Approximate Performability Analysis using Generalized Stochastic Petri Nets

Boudewijn R. Haverkort
Tele-Informatics and Open Systems
University of Twente
P.O. Box 217, 7500 AE Enschede, the Netherlands

Abstract

In this paper we address the problem of calculating performability measures from performability models of fault-tolerant computer systems. Since these systems tend to be large and complex, the corresponding performability models will in general also be large and complex. To alleviate the largeness problem to some extent we use generalized stochastic Petri nets to describe the models. Still however, many models can not be solved with the current numerical techniques, although they are conveniently and often compactly described. In this paper we discuss two heuristic state space truncation techniques that allow us to obtain very good approximations while only assessing a few percent of the overall state space. We give examples of the usage, but also theoretical evidence in the correctness of the employed truncation techniques. We furthermore show that GSPNs are very suitable for implementing (describing) the proposed truncation techniques. We finally address related work.

1 Introduction

Current day fault tolerant and distributed computer systems (FTDCSs) are used more and more for highly responsible tasks. The well operation of these systems therefore becomes more and more important since the failure of these types of systems generally involve high costs, or even worse, may cause severe environmental damage or human loss. Because the correct and timely operation of FTDCSs is of importance, these types of systems should be designed in such a way that they fulfill predefined requirements regarding their performance and dependability, i.e. their performability. In order to be able to assess these systems' performability, there is a strong need for modelling them in order to obtain estimates of the expected performability, already in early phases of the design.

With modelling FTDCSs we encounter (at least) two problems. The first problem is the generation of the model; the second problem is the model solution. Clearly, both problems are aggrevated by what is generally called the largeness of the model, i.e. due to the fact that the systems we are willing to analyse are very complex and large themselves, their corresponding models will have the same complicating features.

The first problem (model generation) is generally partly alleviated by the use of high level mechanisms to describe the models. From these high level models a lower level, mathematically assessable model is then derived automatically and subsequently analyzed. The results from the analysis of the lower level model are then interpreted (translated back) in the context of the high level description technique. Although we suggest a two-level approach here, a modelling approach involving multiple abstraction levels is also possible [9].

We have chosen to use generalized stochastic Petri nets for the description of the dependability aspects of the performability models, i.e. we use a GSPN for the generation of a Markov chain. The performance aspects are brought into the model by associating rewards with the states of the Markov chain. Although the derivation of the rewards is in general a problem in itself [9, 10], this will not be emphasized in this paper.

For the second mentioned problem (model solution) various solutions exist (see also Section 5). Although the capabilities of current day workstations increase rapidly, the models we are willing to analyze seem to increase in size at about the same pace. Today, the class of Markov models that can be coped with numerically is enormous—state spaces of hundreds of thousands states can be handled by some tools—there will always remain models that are not soluble. What we should do then, is simply avoid generating these large models, not by not at all addressing the systems that yield these models, but by using insight in the oper-

TH0386-3/91/0000/0300/$01.00 © 1991 IEEE

ation of the system, in the parameters involved, and in the measures we want to obtain from the model, in order to construct soluble models that yield fairly reasonable approximations of the desired results.

In this paper we discuss state space truncation as a basic heuristic to come to approximate models. State space truncation can be useful in two cases:

1. To reduce the model size only to speed up the computation for evaluating a model. One should be careful with this because decreasing the model size implies increasing the inaccuracy.

2. To reduce the model size to such a level that it can be handled. Models can be imagined with millions of states. E.g. by allowing not too many simultaneous failures, the model can be generated and evaluated.

This paper is organized as follows. In Section 2 we discuss the class of models that we need to solve, first at the level of Markov chains, then at the level of GSPNs. In Section 3 we then discuss truncation techniques for time-reversible models, whereas in Section 4 truncation techniques for general models are addressed. In both sections, we first informally discuss the truncation techniques and the models themselves, then show numerical results, and finally go in the theory that proves the correctness of the employed truncations. Section 5 discusses related work and Section 6 concludes the paper.

2 Problem statement

In this section we discuss the performability models we will address in the rest of this paper. We do this first at the Markov chain level in Section 2.1, and then at the GSPN level in Section 2.2.

2.1 Markovian performability models

The approach generally followed in performability modelling is to describe the system dependability aspects by a continuous time Markov chain and to associate rewards with the states of the Markov chain (see e.g. [13, 19, 18]). We follow this approach here too.

We consider a Markov chain on the state space M. The number of states of the Markov chain is denoted $\#M$. This Markov chain is totally described by the $\#M \times \#M$ generator matrix $\mathbf{Q} = [q_{ij}]$ and the initial probability vector $\pi = [\cdots, \pi_m, \cdots]$. We furthermore have the reward function $r : M \to \mathcal{R}$ which associates with every state $m \in M$ a real-valued reward $r(m)$. Let $p = [\cdots, p_m, \cdots]$ denote the steady state probability distribution of the Markov chain. Then the steady state performability[1] P is defined as

$$P = \sum_{m \in M} p_m r(m). \tag{1}$$

Many papers on performability stress the importance of transient measures as opposed to the above given steady state measure. Steady state measures are however useful for a number of reasons. For highly dependable systems such as telephone and telecommunication switches, long run performability measures are often part of the requirement specifications, so that calculation of them is of importance. Furthermore, when dealing with highly dependable systems, the initial performability will in general be very close to the steady state performability. Finally, under natural monotonicity conditions, the steady state performability provides a lower bound for the point as well as for the interval performability [14]. Thus, obtaining the steady state performability implies obtaining a lower bound for other performability measures. If the above conditions do not hold, steady state measures can still be used as rough first indications.

Given a Markov chain and a reward function, the main problem for obtaining P lies in the derivation of the steady state vector p. Deriving p only requires the system of linear equations

$$p\mathbf{Q} = 0, \quad \sum_{m \in M} p_m = 1, \tag{2}$$

to be solved, which can either be done directly (Gaussian elimination) or iteratively (Gauss-Seidel iterations or successive over-relaxation). When dealing with large models, the iterative methods are preferable.

When looking solely at dependability, all the rewards will either be 0 or 1. We can then partition the set M in a set M_u of "up" states, and a set M_d of "down" states, i.e. $M = M_u \cup M_d$, such that $r(m) = 1$ whenever $m \in M_u$, and 0 elsewhere.

2.2 GSPN performability models

Performability models as described in the previous section at the Markov chain level, can easily described at the GSPN level, as shown by e.g. [2, 9, 12, 16, 17].

[1] When we are addressing models in which the rewards $r(m)$ are either 0 or 1, we formally deal with dependability models rather than with performability models. However, since the former can be seen as a special case of the latter, we most often use the latter name.

Without repeating all the details of GSPN models here, we define a GSPN G with initial marking M_0. The set of all markings is called the reachability set M. Under the usual boundedness conditions, this GSPN can be translated in a Markov chain with $\#M$ states, generator matrix $\mathbf{Q}$ and initial probability vector π.

With every GSPN marking m, a reward $r(m)$ can be associated. We assume that all the rewards are explicitly available.

Important to note here is that all actual calculations are done on Markov chains. In first instance however, the models as well as the employed truncations are described at the GSPN level.

3 Time-reversible performability models

In this section we address truncation techniques for time-reversible performability models. In Section 3.1 we discuss the basic heuristic that is used. We illustrate this heuristic in Section 3.2 and show the correctness of the heuristic in Section 3.3.

3.1 The basic truncation heuristic

In highly dependable computer systems the components are generally of such a high quality that the probability that two or more components of the same type are down simultaneously is negligible. Furthermore, if in a system a particular subset of the components is down, the system might be switched-off until repair has been completed. When switched-off, the non-failed components do not fail. Consequently, the notion of stopping failure processes can be used as the basis for state space truncation technique.

An important point of notice here is the fact that an intuitive increase in the system dependability can actually *decrease* the dependability (see e.g. [1]). Care should therefore be taken in applying heuristics! We propose two truncation techniques here:

1. To obtain an upper bound on the system performability we apply a state space truncation such that all states signifying operational states still exists in the truncated model, i.e. only "down" states are pruned.

2. To obtain a lower bound on the system performability we apply a state space truncation such that all states signifying down states still exists in the truncated model, i.e. only "up" states are pruned.

3.2 A numerical example

To illustrate the two truncation techniques we address a system with 10 identical components. The components can fail with individual failure rate f and be repaired by a single repair unit with rate r. All distributions are assumed to be of exponential type. This type of model can easily be described by the GSPN in Figure 1, where the number of tokens in places up and down respectively state the number of component that is up or down. The transition `fail`, with marking dependent rate, models the failure of components. Transition `repair` models the repair of components.

The underlying birth-death model has state space $M = \{0, \cdots, 10\}$, where state m denotes that there are m components up (corresponding to m tokens in place up). There are (failure) transitions from state m to state $m-1$ ($m = 1, \cdots, 10$) with rate $m \times f$, where f is the failure rate per component. There are repair transitions from state m to state $m+1$ ($m = 0, \cdots, 9$) with rate $r = 1$. The overall system is assumed to be operational when there are 8 or more components up, i.e. $M_u = \{8, 9, 10\}$.

The proposed truncations can be "implemented" by not allowing all components to be down. This can be accomplished by adding a (multiple) inhibitor arc from place `down` to transition `fail`. On the other hand, not allowing all components to be up can be accomplished by adding a (multiple) inhibitor arc from place up to transition `repair`. Combinations are of course also possible.

In Table 1 we show some numerical results. The first column states the component failure rate f. The second column shows the exact steady state performability[2] P.

The third column shows the range $[\widetilde{P}_l, \widetilde{P}_u]$ where $\widetilde{P}_l$ is based on a truncated model with state space $\{0, \cdots, 8\}$ and $\widetilde{P}_u$ on a truncated model with state space $\{7, \cdots, 10\}$.

The fourth column shows the range $[\widetilde{P}_l, \widetilde{P}_u]$ where $\widetilde{P}_l$ is based on a truncated model with state space $\{0, \cdots, 9\}$ and $\widetilde{P}_u$ on a truncated model with state space $\{6, \cdots, 10\}$.

Comparing the third and fourth column, it is clear that when more states are included in the approximation, a higher accurracy is obtained.

The fifth column shows an approximation $\widetilde{P}_*$ based on only two up and two down states, i.e. on $\{6, 7, 8, 9\}$. As can be observed from this column, a small approximate model with states concentrating around the up/down boundary might do better than a model with

[2] We basically address the steady state availability here.

all down (up) states and only a few up (down) states, although in general it is not sure anymore whether an upper or a lower bound is obtained in these cases. A general observation from Table 1 is that for more realistic parameters (the lower lines in Table 1) the bounds become tighter.

3.3 Proof of the correctness of the truncation techniques

The proposed truncation techniques work out right in case the underlying Markov chain is time-reversible [20]. A Markov chain with generator matrix $\mathbf{Q} = [q_{ij}]$ and steady state distribution $p = [\cdots, p_i, \cdots, p_j, \cdots]$ is time-reversible if and only if $p_i q_{ij} = p_j q_{ji}$, for all i, j.

Now, consider a truncated version of the model on state space $\widetilde{M}$. Again this state space is partitioned into two disjoint subsets, i.e. $\widetilde{M} = \widetilde{M}_u \cup \widetilde{M}_d$ with $\widetilde{M}_u \subseteq M_u$ and $\widetilde{M}_d \subseteq M_d$. The measure of interest is the steady state performability $\widetilde{P} = \sum_{i \in \widetilde{M}_u} \widetilde{p}_i$, where $\widetilde{p} = [\widetilde{p}_1, \cdots, \widetilde{p}_{\#\widetilde{M}}]$ is the steady state distribution.

Upper bound. The truncated model now provides an upper bound on the steady state performability P whenever all the up states of the original model are included in the truncated model, i.e. if $\widetilde{M}_u = M_u \Rightarrow \widetilde{P} \geq P$, where the equal sign applies only if $\widetilde{M}_d = M_d$.

To understand this, we have to recall a well-known result for time-reversible stochastic processes (see e.g. Walrand [20, Fact 3.7.1]). Walrand's Fact 3.7.1 basically states that the steady state probabilities in the truncated Markov chain are equal to the ones in the original model apart from normalization. This comes from the fact that the original model is time-reversible. Thus, we have $\widetilde{p}_i = p_i/\sigma$ for all $i \in \widetilde{M}$ and $\sigma = \sum_{i \in \widetilde{M}} p_i$. Using this result we can easily proof the above claim as follows. Clearly, $\sigma = \sum_{i \in \widetilde{M}} p_i \leq 1$. Recalling that $\widetilde{P} = \sum_{i \in \widetilde{M}_u} \widetilde{p}_i$ and $P = \sum_{i \in M_u} p_i$, the question is: $\widetilde{P} \geq P$? Since $\widetilde{p}_i = p_i/\sigma$ and $\widetilde{M}_u = M_u$ we have

$$\widetilde{P} = \sum_{i \in \widetilde{M}_u} \widetilde{p}_i = \sum_{i \in \widetilde{M}_u} \frac{p_i}{\sigma} = \frac{\sum_{i \in M_u} p_i}{\sigma} = \frac{P}{\sigma} \geq P, \quad (3)$$

where the $\geq$ sign comes from the fact that $\sigma \leq 1$.

Lower bound. By a similar argument as above, it can be shown that the truncated model provides a lower bound on the steady state performability P whenever all the down states of the original model are included in the truncated model, i.e. if $\widetilde{M}_d = M_d \Rightarrow \widetilde{P} \leq P$, where the equal sign applies only if $\widetilde{M}_u = M_u$.

A problem with the sketched approach is that a lot of practical models are not time-reversible. Consequently, the presented truncation techniques can not be applied with proven correctness. Still however, we can "illegally" apply them. However, we can then not state whether upper or lower bounds are obtained.

4 General performability models

For more general Markov chain performability models it is not possible to derive explicit upper and lower bounds by truncation. What is possible is deriving approximate performability measures together with an estimate for the maximum error made. In this section we will discuss this type of truncation technique. We first introduce the basic GSPN model on which we have performed the experiments. We also explain how the truncation strategies are implemented at the GSPN level. We then discuss various numerical studies we did. We finally state the theoretical result underlying the truncations.

4.1 Basic GSPN performability model

The model we used for the experiments is depicted in Figure 2. Basically, there is a place `Upc` for every component class c ($c = 1, \cdots, C$). Class c has N_c components. The transitions `Failc` model the failure of components. After failure, a component of class c enters the wait-on-repair place `WoRc`. 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 `Startc`. During the repair, the component resides in place `InRepc`. After the completion of the repair (transition `Repairc`), the component is brought up again and the repair unit becomes available for other repair actions.

We distinguish 4 different up-functions. Denoting with $n_c(m)$ the number of up components of class c in marking m, we have:

- $Up_1(m) = \bigwedge_{i=1}^{C}(n_i(m) = N_i)$: the system is operational if all components are operational;
- $Up_2(m) = \bigwedge_{i=1}^{C}(n_i(m) \geq N_i - 1)$: the system is operational if at most one component per class has failed;

- $Up_3(m) = \bigwedge_{i=1}^{C}(n_i(m) \geq 1)$: the system is operational if at least one component per class is operational;

- $Up_4(m) = \bigwedge_{i=1}^{C}(n_i(m) \geq g_i)$: the system is operational if at least g_i components per class are operational.

The following truncation strategies have been used to obtain the approximations:

T0: no truncation;
T1: maximum of 1 down component per class;
T2: maximum of 2 down components per class;
T3: maximum of 3 down components per class;
T4: maximum of 4 down components;
T5: maximum of 5 down components;
T6: maximum of 7 down components;
T7: maximum of 8 down components.

The truncation strategies are easily implemented using only a few extra places and some (multiple) inhibitor arcs. First consider the "per component class"-truncations (T1-T3). For every class c a new place `Downc` is introduced. A token is put in `Downc` upon the failure of a component of class c by adding an arc from `Failc` to `Downc`. A token is removed from `Downc` whenever a component of class c has finished its repair by adding an arc from `Downc` to the transition `Repairc`. A (multiple) inhibitor arc is finally introduced from `Downc` to transition `Failc`. Thus, depending on the inhitor arc multiplicity, the failure process of components of class c is stopped whenever some threshold of already down components is reached.

The "overall"-truncations (T4-T7) can be established in a similar fashion. Instead of introducing an extra place for every component class, only a single extra place `Down` is necessary. From all the transitions `Failc` an arc is added to place `Down`. Furthermore, arcs are added from place `Down` to all the transitions `Repairc`. From place `Down` (multiple) inhibitor arcs can be added to the failure transitions `Failc`. Note that the multiplicity of these arcs can be different for the various transitions `Failc`. Combinations of the two sketched truncation extensions are of course also possible.

Assuming a reward equal to 1 whenever the system is up and 0 elsewhere, the performability becomes

$$P_i = \sum_{m \in M \text{ and } Up_i(m)} p(m). \qquad (4)$$

4.2 Numerical evaluation

In the numerical example we address a model with $C = 4$ classes of components, with $N_1 = 4$, $N_2 = 12$, $N_3 = 5$ 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. The parameter g_i in $Up_4()$ equals 3, independently from i. The measure P_i corresponds to the usage of up-function $Up_i()$.

Let us first address the numerical evaluation of the system in case there is no truncation (T0). Table 2 shows the exact results for the untruncated model. The first column shows the used truncation strategy. The second column shown the number of GSPN markings (tangible+vanishing) and the third column shows the state coverage, i.e. the percentage of states that is used in the approximate model, as compared to the original model (T0). Columns $P1$ through $P4$ show the four different performability measures, corresponding to up-functions $Up_1(m)$ through $Up_4(m)$.

We can now compare the above exact result with the results of the truncations T1 through T7. These results are listed in Table 3 where $\tilde{P}_i$ is the approximation of P_i.

Let us now discuss the results of the various truncation strategies. First thing to observe is that even for truncation strategies with very low state coverage factors the approximations are relatively good. Taking into account the relatively large values for the component failure rates, we think that in practical applications the above observation can be used as engineering rule of thumb.

In the model there is only 1 repair unit available; in all evaluations this repair unit was idle for more than 97.6% of its time. The inclusion of an extra repair unit will therefore not significantly increase the system performability.

Concluding, the example shows that truncation is indeed a very powerful technique in obtaining performability approximations.

We now will show how the state space of our "basic" GSPN performability model grows with increasing number of component classes and with increasing number of components per class. After having shown some models with a prohibitively large state space, we will solve one of them by using truncation. Note that we will not be able to "check" the truncated model; the original model simply can not be solved.

The number of (tangible) states of the Markov chains underlying the GSPNs can easily be derived. Recalling some notation, we have C classes of compo-

nents, numbered 1 through C, with N_c components for class c. Let $\mathcal{C} = \{1, \cdots, C\}$ and let $\mathcal{T}$ be the power set of $\mathcal{C}$. The number of states NoS then equals

$$NoS = \sum_{T \in \mathcal{T}} \max\{1, \#T\} \times \prod_{c \in T} N_c, \qquad (5)$$

where the product $\prod_{c \in T} N_c$ equals 0 when $T = \emptyset$. Formula (5) can be understood when it is noticed that every $T \in \mathcal{T}$ is the set of numbers of classes for which at least one component is down. Given that the classes with at least one component down are given by T, the single repair unit can either be busy with repairing one of the $\#T$ components or do nothing (in case $\#T = 0$), which explains the factor $\max\{1, \#T\}$. Given that for all classes $c \in T$ there is at least one component down but maximally N_c, implies that there are N_c possibilities per class, which explains the latter product.

Given Equation (5), we can calculate the number of states for models like the one in Figure 2 with more classes of components and varying numbers of components per class. In Table 4 we present some state space sizes for varying C and N_c values. As can be observed from Table 4 the state space sizes increase very rapidly; direct analysis of the models 7 through 11 is practically not feasible.

We will now address an approximate version of model number 9 of Table 4. As can be seen from Table 4 this model has 3240469 states. We will use six truncation strategies, numbered S1 through S6, where the strategy number denotes the maximum number of simultaneously failed components. We use the three up-functions $Up_1(m)$, $Up_2(m)$, and $Up_3(m)$, as already defined in Section 4.

In Table 5 we show the results. The first column states the used model truncation technique. The second column shows the number of tangible and vanishing markings of the GSPN model. The number of tangible markings corresponds to the number of Markov chain states. The column U_{rep} indicates the probability that there is no repair unit in the place repair, i.e. the repair unit unavailability. Columns $\widetilde{P}_1$ through $\widetilde{P}_3$ give the performability approximations, where $\widetilde{P}_i$ corresponds to the case where up-function Up_i is used.

What can be observed is that the results for truncations S3 through S6 do not differ for their first 5 decimals. We thus seem to have a five digit accuracy when we only take into account 0.0078 percent of all possible states!

4.3 A general Markov chain truncation technique with provable error bounds

In this section we describe a recently developed technique to obtain error bounds for the truncation of arbitrary continuous time Markov chains (see van Dijk [3, 4, 5]).

Model and truncation description. Consider the Markovian model as discussed in Section 2. Define the so-called uniformized one-step transition matrix $\mathbf{H}$ as $\mathbf{H} = \mathbf{I} + \mathbf{Q}/q$, with $q \geq \max_i\{q_i\}$. We denote by $\mathbf{H}^k$ the k-th power of the one-step matrix $\mathbf{H}$, and by $\mathbf{H}^k(i,j)$ the i,j-th element of that matrix. Furthermore, for arbitrary reward function r, the functions V^k can be defined as

$$V^k(i) = \sum_{s=0}^{k-1} \sum_{l} \mathbf{H}^s(i,l) r(l), \text{ for all } i \text{ and } k \geq 1.$$

Equivalently, we have the recursion

$$V^{k+1}(i) = r(i) + \sum_{j} \mathbf{H}(i,j) V^k(j),$$

with $V^0(i) = 0$, for all i. $V^k(i)$ represents the expected cumulative reward over k steps of the discrete time Markov reward model, with one-step probabilities $\mathbf{H}(i,j)$ and one-step rewards r, whenever the system is in state i.

Now, by virtue of the uniformization method, the steady state performability P of the continuous time Markov chain in steady state is given by

$$P = \sum_j p_j r(j) = \lim_{k \to \infty} \frac{1}{k} V^k(i), \qquad (6)$$

for arbitrary initial state i. Now, assume that for computational convenience we truncate the Markov chain by setting $q_{ij} = 0$, for all $i \in T$ and $j \notin T$, where $T \subset M$. In words that is, transitions from i to $j \notin T$ are transformed into dummy transitions from i into itself. Define the matrices $\widetilde{\mathbf{H}}^k$ ($k \geq 1$), the functions $\widetilde{V}^k$ ($k \geq 0$), and the steady state performability $\widetilde{P}$ for the truncated model as has been done above for the original model. The reward function r remains the same. The following result then enables us to conclude an a priori error bound on the truncation error. This result may seem technical and impractical to apply, however, as will be discussed, its conditions can frequently be verified relatively easy, also for complex models.

Error bound result. Assume that for some K and Δ, some initial state $l \in T$, some function Θ, for all

$i \in T$ and for all $k \geq 0$ we have:

$$\sum_{j \notin T} \mathbf{H}(i,j)|V^k(j) - V^k(i)| \leq K\Theta(i), \tag{7}$$

$$\text{and } \sum_j \mathbf{H}^k(l,j)\Theta(j) \leq \Delta. \tag{8}$$

We then have

$$|P - \widetilde{P}| \leq \Delta K. \tag{9}$$

For a proof of this result we refer to van Dijk [3, 4, 5].

Discussion. Roughly speaking, Condition (7) measures the effect of the total rate at which the truncation set T can be left by the difference $|V^k(j) - V^k(i)|$ of the cumulative reward, the so-called bias term. While V^k generally grows linearly in k, the bias terms can generally be bounded independently of k, and often also independently of the initial state l, say by a constant K. Frequently, the set of states T can only be left from states at or near the boundary between T and $M - T$. Thus, by taking

$$\Theta(i) = \begin{cases} 1, & \text{if } \sum_{j \notin T} \mathbf{H}(i,j) > 0, \\ 0, & \text{elsewhere,} \end{cases} \tag{10}$$

Condition (7) seems generally satisfied. Condition (8) then roughly requires the probability of being in states where the truncation takes place to be reasonably small. This condition is most naturally fulfilled as the truncation, i.e. the set T, is usually determined with a similar intuitive notion.

5 Related work

The problem of state spaces that are too large to handle is not new. A lot of other work has been going on in this field. In this section we briefly mention some of this work.

As already mentioned, van Dijk [3, 4, 5] has done numerous studies in the field of Markov chain truncation techniques. His work however is very general and not specifically related to the performability field.

More directly related to the performability field is the approach followed by Muntz *et al.* [15]. Instead of simply truncating the state space by excluding all states in which the number of simultaneous failures is larger than some threshold, they aggregate these states into one or more "macro" states, e.g. one macro state for all states with a fixed number of simultaneous failures. Upper bounds and lower bounds are then derived by adjusting the transition rates into and out of the macro states appropriately.

GSPN dependability modelling is addressed by Ibe *et al.* [11, 12]. The proposed approximation in [11] is based on hierarchical decomposition and combinatorial modelling, and is therefore not generally applicable.

In the Save manual [8], Goyal employs a state space truncation technique similar to the one discussed in Section 4 of this paper. He does however not justify these truncations by providing a theoretical framework. In newer versions of Save [6, 7] the method by Muntz *et al.* [15] seems to be implemented.

Finally, the work by Sanders and Meyer [17] deserves special attention. They use Stochastic Activity Networks (SANs), a GSPN-look-alike formalism, for the description of the performability models. By using lumping theorems they derive an underlying stochastic model from their SANs that is "just detailed enough" to exactly derive the desired measures. Important to note is that their so-called *reduced base model construction* technique is exact and that it does not require the total state space to be constructed before the lumping can take place, i.e. the strategy to come to the reduced base model is set *a priori*, based on the model structure and the desired measure.

6 Concluding remarks

In this paper we have addressed the problem of solving large performability models by using truncation techniques. We have proposed two truncation techniques that are not only intuitively appealing, but provably correct too. The main merits of the two truncation techniques are that they are very simple to apply, and that they yield reasonable approximations for models that are significantly reduced in size. The results obtained with the approximations can serve perfectly for "quick engineering" purposes. We have illustrated the truncation techniques with a number of examples and proven their correctness. We furthermore indicated how the proposed truncation techniques can be very easily implemented at the GSPN level of description. We finally discussed some related activities in this field.

Acknowledgements

The work presented in this paper forms part of a chapter of my Ph.D. thesis [9]. I would therefore like to thank my Ph.D. supervisor Ignas G. Niemegeers (University of Twente, Enschede), as well as Nico M. van Dijk (Free University, Amsterdam) for their many

comments which greatly improved that chapter. I would also like to thank the research group of Gianfranco Balbo (University of Torino) for making the GreatSPN software available. Without this, the practical work presented in this paper could not have been done.

References

[1] J. Bechta Dugan, K.S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems", *IEEE Tr. Comp.* 38(6), pp.775–787, 1989.

[2] G. Ciardo, J. Muppala, K.S. Trivedi, "SPNP: Stochastic Perti Net Package", *Proc. 3rd Int'l Workshop on Petri Nets and Performance Models*, IEEE Comp. Soc. Press, pp.142–151, 1989.

[3] N.M. van Dijk, "Analytic Error Bounds for Approximations of Queueing Networks with an Application to Alternate Routing", *J. Austral. Math. Soc. Ser. B* 31, pp.241–258, 1990.

[4] N.M. van Dijk, "Transient Error Bound Analysis for Continuous Time Markov Reward Structures", to be published in *Perform. Eval.*, 1991.

[5] N.M. van Dijk, "Truncation of Markov Chains with Applications to Queueing", to be published in *Op. Res.*, 1991.

[6] A. Goyal, S.S. Lavenberg, K.S. Trivedi, "Probabilistic Modelling of Computer System Availability", *Ann. Op. Res.* 8, pp.285–306, 1987.

[7] A. Goyal, S.S. Lavenberg, "Modelling and Analysis of Computer System Availability", *IBM J. Res. Dev.* 31(6), pp.651–664, 1987.

[8] A. Goyal, "System Availability Estimator – User's Manual", Version 2.0, IBM Yorktown Heights, NY, USA, 1987.

[9] B.R. Haverkort, *Performability Modelling Tools, Evaluation Techniques, and Applications*, Ph.D. thesis, University of Twente, 1990.

[10] B.R. Haverkort, I.G. Niemegeers, P. Veldhuyzen van Zanten, "DyQNtool—A Performability Modelling Tool Based on the Dynamic Queueing network Concept", in *Proc. 5th Int'l Conf. Modelling Tools and Performance Evaluation of Computer Systems*, Editor: G. Balbo, North-Holland, forthcoming 1991.

[11] O.C. Ibe, R.C. Howe, K.S. Trivedi, "Approximate Availability Analysis of VAXcluster Systems", *IEEE Trans. Rel.* 38(1), pp.146–152, 1989.

[12] O.C. Ibe, A. Sathaye, R.C. Howe, K.S. Trivedi, "Stochastic Petri Net Modelling of VAXcluster System Availability", *Proc. 3rd Int'l Workshop on Petri Nets and Performance Models*, IEEE Comp. Soc. Press, pp.112–121, 1989.

[13] J.F. Meyer, "Closed-Form Solutions of Performability", *IEEE Trans. Comp.* 31(7), pp.648–657, 1982.

[14] D.R. Miller, "Almost Sure Comparisons of Renewal Processes and Poisson Processes, with Application to Reliability Theory", *Math. Op. Res.* 4(4), pp.406–413, 1979.

[15] R.R. Muntz, E. de Souza e Silva, A. Goyal, "Bounding Availability of Repairable Computer Systems", *IEEE Trans. Comp.* 38(12), pp.1714–1723, 1989.

[16] J.K. Muppala, K.S. Trivedi, "Composite Performance and Availability Analysis Using a Hierarchy of Stochastic Reward Nets", in *Proc. 5th Int'l Conf. Modelling Tools and Performance Evaluation of Computer Systems*, Editor: G. Balbo, North-Holland, forthcoming 1991.

[17] W.H. Sanders, J.F. Meyer, "Reduced Base Model Construction for Stochastic Activity Networks", *IEEE J. Sel. Ar. Comm.*, pp.25–36, 1991.

[18] R.M. Smith, K.S. Trivedi, A.V. Ramesh, "Performability Analysis: Measures, an Algorithm and a Case Study", *IEEE Trans. Comp.* 37(4), pp.406–417, 1988.

[19] K.S. Trivedi, J.K. Muppala, S.P. Woolet, B.R. Haverkort, "Composite Performance and Dependability Analysis", to be published in *Perf. Eval.*, 1991.

[20] J. Walrand, *An Introduction to Queueing Networks*, Prentice Hall International, 1988.

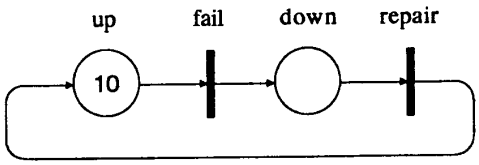

Figure 1: Simple GSPN dependability model

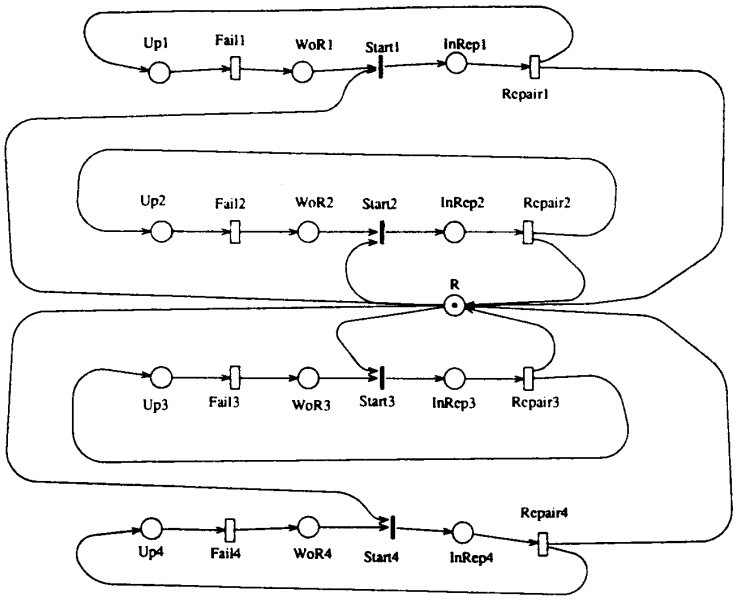

Figure 2: GSPN dependability model used for truncation experiments

f	P	$[\widetilde{P}_l, \widetilde{P}_u]$	$[\widetilde{P}_l, \widetilde{P}_u]$	$\widetilde{P}_*$
0.100	0.62228	[0.33831, 0.80110]	[0.51909, 0.70320]	0.60819
0.025	0.98937	[0.80582, 0.99147]	[0.95762, 0.98998]	0.95864
0.010	0.99930	[0.92054, 0.99935]	[0.99292, 0.99930]	0.99292
0.001	0.99991	[0.95964, 0.99991]	[0.99819, 0.99991]	0.99991

Table 1: Steady state performability by state space truncation for time-reversible models

truncation strategy	number of states	state coverage (%)	Performability measures P_1	P_2	P_3	P_4
T0	5159+1558	100.00	0.976025	0.999826	1.000000	0.996927

Table 2: Steady state performability measures for the untruncated model

truncation strategy	number of states	state coverage (%)	Performability approximations $\widetilde{P}_1$	$\widetilde{P}_2$	$\widetilde{P}_3$	$\widetilde{P}_4$
T1	33+14	0.640	0.976195	1.000000	1.000000	0.996946
T2	217+79	4.206	0.976026	0.999827	1.000000	0.996927
T3	769+254	14.906	0.976025	0.999826	1.000000	0.996927
T4	140+34	2.714	0.976024	0.999827	1.000000	0.996927
T5	272+68	5.272	0.976024	0.999826	1.000000	0.996927
T6	466+119	9.033	0.976024	0.999826	1.000000	0.996927
T7	1036+274	20.081	0.976025	0.999826	1.000000	0.996927

Table 3: Steady state performability approximations for various truncated models

model number	number of component classes C	number of components in classes $N_1, \cdots, N_C$	number of states
1	4	4, 12, 5, 3	5159
2	4	5, 13, 6, 4	10053
3	4	6, 14, 7, 5	17635
4	8	$2, \cdots, 2$	34993
5	5	4, 12, 5, 3, 5	38749
6	5	5, 13, 6, 4, 6	88005
7	6	4, 12, 5, 3, 5, 4	231181
8	6	5, 13, 6, 4, 6, 5	630925
9	7	5, 13, 6, 4, 2, 6, 8	3240469
10	9	$2, \cdots, 2$	21925618
11	10	$2, \cdots, 2$	50029671

Table 4: State space sizes for various component configurations

truncation strategy	number of states	unavailability U_{rep}	Performability approximations $\widetilde{P}_1$	$\widetilde{P}_2$	$\widetilde{P}_3$
S1	8+7	0.03661	0.9633391	1.000000	1.000000
S2	57+7	0.03795	0.962025	0.999735	0.999998
S3	252+35	0.03800	0.962002	0.999710	0.999998
S4	827+118	0.03800	0.962000	0.999708	0.999998
S5	2226+321	0.03800	0.962000	0.999708	0.999998
S6	5194+754	0.03800	0.962000	0.999708	0.999998

Table 5: Steady state performability approximations for a model that is too large to handle directly