Section 7.2 we present a similar model without repairs; these two sections focus on the various uniformization methods. In Section 7.3 we focus on the different state space exploration techniques applied to the introduced models. We finally discuss the obtained results in Section 7.4.

7.1 A Repairable Multiclass Dependability Model

We consider a multi-component class, single repair unit dependability model as depicted in Figure 9. This model has also been used as an example in [10], where approximations for steady-state dependability measures were studied.
There is a place \( Up_c \) for every component class \( c \ (c = 1, \ldots, C) \). Class \( c \) has \( N_c \) components. The transitions \( Fail_c \) model the failure of components. After failure, a component of class \( c \) enters the wait-on-repair place \( WoR_c \). If a repair unit is available, i.e., if there is at least one token in place \( R \), the repair of the component starts via the firing of the immediate transition \( Start_c \). During the repair, the component resides in place \( InRepc \). After the completion of the repair (transition \( Repair_c \)), the component is brought up again and the repair unit becomes available for other repair actions. The initial marking is such that there are \( N_c \) tokens in place \( Up_c \ (c = 1, \ldots, C) \), and that there is one token in place \( R \).

As has been derived in [10], if there is only a single repair unit, the number of states \( NoS \) in the underlying Markov model equals

\[
NoS = 1 + \sum_{T \in \mathcal{T}, T \neq \emptyset} \left( |T| \times \prod_{c \in T} N_c \right),
\]

where \( T \) ranges over all non-empty elements of \( \mathcal{T} \), which is the power set of \( \{1, \ldots, C\} \). Clearly, the state space size increases rapidly with \( C \) and \( N_c \ (c = 1, \ldots, C) \).

The above model is quite generally applicable. The component classes can, for instance, signify replicated groups of components in a power plant control system where a single repair man is on duty for performing repairs of failed components. The overall dependability of such a system can then be expressed as the probability of a certain number of components per group to be up. As another example, the component classes can signify typical system component groups, such as they exist in modern high-speed wide-area networks, i.e., communication links, switches, routers, intermediate packet reassembly servers, and end systems. The repair unit then models a central network management entity. For successful communication between end-users to be possible, a certain subset of components in each component group should be operational. The probabilities on success and failure of such a system can then be computed from the presented model.

For our evaluations, we assume the following numerical parameters. The number of classes \( C \) equals 4, and \( N_1 = 4 \), \( N_2 = 12 \), \( N_3 = 5 \) and \( N_4 = 3 \). The failure rate per component is 0.001 failures per hour, whereas a repair takes 1 hour. There is only one repair unit for doing all the repair. These parameter choices imply that \( NoS = 5159 \).

The complete model can be solved with PET, as well as with SPNP, which both yield exact (and the same) transient state probabilities, thereby using SU. We are interested in time instance \( t = 0.5 \) and apply the accuracy \( \epsilon_N = 10^{-9} \), and for the moment use a breath-first search. The first 10 state probabilities are given in Table 1 in column SU/DU. Notice that the uniformization rate \( \lambda = 1.023 \). This can be seen as follows. There is at most one repair transition enabled at any time, with rate 1. If this is the case, at most 23 components can still be in their initial place, yielding an overall failure rate of 23 \times 0.001. In total, the outgoing rate of a single state is maximally 1.023. For DU, OU and iterative PU we need to set this value ourselves.
Table 1: Comparing SU (5129 states), DU (1392 states), PU (11 states) and OU (11 states) for a repairable dependability model.

<table>
<thead>
<tr>
<th>state</th>
<th>SU/DU</th>
<th>PU</th>
<th>PUcum</th>
<th>OU</th>
<th>OUCum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9.906006x10^{-1}</td>
<td>9.905998x10^{-1}</td>
<td>9.905998x10^{-1}</td>
<td>9.880717x10^{-1}</td>
<td>9.880717x10^{-1}</td>
</tr>
<tr>
<td>1</td>
<td>1.558320x10^{-3}</td>
<td>1.557864x10^{-3}</td>
<td>9.921577x10^{-1}</td>
<td>1.555460x10^{-3}</td>
<td>0.986272x10^{-1}</td>
</tr>
<tr>
<td>2</td>
<td>4.674950x10^{-3}</td>
<td>4.671713x10^{-3}</td>
<td>9.968294x10^{-1}</td>
<td>4.666381x10^{-3}</td>
<td>0.942936x10^{-1}</td>
</tr>
<tr>
<td>3</td>
<td>1.947900x10^{-3}</td>
<td>1.946547x10^{-3}</td>
<td>9.987759x10^{-1}</td>
<td>1.944325x10^{-3}</td>
<td>0.962379x10^{-1}</td>
</tr>
<tr>
<td>4</td>
<td>1.168740x10^{-3}</td>
<td>1.167928x10^{-3}</td>
<td>9.999439x10^{-1}</td>
<td>1.166595x10^{-3}</td>
<td>0.974045x10^{-1}</td>
</tr>
<tr>
<td>5</td>
<td>1.071536x10^{-6}</td>
<td>1.070912x10^{-6}</td>
<td>9.994490x10^{-1}</td>
<td>1.070048x10^{-6}</td>
<td>9.74055x10^{-1}</td>
</tr>
<tr>
<td>6</td>
<td>4.286145x10^{-6}</td>
<td>4.283649x10^{-6}</td>
<td>9.999492x10^{-1}</td>
<td>4.280193x10^{-6}</td>
<td>9.974098x10^{-1}</td>
</tr>
<tr>
<td>7</td>
<td>1.785894x10^{-6}</td>
<td>1.784854x10^{-6}</td>
<td>9.999510x10^{-1}</td>
<td>1.783414x10^{-6}</td>
<td>9.974116x10^{-1}</td>
</tr>
<tr>
<td>8</td>
<td>1.071536x10^{-6}</td>
<td>1.070912x10^{-6}</td>
<td>9.995210x10^{-1}</td>
<td>1.070048x10^{-6}</td>
<td>9.974127x10^{-1}</td>
</tr>
<tr>
<td>9</td>
<td>4.286145x10^{-6}</td>
<td>4.282734x10^{-6}</td>
<td>9.995614x10^{-1}</td>
<td>4.280193x10^{-6}</td>
<td>9.974170x10^{-1}</td>
</tr>
</tbody>
</table>

If we use PU on the first set of 10 states, we obtain the state probabilities as given in Table 1 in the column PU. We also show the total probability mass, in column PUcum, that is included in these 10 states. To obtain the required accuracy $\epsilon$, we require 10 iterations to be taken. Notice that with only 10 epochs in the uniformized Markov chain, at most 1392 states can be reached. When using DU instead of SU, only those states are taken into consideration in the computation, instead of all the 5159 states of the complete model. This of course, will still yield equally accurate results.

The cumulative probability mass that is covered by PU is quite considerable. As indicated with state 9, PUcum = 9.999564x10^{-1} of the total probability mass is covered by the first 10 states; of the 1-complement of this value, i.e. 4.36x10^{-5}, we do not exactly know “where it is”. Since the model we investigate is non-acyclic, this complement might belong to any state not yet taken into account, or to any of the states yet to be taken into account. If the state that denotes “system failure” belongs to the yet unexplored states, we know for sure that that state has a probability of at most 4.36x10^{-5}.

When considering the columns for OU, we see that the values are smaller than the corresponding values for PU. This was to be expected as OU is only exact for acyclic models. In a way, probability mass leaks away from the model. For OU we see that a total probability mass of 2.583x10^{-3} is left unspecified; a considerably larger value than for PU. Again, if the state that denotes “system failure” belongs to the yet unexplored states, we know for sure that that state has a probability lower than 2.583x10^{-3}.

If we increase the number of states taken into account to 25, PU and OU respectively provide the following unresolved probability masses: 2.0x10^{-7} and 2.54630x10^{-3}. For PU there is considerable gain in addressing some more states; for OU this is not so much the case.
Table 2: Comparing PU and OU (both with 25 states) for a repairable dependability model and increasing time instance $t$ of interest.

<table>
<thead>
<tr>
<th>$t$</th>
<th>steps</th>
<th>OUcum</th>
<th>PUcum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>10</td>
<td>$9.880717 \times 10^{-1}$</td>
<td>$9.905998 \times 10^{-1}$</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>$9.912456 \times 10^{-1}$</td>
<td>$9.999888 \times 10^{-1}$</td>
</tr>
<tr>
<td>10</td>
<td>36</td>
<td>$8.059607 \times 10^{-1}$</td>
<td>$9.999106 \times 10^{-1}$</td>
</tr>
<tr>
<td>100</td>
<td>170</td>
<td>$9.294764 \times 10^{-2}$</td>
<td>$9.989095 \times 10^{-1}$</td>
</tr>
<tr>
<td>1000</td>
<td>1222</td>
<td>$3.867921 \times 10^{-11}$</td>
<td>$9.889529 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

If we increase the time point of interest to higher values, more states will be needed to reach the required accuracy. For instance, if $t = 1$, the number of time steps required increases to 13 and the number of reachable states to 2579 which will make DU more expensive to perform. SU remains as expensive as it was before, i.e. more expensive than any of the other methods.

In Table 2 we show the amount of probability mass covered by PU and OU, for increasing values of the time instance $t$. We also show the number of steps that need to be taken to obtain the required accuracy. Both PU and OU in this case only consider the first 25 states.

As becomes very clear here, the OU method is not suitable for non-acyclic models. Much probability mass is lost since cycles in the model are not taken into account. As an example, a very probable path in the Markov model is to first have one component failed, and have it repaired very soon after that, before any other component fails. Such a behavior is not taken into account with OU. With PU, on the other hand, more probability mass is maintained, as the method takes into account cycles in the model.

### 7.2 An Absorbing (Unrepairable) Multiclass Dependability Model

Let us now address an unrepairable variant of the model we used in the previous subsection. If we simply remove the repair facility, the submodels for the individual component classes become independent, and the overall dependability model has a closed-form solution. Therefore, we include an extra dependency in this model. The failure rate of every component is made dependent on the total number of failed components in a linear way; if there are $n$ components down, the failure rates increase (multiplicatively) with a factor $n$ as well.

Since the model is acyclic, OU can be applied, yielding exact results. In Table 3 we present some results. We have set the required probability mass to be covered to be at least equal to 0.99999, i.e. the non-evaluated probability mass is at most $10^{-5}$. For different time values $t$, PET computes the number of required steps in the uniformization process. Depending on the required accuracy, the OU process is performed for more or less states. As can be seen from this table, the non-evaluated probability mass is less than $10^{-5}$. We finally also have
<table>
<thead>
<tr>
<th>$t$</th>
<th>steps</th>
<th>states</th>
<th>non-evaluated</th>
<th>$\text{Pr}{\text{no failures}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>5</td>
<td>5</td>
<td>$2.755668 \times 10^{-6}$</td>
<td>9.976029 $\times 10^{-1}$</td>
</tr>
<tr>
<td>0.2</td>
<td>5</td>
<td>7</td>
<td>$8.858598 \times 10^{-6}$</td>
<td>9.952115 $\times 10^{-1}$</td>
</tr>
<tr>
<td>0.5</td>
<td>6</td>
<td>12</td>
<td>$7.395004 \times 10^{-6}$</td>
<td>9.880717 $\times 10^{-1}$</td>
</tr>
<tr>
<td>1.0</td>
<td>7</td>
<td>14</td>
<td>$6.867516 \times 10^{-6}$</td>
<td>9.762857 $\times 10^{-1}$</td>
</tr>
<tr>
<td>2.0</td>
<td>8</td>
<td>27</td>
<td>$9.700907 \times 10^{-6}$</td>
<td>9.531338 $\times 10^{-1}$</td>
</tr>
<tr>
<td>5.0</td>
<td>11</td>
<td>59</td>
<td>$8.359777 \times 10^{-6}$</td>
<td>8.886920 $\times 10^{-1}$</td>
</tr>
<tr>
<td>10.0</td>
<td>14</td>
<td>129</td>
<td>$9.951901 \times 10^{-6}$</td>
<td>7.866279 $\times 10^{-1}$</td>
</tr>
<tr>
<td>20.0</td>
<td>18</td>
<td>356</td>
<td>$9.888459 \times 10^{-6}$</td>
<td>6.187843 $\times 10^{-1}$</td>
</tr>
<tr>
<td>50.0</td>
<td>27</td>
<td>1225</td>
<td>$9.585102 \times 10^{-6}$</td>
<td>3.011942 $\times 10^{-1}$</td>
</tr>
</tbody>
</table>

Table 3: Applying OU on a non-repairable dependability model and increasing time instance $t$ of interest.

<table>
<thead>
<tr>
<th>required</th>
<th>states</th>
<th>non-evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 - 10^{-1}$</td>
<td>1</td>
<td>$2.371429 \times 10^{-2}$</td>
</tr>
<tr>
<td>$1 - 10^{-2}$</td>
<td>3</td>
<td>$8.085906 \times 10^{-3}$</td>
</tr>
<tr>
<td>$1 - 10^{-3}$</td>
<td>5</td>
<td>$2.717139 \times 10^{-4}$</td>
</tr>
<tr>
<td>$1 - 10^{-4}$</td>
<td>11</td>
<td>$6.604565 \times 10^{-5}$</td>
</tr>
<tr>
<td>$1 - 10^{-5}$</td>
<td>14</td>
<td>$6.867516 \times 10^{-6}$</td>
</tr>
<tr>
<td>$1 - 10^{-6}$</td>
<td>28</td>
<td>$8.163139 \times 10^{-7}$</td>
</tr>
<tr>
<td>$1 - 10^{-7}$</td>
<td>33</td>
<td>$9.269787 \times 10^{-8}$</td>
</tr>
</tbody>
</table>

Table 4: Applying OU on a non-repairable dependability model and increasing accuracy requirements.

included the probability of a totally functioning system, i.e. the probability of having no failed components at all. As can be observed, this probability decreases as $t$ increases. Also notice that as $t$ increases many more states need to be included. For the uniformization rate $\lambda$ we have always used 0.126. As before, this value can be precomputed if one has understanding of the model.

Finally, we investigate the number of required states when changing the required probability mass to be evaluated. We do this for $t = 1$; the number of required steps is therefore always 5, as can be computed from the Poisson probability distribution. Table 4 presents the results. As can be observed, the number of states increases, albeit not dramatically. The actually non-evaluated probability mass indeed is conform the requirements.
7.3 Influence of State-Space Exploration Techniques

In this section we present a comparison of the three implemented state-space exploration techniques. We do this for a non-repairable model in Section 7.3.1 and for a repairable model in Section 7.3.2.

7.3.1 Non-Repairable Model

In this section we reconsider the repairable multiclass dependability model of Section 7.2. In that section, we evaluated various uniformization strategies for probabilistic evaluation, thereby always taking states into account in the order as they “appeared” from a breath-first generation. In this section we compare the three indicated state-space exploration methods.

We first address the non-evaluated probability mass, when only a limited number of states is taken into account. This value is obtained by doing PU over the generated states. We decide, quite arbitrarily, to take only 10 states into account. Then, the three different state space exploration techniques generate the following states (in the given order):

\[
\begin{align*}
\text{BF} & \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
\text{HEU} & \{0, 2, 9, 3, 1, 4, 10, 25, 6, 11\} \\
\text{MTTU} & \{0, 2, 9, 3, 10, 1, 6, 26, 19, 4\}
\end{align*}
\]

First of all, we observe that the BF states differ substantially from the states generated by HEU and MTTU. The latter two exploration techniques also differ slightly. Note that the indicated sets also indicate the ordering of the selection of the states, e.g. with the MTTU-approach, state 10 is selected as 5-th, whereas it is selected as 7-th with HEU.

With these 10 states, PU is employed. The non-evaluated probability mass is presented in Table 5, for increasing time values \( t \). As can be observed, for small values of \( t \), the accuracy is higher. This is easy to understand, as the probability of moving away from the first few states increases with larger \( t \). Also observe that the set of states selected by HEU and MTTU result in smaller non-evaluated probability masses, especially for larger \( t \) (relatively). In this example, the heuristic method works best, even better than the more expensive MTTU approach.

If the non-evaluated probability mass is not small enough when using only 10 states, we can increase the number of allowed states to 20. Then, the following states are selected by the three heuristics (in the given order):

\[
\begin{align*}
\text{BF} & \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19\} \\
\text{HEU} & \{0, 2, 9, 3, 1, 4, 10, 25, 6, 11, 26, 55, 19, 7, 12, 27, 13, 56, 5, 8\} \\
\text{MTTU} & \{0, 2, 9, 3, 10, 1, 6, 26, 19, 4, 11, 25, 56, 27, 45, 20, 46, 89, 7, 57\}
\end{align*}
\]

Repeating the uniformization process (PU) for these 20 states, we obtain Table 6. For smaller values of \( t \) the BF method does best. When \( t \) increases, the MTTU and the HEU method do better. Again, as we have seen before, the HEU method does better than the MTTU method, even though the MTTU method is more expensive.
### Table 5: Applying PU on the first 10, differently generated states of a non-repairable dependability model.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\text{BF}$</th>
<th>$\text{HEU}$</th>
<th>$\text{MTTU}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>$3.1033\times10^{-5}$</td>
<td>$1.6698\times10^{-5}$</td>
<td>$2.5471\times10^{-5}$</td>
</tr>
<tr>
<td>1.0</td>
<td>$1.2425\times10^{-4}$</td>
<td>$6.7562\times10^{-5}$</td>
<td>$1.0177\times10^{-4}$</td>
</tr>
<tr>
<td>5.0</td>
<td>$3.1164\times10^{-5}$</td>
<td>$1.8247\times10^{-3}$</td>
<td>$2.5211\times10^{-3}$</td>
</tr>
<tr>
<td>10.0</td>
<td>$1.2398\times10^{-2}$</td>
<td>$7.8106\times10^{-3}$</td>
<td>$9.9615\times10^{-3}$</td>
</tr>
</tbody>
</table>

### Table 6: Applying PU on the first 20, differently generated states of a non-repairable dependability model.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\text{BF}$</th>
<th>$\text{HEU}$</th>
<th>$\text{MTTU}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>$4.0529\times10^{-7}$</td>
<td>$9.9091\times10^{-7}$</td>
<td>$1.1519\times10^{-5}$</td>
</tr>
<tr>
<td>1.0</td>
<td>$3.2103\times10^{-6}$</td>
<td>$4.9109\times10^{-6}$</td>
<td>$4.6152\times10^{-5}$</td>
</tr>
<tr>
<td>5.0</td>
<td>$3.7180\times10^{-4}$</td>
<td>$3.0276\times10^{-4}$</td>
<td>$1.1699\times10^{-3}$</td>
</tr>
<tr>
<td>10.0</td>
<td>$2.7027\times10^{-5}$</td>
<td>$2.0128\times10^{-3}$</td>
<td>$4.7686\times10^{-3}$</td>
</tr>
<tr>
<td>50.0</td>
<td>$1.5740\times10^{-1}$</td>
<td>$1.2846\times10^{-1}$</td>
<td>$1.3153\times10^{-1}$</td>
</tr>
</tbody>
</table>
Table 7: Comparing the application of PU for the three state-space exploration techniques for different numbers of states at time $t = 1$.

### 7.3.2 Repairable Model

In this section we reconsider the repairable multiclass dependability model of Section 7.1. We will study the influence of the MTTU parameter on the choice of states and, finally, on the non-evaluated probability mass. We do so as follows. We stepwise increase the *required MTTU*, and generate all those states that are needed to fulfill the MTTU requirement. As can be observed from Table 7, the number of generated states increases with increasing required MTTU. We then employ PU over those states and tabulate the non-evaluated probability mass. For the same number of states, we also employ PU for the two other state-space exploration techniques. These techniques may use different states in their evaluation, however, the *number* of employed states is the same.

In Table 7 we present the non-evaluated probability mass for $t = 1$. As can be observed, both the MTTU- and the HEU-method outperform the BF state-space exploration method by several orders of magnitude, especially in case of moderate number of states. The MTTU-approach appears to be slightly better than the HEU method, however, it is far more costly. The HEU method is about equally expensive as the BF method, whereas the computations required for the MTTU-method increases with the number of states involved.

### 7.4 Discussion of Examples

In this section we have presented a number of probabilistic evaluations, using the newly developed uniformization and state-space exploration techniques. It is difficult to generally state conclusions as all the results obtained are model-dependent. However, the used example models seem to be fairly representative for models used in literature, and some of the derived conclusions might therefore hold more general.

Regarding the uniformization techniques, both OU and PU outperform standard uni-
formization for the example models. OU works best with acyclic models; it can be applied to non-acyclic models as well but at the cost of accuracy. In both OU and PU, the memory bottleneck is no longer the model size, as states are treated individually or group-wise, instead of all at once. PU, and especially its iterative variant, is very generally applicable and is a promising technique to probabilistically evaluate very large models. Further experiments with large or infinite models are in order to establish the practical utility of the proposed uniformization methods.

Regarding the state-space exploration techniques, both the heuristic and the MTTU-approach have shown to be advantageous in comparison to a BF-approach. The MTTU approach seems to operate best for relatively high time instances. As the heuristic approach always operates about equally well, or even better, it may be the preferred method, as it is less computation intensive (see also [11]).

8 Concluding Remarks

In this paper we have introduced the probabilistic evaluation as a solution approach for transient performance and dependability measures of large Markov reward models. In probabilistic evaluation a bound for the measure is obtained based on a partial and/ or iterative search through the state space. We have introduced several specific uniformization techniques, among which orthogonal uniformization and partial uniformization, to deal with Markov models with a very large or even infinite state space. These methods provide bounds on dependability measures with only limited computational effort and without generating the complete model. The bounds can be made tighter through iteratively increasing the probability mass that is taken into account.

We also have proposed and evaluated various techniques for the selection of which states to consider during the solution. Three state space exploration methods have been compared, of which the newly proposed heuristic method performed best, i.e. it gave good results at very little cost.

The methods for probabilistic evaluation have been implemented in a prototype package called PET, which supports our extensions of the uniformization technique as well as three different state-space exploration techniques. We have applied our methods to a number of dependability models. For those examples, our methods turn out to perform very well, i.e. we are able to provide very tight bounds on dependability measures of interest at very little cost. Also, we are able to obtain bounds for models which simply cannot be addressed in full because the state space is too large to generate, or even infinite.

Issues for future research concentrate around the further improvement of the presented algorithms, since several implementation issues still have to be resolved. Definitely, the actual integration of the proposed algorithms in existing modeling packages will be an important step to further explore the practical utility of the proposed probabilistic evaluation methods.
Acknowledgements

The work in this paper is based on the Ph.D. thesis of A.P.A. van Moorsel, written at the University of Twente. At that time, B.R. Haverkort was an assistant professor at the University of Twente. Part of the work has been carried out when A.P.A. van Moorsel was a visiting researcher at the UltraSAN group at the University of Arizona in the spring of 1992, in cooperation with W.H. Sanders (now with the University of Illinois). This visit was financed by the Netherlands Organization for Scientific Research (NWO), grant R62-385 and SIR13-1260.

Furthermore, the following students contributed to this work: S. de Graaf implemented the first version of OU [6], E. Klein contributed to developing the uniformization algorithms and implemented them in PET [13], and K. Wolters incorporated the state space exploration methods in PET.

References


Figure 1: Example Machine-Repairman Model.

Figure 2: Computational scheme of SU. The circles denote $\pi_i(k), k = 0, 1, \ldots, N$. To compute $\pi_i(n + 1)$ the elements $\pi_i(n)$ have to be kept in memory (black circles).
Figure 3: Computational scheme of OU for an acyclic model. To compute $\pi_{(0,2,0)}(k)$, the elements $\pi_{(1,1,0)}(k)$ and $\pi_{(1,0,1)}(k), k = 0, 1, \ldots, N$, are kept in memory.

Figure 4: Computational scheme of hybrid OU/SU for a partly acyclic model. SU has been carried out over the subset $S_1 = \{(2, 0, 0), (1, 1, 0), (0, 2, 0)\}$, keeping ascenders of states outside $S_1$ in memory. Iteration $n$ of SU over the subset $S_2 = \{(1, 0, 1), (0, 1, 1)\}$ has just been completed.
Figure 5: Computational scheme of DU. The circles $\pi_i(k), k = 0, 1,$ with a crossbar pattern denote states $i$ that do not have to be generated at iteration $k$.

Figure 6: Computational scheme of PU. Circles $\pi_i(k)$ marked with an ‘A’ denote states substituted by a single absorbing state. If iterative PU is applied, some results (denoted by the black circles) have to be stored.
Figure 7: Computational scheme of iterative PU, the second iteration. The dotted arcs denote the use of elements stored in the first iteration according to Figure 6.

Figure 8: Overview of PET.
Figure 9: Multi-component class SPN dependability model with repair unit.