Dependability and Performability Analysis*

Kishor S. Trivedi[1], Gianfranco Ciardo[2], Manish Malhotra[3], Robin A. Sahner[4],

[1] Department of Electrical Engineering, Duke University
[2] Department of Computer Science, College of William and Mary
[3] AT&T Bell Laboratories, Holmdel, NJ 07733
[4] Urbana, IL, 61801

Abstract. In this tutorial, we discuss several practical issues regarding specification and solution of dependability and performability models. We compare model types with and without rewards. Continuous-time Markov chains (CTMCs) are compared with (continuous-time) Markov reward models (MRMs) and generalized stochastic Petri nets (GSPNs) are compared with stochastic reward nets (SRNs). It is shown that reward-based models could lead to more concise model specification and solution of a variety of new measures. With respect to the solution of dependability and performability models, we identify three practical issues: largeness, stiffness, and non-exponentiality, and we discuss a variety of approaches to deal with them, including some of the latest research efforts.

1 Introduction

Dependability, performance, and performability evaluation techniques provide a useful method for understanding the dynamic behavior of a computer or communication system. To be useful, the evaluation should reflect important system characteristics such as fault-tolerance, automatic reconfiguration, and repair; contention for resources; concurrency and synchronization; deadlines imposed on the tasks; and graceful degradation. Furthermore, complexity of current-day systems and corresponding system evaluation should be explicitly addressed.

Traditional performance evaluation is concerned with contention for system resources. Performance evaluation of parallel and distributed systems also address concurrency and synchronization of tasks. Real-time system performance evaluation takes into account various hard and soft deadlines on task exection times.

Reliability, availability, safety, and related measures are collectively known as dependability. Dependability evaluation encompasses fault-tolerance, reconfiguration, and repair aspects of system behavior. More recently, interest in combining performance and dependability evaluation has grown. Such performability

* This research was partially supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while the first two authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681.

evaluation considers the graceful degradation of the system in addition to the dependability aspects.

While measurement is an attractive option for assessing an existing system or a prototype, it is not a feasible option during the system design and implementation phases. Model-based evaluation has proven to be an attractive alternative in these cases. A model is an abstraction of a system that includes sufficient detail to facilitate an understanding of system behavior. Several types of models are currently used in practice. The most appropriate type of model depends upon the complexity of the system, the questions to be studied, the accuracy required, and the resources available for the study.

Discrete-event simulation is the most commonly used modeling technique in practice but it tends to be relatively expensive. Analytical modeling provides a cost-effective alternative to simulation for studying the performance and dependability of computer and communication systems. Due to recent developments in model generation and solution techniques and automated tools, large and realistic models can be developed and studied. In this tutorial we concentrate on such analytic models. The rest of this tutorial is organized as follows. In the next section, we present an overview of various approaches to dependability and performance modeling. In Section 3, we show how performability analysis can be carried out using MRMs. We also show how dependability measures can be obtained via performability analysis using special reward rate assignment.

In Section 4, we compare GSPNs and stochastic reward nets. In Section 5, we discuss in detail some practical issues in solving dependability and performability models: largeness, stiffness, and non-exponentiality.

2 Approaches to Modeling

2.1 Dependability Modeling

Reliability block diagrams, fault trees, and reliability graphs are commonly used to study the dependability of systems [59]. Although these models are concise and have efficient solution methods, they cannot represent dependencies among components [56] as easily as CTMC models can [21, 23].

We begin by considering a fault-tolerant, multi-processor computer with multiple, shared memory modules. The system is able to detect a processor or memory module failure and reconfigure itself to continue operation without the failed component. The system can operate with just one processor and one memory module.

Our first model of this system is the reliability block diagram in Figure 1. We could attach to each component the probability of having failed by a particular time. In a more general parameterization, a failure time distribution function, rather than a probability value, can be attached to each component. For example, one can assign the exponential distribution $F_p(t) = 1 - e^{-\lambda_p t}$ to processors and $F_m(t) = 1 - e^{-\lambda_m t}$ to memories. We can request the system failure time distribution as a function of the time variable t. For a system with

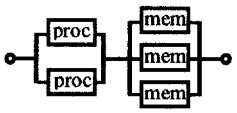


Fig. 1. A reliability block diagram model.

two processors and three memory modules,

$$F_{sys}(t) = 1 - (1 - (1 - e^{-\lambda_p t})^2)) \cdot (1 - (1 - e^{-\lambda_m t})^3)) \ .$$

We can also ask for the mean time to system failure,

$$MTTF_{sys} = \int_0^\infty (1 - F_{sys}(t)dt$$

$$= \frac{6}{\lambda_p + \lambda_m} - \frac{3}{2\lambda_p + \lambda_m} - \frac{6}{\lambda_p + 2\lambda_m} + \frac{3}{2\lambda_p + 2\lambda_m} + \frac{2}{\lambda_p + 3\lambda_m} - \frac{1}{2\lambda_p + 3\lambda_m} \ .$$

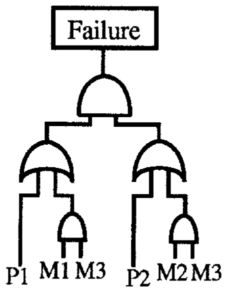


Fig. 2. A fault tree model.

Now suppose we want to investigate a different computer design where the two processors have fast private memory modules and the system has slower, shared memory modules. We assume that the system operates as long as there is at least one operational processor with access to either a private or shared memory. We cannot model this system with a block diagram, because there is no way to model how the shared memories are connected to all processors while private memories are connected to particular processors. So, we turn to a fault tree model, shown for two processors and three memory modules in Figure 2. We could also use a reliability graph, where time-to-failure distributions are assigned to the edges. The system is operational as long as there is a path from source (src) to sink. In this particular model (Figure 3), processor failures happen along the edges labeled *P1* and *P2* and memory failures happen along the edges *M1*, *M2*, and *M3*. The edges *I1* and *I2* do not represent system components; they represent the structure of the system (the sharing of *M3*). We assign the "infinite" distribution, defined by $I(t) = 0$, to them. There is a path from source to sink if *P1* and *M1* are up or if *P1* and *M3* are up, and similarly for paths involving *P2*. Analysis of the reliability graph results in the same failure time distribution as the fault tree analysis.

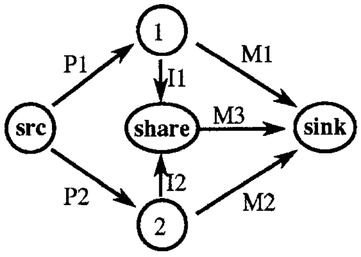


Fig. 3. A reliability graph model.

Now we extend our models to take into account repair or replacement of parts. We calculate the "availability" of the system, the (transient or steady-state) probability that the system is functioning. We examine the all-shared-memory system and look at three repair strategies:

1. There are enough repair resources to repair all components at the same time, if necessary.
2. There are two repair facilities, one for processors and one for memory modules, each able to handle one component at a time.
3. There is one repair facility, able to handle one component at a time. Processor repair has preemptive priority over memory repair.

For the first strategy, the state of the components (either up or down) are mutually independent, since the failure and repair of each component does not depend on that of any other component. Because of this independence, we can use the block diagram used to model reliability (Figure 1) to model availability as well. Instead of assigning to each component the time-to-failure distribution, we use the transient unavailability. If the i^{th} component has exponentially distributed failure behavior with rate λ_i and repair is also exponentially distributed with rate μ_i, its unavailability at time t is

$$U_i(t) = \frac{\lambda_i}{\lambda_i + \mu_i} - \frac{\lambda_i}{\lambda_i + \mu_i} e^{-(\lambda_i+\mu_i)t} \tag{1}$$

and the steady-state unavailability is given by

$$\lim_{t\to\infty} U_i(t) = \frac{\lambda_i}{\lambda_i + \mu_i}$$

These expressions can be derived by solving the two-state (up/down) CTMC for a component [62].

If we analyze the reliability block diagram of Figure 1 with the assignment of distribution functions of Equation 1 to the components, the resulting function is the system unavailability at time t, $U_{sys}(t)$, and the "mass at infinity" $(1 - \lim_{t\to\infty} U_{sys}(t))$ is the steady-state system availability.

To deal with the second and third repair strategies, we can no longer use the block diagram model. The block diagram assumes that all components are statistically independent, but, if components share repair facilities, the failure and repair behavior of one component is dependent on the state of all components.

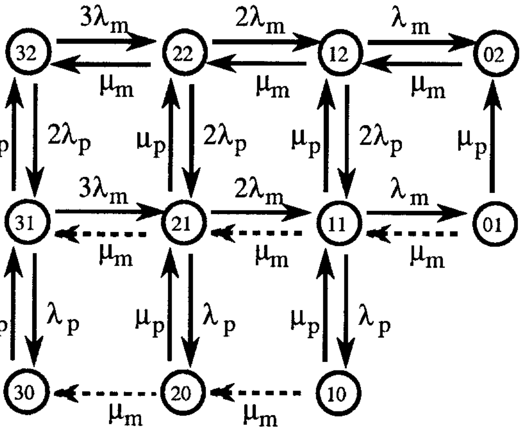


Fig. 4. A CTMC model.

If the failure and repair distributions are exponential, we can use a CTMC model. Consider the CTMC in Figure 4. State *mp* represents the system when *m* memory units and *p* processors are functional. The model with all of the solid and dashed-line transitions is for the second repair strategy (one repair facility for processors and one for memories). The model for the third strategy (only one repair facility giving priority to the processors) is obtained by excluding the dashed lines, since no memory is repaired while there are failed processors.

We note that we could have used a CTMC for the first repair strategy as well. We would have assigned different transition rates to the repair transitions to reflect the fact that more than one component can be repaired at a time. As an example, the rate for the transition from *02* to *12* would be $3 * \mu_m$ rather than μ_m. The block diagram model, though, is both easier to construct and more efficient to analyze.

Before leaving the subject of unavailability, we illustrate the use of one more model type, the GSPN. For a discussion of this model type, the reader is referred to [1]. Modeling the availability of this system with a GSPN does more than just give us another validity check. It allows us to find the unavailability for a system with any number of processors and memories without having to construct a separate model for each number of components. The GSPN in Figure 5 is a model of the system in which there is one repair facility to be shared for all components.

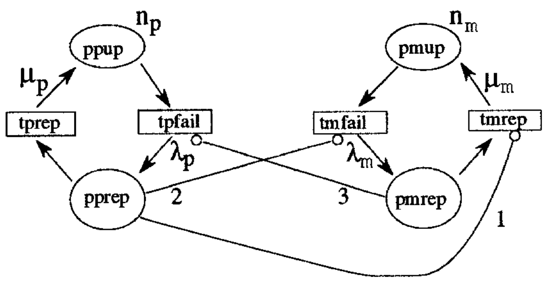


Fig. 5. A GSPN model.

There is a token for each processor and each memory. Initially, there are n_p tokens in the place *ppup* (place: processors up) and n_m tokens in the place *pmup* (place: memories up). When a processor fails, its token moves from place *ppup* through transition *tpfail* (transition: processor fails) to place *pprep* (place: processor waiting for repair). Processor repair is represented by a token moving from place *pprep* through transition *tprep* to place *ppup*. The inhibitor arcs from *pprep* to *tmfail* and *pmrep* to *tpfail* reflect the assumption that if the system has already failed because all processors or all memories have failed, the remaining working components do not fail while they are not running. This aspect of the system was modeled only implicitly in the CTMC model, by the absence of failure transitions from the places with either no operating processors or no operating memory modules. The inhibitor arc from *pprep* to *tmrep* is the one that represents our assumption that there is only one repair facility; if there are any failed processors, there can be no memory repair.

We can verify that analyzing this GSPN with $n_p = 2$ and $n_m = 3$ gives the same result for system steady-state unavailability as the CTMC model. We note that the GSPN, although a more efficient specification, is no more efficient to analyze than the CTMC, since analysis of a GSPN involves translating the GSPN into a CTMC. However, dependability modeling with GSPN tends to be clumsy [41]. Stochastic reward nets remove this restriction from GSPN models. We elaborate more on this in Section 4.

2.2 System Performance Models

In this section, we look at aspects of system performance, including performance of gracefully degraded systems. In the performance domain, task precedence graphs [31, 55] can be used to model the performance of concurrent programs with unlimited resources. Product form queueing networks [35, 36], on the other hand, can represent contention for resources. However they cannot model concurrency within a job, synchronization, or server failures, since these violate the product form assumptions.

We consider the same two system architectures as in Section 2.1: the first containing two processors and three shared memory modules and the second containing two processors, each with a private memory module, and one shared memory module.

To capture the effects of contention for the processor and memory resources, we use queueing network models. We assume that the memory modules are servers in the sense that they queue requests and perform block transfers. To set up a realistic queueing model, we would have to take into account the proposed operating system design, especially the scheduling aspects, and we would need some kind of expected workload characterization. For the sake of illustration, we use the closed queueing network models shown in Figures 6 and 7.

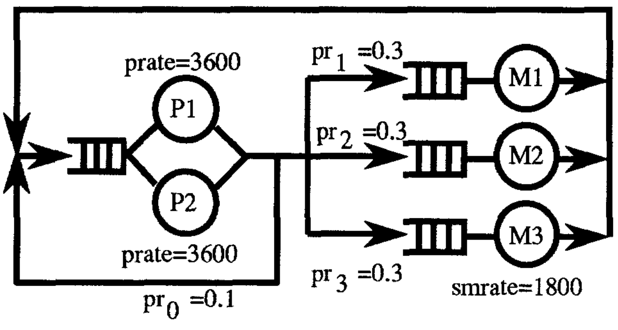


Fig. 6. A product form queueing network for the system with three shared memories.

The network in Figure 6 is for the design containing two processors and three shared memory modules. We model the two processors by a multiple-server station. That is, jobs wait in a single queue and enter whichever server becomes

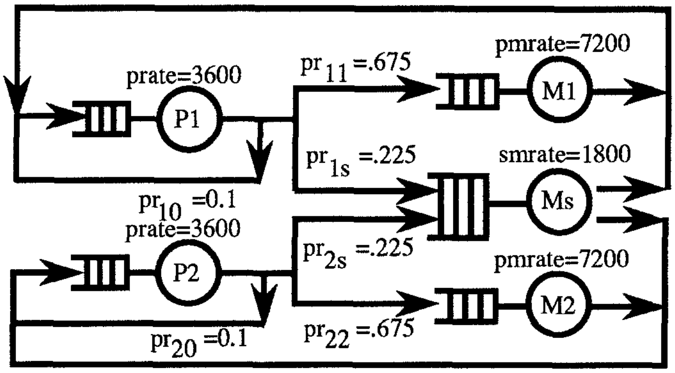


Fig. 7. A product form queueing network for the system with one shared memory and two local memories.

free. When a job wants to access the memory, it requires memory module *Mi* with probability pr_i. After some visits to the processor, a job finishes: pr_0 is the probability that a job is finished when it leaves one of the processors. As is usual for closed queueing networks, the assumption is that each finished job is replaced by a statistically identical new job.

The network in Figure 7 is for the design containing two private memory modules. For this system, we assume that jobs are targeted to particular processors. This is reasonable, since, once a job starts on a processor, we want it to continue where it has access to that processor's private memory. We carry out this assumption by making the queueing network a "multiple-chain" queueing network, in this case having two "chains", or classes of jobs. Jobs in the first class go from *P1* to either *M1* or *Ms* and back to *P1* and jobs in the second class go from *P2* to either *M2* or *Ms* and back to *P2*.

As expected, the system with private memories provides higher system throughput as opposed to that for the shared-memory system.

To model the systems when one memory has failed, we remove the server *M1* (and its queue) from each of the models and adjust the probabilities pr_i and pr_{ij} appropriately.

Queueing models are able to capture the effects of resource contention, but measures related to the total number of jobs serviced do not capture the performance of the system as seen by a single parallel program: series-parallel acyclic graph models [55] can be used for this purpose.

Also CTMCs provide a useful framework to model system performance, but a detailed CTMC model is often large and complex and its construction is an error-prone process. Hence there is a need for a higher-level model-type having an underlying CTMC, which is then automatically generated from it. Some attempts in the specific instance of dependability modeling have resulted in useful packages like SAVE [23], for availability modeling, which uses a block diagram input, and HARP [21], for reliability modeling, which uses a fault-tree input. A suitable interface is necessary for a more general modeling environment. GSPNs [1] and SRNs [13] provide an excellent interface for detailed performance modeling of complex systems.

The advent of fault-tolerant computing has resulted in the design of machines which continue to function even in the presence of failures, albeit at a reduced level of performance. Pure reliability or performance models of such systems do not capture the whole picture. This has prompted researchers to consider the combined evaluation of performance and reliability [44, 63]. The CTMC is extended by associating rewards with its states to obtain a "Markov reward process", or "Markov reward model" (MRM). This process not only facilitates modeling of performance and reliability but also the combined evaluation of performance and reliability. Since this paper considers the automatic generation of the CTMC from the GSPN description of the model, the reward structure must also be defined in terms of the GSPN entities. Consequently the GSPN description is modified to obtain "stochastic reward nets" [13] which can be automatically transformed to obtain the underlying MRM.

3 CTMCs versus MRMs

CTMCs have been traditionally used to model dependability. MRMs [25] are CTMCs in which reward rates may be associated with states of the CTMC (*rate-type* rewards) or with transitions of the CTMC (*impulse-type* rewards) or both. We consider MRMs with rate-type rewards. MRMs have been successfully used for performability analysis [44, 63] according to the following methodology. Initially, a dependability model (also known as *structural model*) of the system is constructed. Assuming the dependability model is state-space type (such as a CTMC), a performance measure is obtained (possibly by solving a performance model) for each state of the dependability model. This performance measure becomes the reward rate of that state in the dependability model. With the reward-rate assignment, the dependability model becomes an MRM which may then be solved for various performability measures. There is an approximation involved in this decomposition of performance and dependability models: the system is assumed to have attained (quasi-)steady-state in each state of the dependability model, so that the reward rate for each state of the reliability model is a steady-state performance measure. Transient or steady-state analysis of the dependability model with rewards is then carried out. The justification for this decomposition lies in the fact that the performance activities are much faster than the dependability events.

CTMCs can also be used for performability analysis if a monolithic model is constructed which combines both the dependability and performance model of the system. However, the state-space of this model is approximately the cross-product of state-spaces of the dependability and performance models. In addition, this monolithic model is stiff because of extreme disparity between the transition rates (job arrival rates could be 10^9 times or more than the fault occurrence rates). One may argue that this approach is more accurate than the MRM approach since no approximation is involved. However, this gain in accuracy may well be negated due to the computational problems posed by largeness and stiffness of the monolithic model. We focus more on these two problems, largeness and stiffness, in later sections. The MRM approach has another significant advantage. No assumptions are made about how the reward rates are obtained. The reward rates may be obtained by simulation, by solving a queuing network, or by solving a semi-Markov process (SMP), etc.

It is easy to see that CTMCs are special cases of MRMs and therefore dependability analysis becomes a special case of performability analysis. In this section, we briefly show how various dependability measures can be analyzed as performability measures when the MRM has a special reward-rate assignment. Let $\{\Theta(t), t \geq 0\}$ be an MRM with state space Ψ and constant reward rate r_i associated with each state i of the CTMC. If the MRM spends τ_i units of time in state i, then $r_i \tau_i$ is the reward accumulated during this sojourn. Let $\mathbf{Q}$ be the generator matrix and $\mathbf{P}(t)$ be the state probability vector of the MRM. Here $P_i(t)$ denotes the transient probability of the MRM being in state i at time t. The transient behavior of this MRM is given by the Kolmogorov differential equation:

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{P}(t)\mathbf{Q} \ , \tag{2}$$

given the initial state probability vector $\mathbf{P}(0)$. The steady-state probability vector π (assuming that it exists and is unique) is obtained by setting the l.h.s. in Equation 2 to zero:

$$\pi\mathbf{Q} = 0 \ , \ \sum_{i \in \Psi} \pi_i = 1 \ ,$$

The cumulative state probability vector of the MRM is defined as $\mathbf{L}(t) = \int_0^t \mathbf{P}(x)dx$, where $L_i(t)$ denotes the expected total time spent by the MRM in state i during the interval $[0, t)$. To compute $\mathbf{L}(t)$, we integrate Equation 2:

$$\frac{d\mathbf{L}(t)}{dt} = \mathbf{L}(t)\mathbf{Q} + \mathbf{P}(0) \ .$$

The reward rate at time t for the MRM is given by $\Upsilon(t) = r_{\Theta(t)}$. The accumulated reward over the interval $[0, t)$ is given by:

$$\Phi(t) = \int_0^t \Upsilon(x)dx = \int_0^t r_{\Theta(x)}dx \ .$$

The expected reward rate at time t of the MRM is:

$$E[\Upsilon(t)] = \sum_{i \in \Psi} r_i P_i(t) \ .$$

The expected reward rate in steady-state for the MRM is:

$$E[\Upsilon_{ss}] = \sum_{i \in \Psi} r_i \pi_i \quad .$$

To compute availability measures, the state-space of the MRM is partitioned into two: a set of system-up states, with reward rate 1, and a set of system-down states, with reward rate 0. We term this a *0-1 reward assignment*. The *transient availability* of the system is given by $E[\Upsilon(t)]$ and *steady-state availability* is given by $E[\Upsilon_{ss}]$.

The expected accumulated reward over the interval $[0,t)$ is:

$$E[\Phi(t)] = \sum_{i \in \Psi} r_i L_i(t) \quad .$$

The expected time-averaged reward rate over the interval $[0,t)$ is given by $\sum_i r_i L_i(t)/t$. In an availability model with 0-1 reward assignment, the *total uptime* of the system over the interval $[0,t)$ is $E[\Phi(t)]$. *Interval availability* is the proportion of time a system is up in a given interval of time and it is given by $E[\Phi(t)]/t$ for the interval $[0,t)$.

For MRMs with absorbing states, the state-space Ψ is partitioned into two: Ψ_A (set of absorbing states) and Ψ_T (set of transient states). Let $\mathbf{Q}_T$ be the submatrix of $\mathbf{Q}$ corresponding to the transitions between transient states. The mean time spent by the MRM in state $i \in \Psi_T$ before absorption is given by $\tau_i = \int_0^\infty P_i(x)dx$, which is obtained by integrating Equation 2 from 0 to ∞:

$$\tau \mathbf{Q}_T + \mathbf{P}_T(0) = 0 \quad .$$

The mean time to absorption is given by:

$$MTTA = \sum_{i \in \Psi_T} \tau_i \quad .$$

To compute reliability measures, all the system-down states are made absorbing states (transitions leaving from them are deleted). The same 0-1 reward assignment is used. The *reliability* is given by $E[\Upsilon(t)]$. The *lifetime* (similar to total uptime) [20] of the system over the interval $[0,t)$ is $E[\Phi(t)]$. The expected accumulated reward until absorption is:

$$E[\Phi(\infty)] = \sum_{i \in \Psi_T} r_i \tau_i \quad .$$

and the *mean time to failure* (MTTF) of the system is $E[\Phi(\infty)]$.

The distribution of the reward rate at time t, $\Upsilon(t)$, is computed as:

$$P[\Upsilon(t) \le \psi] = \sum_{r_i \le \psi, i \in \Psi} P_i(t) \quad .$$

The distribution of accumulated reward until absorption or a finite period can also be computed. If the time to accumulate a given reward r is $\Gamma(r)$, then the distribution of $\Gamma(r)$ is known once the distribution of accumulated reward is known [32]:

$$P[\Gamma(r) \leq t] = 1 - P[\Phi(t) < r] \ . \tag{3}$$

For instance, the distribution of time to complete a job that requires r units of processing time on a system which is modeled by an MRM can be computed in this fashion.

From the above discussion, it is clear that dependability analysis can be carried out using MRMs with special reward rate assignment to various system states. This analysis can also be carried out using CTMCs (without rewards) in an equally efficient manner. However, performability analysis, which can be easily carried out using MRMs, becomes cumbersome if rewards are not used.

4 SPNs versus SRNs

CTMCs modeling real systems tend to be large, sometimes with hundreds of thousands states. A higher-level specification mechanism is thus needed for the concise description of the model and the automatic conversion into a CTMC. Stochastic Petri nets (SPNs) provide such a mechanism. Molloy [48] used SPNs for performance analysis and showed that they are isomorphic to CTMCs. Since then, several extensions have been made to SPNs. Some of these extensions have enhanced the flexibility of use and allowed for even more concise description of performance and reliability models. Some other extensions have enhanced the modeling power by allowing for non-exponential distributions (see Section 5.3).

In this section, we compare SPNs with and without rewards. Specifically, we compare SRNs as defined by Ciardo et al. [13] and GSPNs as defined by Ajmone-Marsan et al. [2]. SRNs are an extension of GSPNs, since they include all the features of GSPNs and add more features. There are several structural extensions such as guards (earlier known as enabling functions), priorities with timed transitions, marking-dependent arc cardinalities, and halting condition. Besides the structural extensions, a reward rate function associates a reward rate with each reachable marking. GSPNs and SRNs have been shown to be isomorphic to CTMCs and MRMs respectively. However, we show in this section that SRNs allow a much more concise description of system behavior than GSPNs. This is particularly true for dependability models. Furthermore, certain reward-based measures as described in Section 3 can be computed using SRNs but cannot be computed using GSPNs.

To compare GSPNs and SRNs, we present an example. Consider a simple network between *src* and *sink* nodes consisting of three links (Figure 8). The network is operational as long as link A and at least one of the links B or C is operational. Assuming that each link has its independent repairperson, the availability of the network can be modeled by the GSPN shown in Figure 9. A token in places pA, pB, and pC respectively indicates that links A, B, and C are operational. A token in place pF implies that the network is failed. A token in place pR implies that, due to repair of one or more links, the component is ready to be operational again. The firing of transition tR removes the token from pF, signifying that the network is operational. The steady-state (transient) probability of a token being in place pF gives the steady-state (transient) unavailability of the network.

The availability of this network can also be modeled by an SRN as shown in Figure 10. The reward rate function is as shown in the table. The expected value of reward rate r in steady-state (or at time t) gives the steady-state (transient) availability of the network. Let us now compare the GSPN and SRN models. A GSPN model requires a mesh of immediate transitions, places, and inhibitor arcs to capture the operational dependence of the network on the links. Part of this mesh captures the dependence such as the subsystem of links B and C fails only when both B <u>and</u> C have failed. The other part of the mesh captures the impact of repairs of links which reflect complementary conditions, such as removal of a token from place pBC as soon as either B <u>or</u> C is repaired. As the systems grow in complexity, this mesh becomes very complex and unwieldy. On the other hand, an SRN captures the operational dependence of the network on links by reward rate function. This results in a simpler and more manageable net.

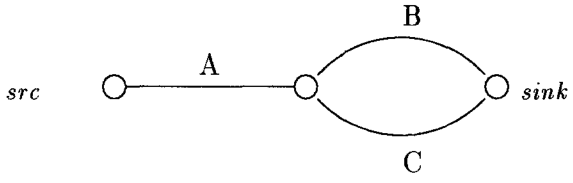


Fig. 8. A simple network

5 Computational Problems

In modeling practice, it is often the case that no single model is adequate to solve a problem. Different parts or levels of detail in a system may require different modeling techniques. In cases where a single model type can be used, it may be too large (a problem both for specification and analysis) or intractable ("stiff" or ill-conditioned). Three main difficulties in analytic models include largeness, stiffness, and the need to model non-exponential distributions. We explore these topics in the following subsections.

5.1 Largeness

The problem of model largeness can be handled in two ways: it can be avoided or it can be tolerated.

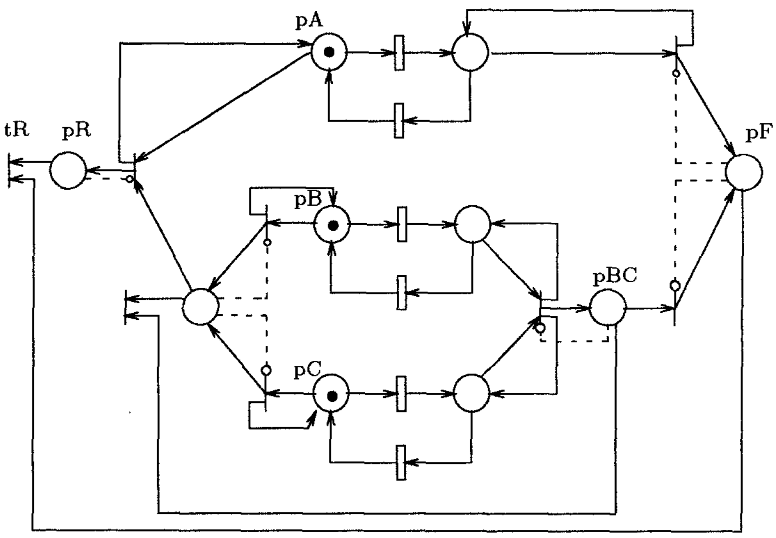
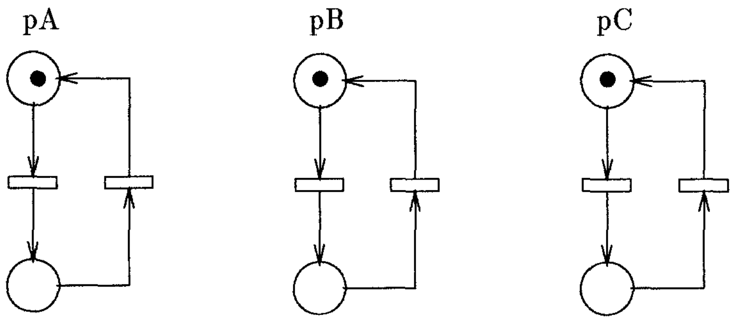


Fig. 9. GSPN availability model of the network



Name	Boolean Function
$bool_A$	$(\#tokens(pA) == 1)$
$bool_{BC}$	$(\#tokens(pB) == 1) \vee (\#tokens(pC) == 1)$
$bool_{NW}$	$bool_A \wedge bool_{BC}$

Reward Rate Function
if $(bool_{NW} == 1)$ then r = 1 else r = 0

Fig. 10. SRN availability model of the network

Largeness Tolerance For the sake of simplicity we assume that the underlying model is a CTMC or an MRM. If we are prepared to store and solve the matrix of a large model, we should start with a concise description of the system model and provide for the automated generation and the solution of the underlying state space. A number of approaches have evolved for such specifications. Haverkort and Trivedi [24] summarize these approaches. They present seven different classes of specification techniques: Stochastic Petri nets and their variants, Communicating processes, Queueing networks, Specialized languages, Fault-trees, Production rule systems, and Hybrid techniques. We refer the reader to the cited paper for further details.

Largeness avoidance If the size of the underlying CTMC (or MRM) is so large as to preclude generation and storage, we must resort to approximations that avoid the large underlying model. State truncation, lumping, decomposition and fluid models constitute the types of approximations that have been utilized. We discuss these four approaches below.

Truncation For many practical systems, the exact number of structural states in a corresponding model might be extremely large, or even infinite. State-space based approaches, then, cannot be applied directly to the model. In many cases, though, the system spends most of the time in a small subset of the entire state space; most states have an extremely small probability.

This is particularly true of highly reliable systems: if a system has K components, and if each component fails with a very small rate (as is normally the case), states with more than a handful of failed components are rarely reached. Indeed, it is common practice in reliability modeling to stop the state-space exploration after $k \ll K$ failures, with the implicit assumption that states with $k+1$ or more failed components have negligible probability. This is just one example of state truncation.

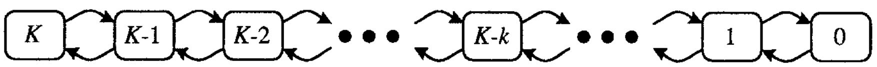


Fig. 11. A typical model that can be truncated.

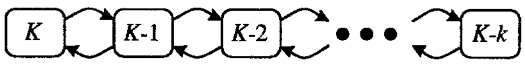


Fig. 12. Strict truncation.

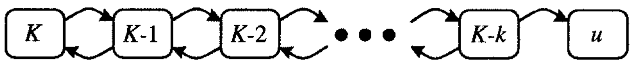


Fig. 13. Aggregation truncation.

As an example, consider a K-processor system, where nodes fail and are repaired with rate λ and μ, respectively. We want to compute the expected cumulative computational capacity during the time interval $[0,t)$, $C(t)$, that is, the expected number of non-failed processors as a function of time, integrated between 0 and t. If the state is characterized by the number of working processors, the model corresponds to a birth-death process with state space $\{K, K-1, \ldots 0\}$ (Figure 11). If the processors have different failure and repair behaviors, the identity of the failed processors must be recorded in the state and the size of the state space grows, dramatically, from $K+1$ to 2^K.

Formally, given a reachability graph $(\mathcal{S},\mathcal{A})$, a state truncation results in a truncated reachability graph $(\mathcal{S}',\mathcal{A}')$.

If $(\mathcal{S}',\mathcal{A}')$ is a subgraph of $(\mathcal{S},\mathcal{A})$, the exact state-space exploration algorithm, or the model, is simply modified to ignore certain arcs which lead to states in $\mathcal{S} \setminus \mathcal{S}'$. In our example, we can prevent a $k+1$-th failure in a state which already has k failed components. We call this case "strict truncation" (Figure 12).

Alternatively, $(\mathcal{S}',\mathcal{A}')$ might be composed by a subgraph of $(\mathcal{S},\mathcal{A})$, augmented with one or more states and arcs. In our example, we might add a new state u (for unknown), and an arc from each state with k failed components to u, corresponding to further failures of the non-failed components. Strictly speaking, this is more an "aggregation", so we call this approach an "aggregation truncation" (Figure 13).

The two approaches often allow us to obtain upper and lower bounds on the measure of interest. In our example, we can solve the two CTMCs of Figures 12 and 13, obtaining two transient probability vectors:

$$\pi^s(t) = [\pi^s_K(t), \ldots \pi^s_{K-k}(t)]$$

and

$$\pi^a(t) = [\pi^a_K(t), \ldots \pi^a_{K-k}(t), \pi^a_u(t)]$$

respectively. If we associate the reward rates

$$\rho^s = [\rho^s_K = K, \ldots \rho^s_{K-k} = K-k]$$

and

$$\rho^a = [\rho^a_K = K, \ldots \rho^a_{K-k} = K-k, \rho^a_u = 0]$$

with the states of the two CTMCs, we obtain the inequalities

$$C^s(t) = \sum_{i\in\{K,\ldots K-k\}} \rho^s_i \pi^s_i \geq C(t) \geq \sum_{i\in\{K,\ldots K-k,u\}} \rho^a_i \pi^a_i \geq= C^a(t)$$

If we are interested in the expected instantaneous computational capacity in steady state, c, that is, the expected number of non-failed processors in the long run, the CTMC in Figure 12 still offers an upper bound, but the one in Figure 13 is of no use, since state u has probability one in steady state, which would simply result in the trivial lower bound 0 for c. In any case, our ability to obtain useful bounds is normally tied to our a priori knowledge of aspects of the CTMC structure and values of the reward rates. In our example, we can prove that $C^s(t)$ is an upper bound on $C(t)$ because we know that

- Removing the set of states $\{K-k-1, \ldots 0\}$ does not decrease the probability of any of the states in $\{K, \ldots K-k\}$
- The maximum reward rates of the states in $\{K-k-1, \ldots 0\}$ is not larger than the minimum reward rates of the states in $\{K, \ldots K-k\}$.

and we can prove that $C^a(t)$ is a lower bound on $C(t)$ because we know that

- Aggregating the set of states $\{K-k-1, \ldots 0\}$ into a single absorbing state u does not increase the probability of any of the states in $\{K, \ldots K-k\}$
- The minimum reward rates of the states in $\{K-k-1, \ldots 0\}$ is not smaller than 0, the reward rate of states u.

For steady state analysis, more sophisticated arguments based on [17] can be used [49]. We conclude by observing that simulation is, in a probabilistic sense, a form of automatic truncation, since the most likely states are visited frequently while unlikely states may not be visited at all.

Lumping Most complex systems (models) consist of a large set of systems (submodels), many of them of the same type. The state of the system is then obtained by composing the state of each subsystem. When performing state-space exploration, though, there are simplifications which might lead to a smaller state space while still allowing an exact solution. For example, in our system with K processors, we could model each of them as an independent subsystem which can be in one of two states, *up* or *down*. The entire system can then be viewed as composed of K such subsystems, thus having 2^K states. This approach is wasteful, though, since it is not necessary to distinguish between processors, if they all have the same failure and repair behavior. Rather, we can represent the state of the system as the number of subsystems in each state (*up* or *down*, but, since the total number of processors is known, we can simply remember the number of *up* processors).

This application of lumping [30, 53] is indeed so natural that we used it in conjunction with truncation, without even justifying its adoption. In real systems, though, the reachability graph of a subsystem might be quite complex. The general algorithm to obtain the lumped state space for a system consisting of K independent subsystems can be easily expressed making use of SPNs [13] (see also [29] for an example of use of this algorithm):

1. Generate the reachability graph for a single subsystem. Markings and arcs are labeled with the number of tokens in each place and the name of the corresponding transition, respectively.

2. Transform the reachability graph into a SPN: for each marking i, add a place p_i, initially empty; for each arc from state i labeled by transition t, add a transition t_i with marking-dependent rate equal $\#(p_i)$ times the rate of t_i in marking i for a single subsystem, an input arc from p_i to t_i, and an output arc from t_i to p_j ($\#(p_i)$ is the number of tokens in place p_i).
3. Set the initial marking of the SPN: for each subsystem, if its initial state is i, add a token in p_i. Note that the subsystems can start in a different initial state without affecting the correctness of lumping.
4. Generate the CTMC underlying this SPN.

Figure 14 shows the application of the algorithm to a system composed of K dual-redundant subsystems, where repair is initiated only when both units have failed. Each subsystem is described by a SPN whose reachability graph has four markings. If no lumping is applied, the total number of states is 4^K. The application of our algorithm, instead, results in a SPN with $(K+3)(K+2)(K+1)/6$ states.

In general, if there are K subsystems with N states each, the size of state space with and without lumping is

$$N^K = \underbrace{N \times \cdots \times N}_{K \text{ terms}} \quad \text{vs.} \quad \binom{N+K-1}{K} = \underbrace{\frac{N+K-1}{K} \times \cdots \times \frac{N+1}{2} \frac{N}{1}}_{K \text{ terms}}$$

Each of the K terms in the second case is smaller than N, with the exception of the last one, which is N, so this approach is always guaranteed to reduce the size of the state space. The reduction is particularly sizable when N is small and K is large: for example, when $N = 2$ we have 2^K vs. $K + 1$.

In practice, the submodels have some interaction, so independence does not hold. If the interaction is limited to a "rate dependence" [14] where the transition rates in a subsystem depend on the number of subsystems in certain states, but not on their identity, the algorithm can still be applied: only a different specification of the firing rates for the resulting SPN is needed. In our example, the repairperson could be a shared resource, so the rate of transition *repair* in each subsystem could be $\lambda/f^{1.1}$, where f is the total number of subsystems being repaired, and the exponent 1.1 models the inherent inefficiency due to resource sharing. The rate of transition $repair_{1010}$ in the resulting SPN should then be specified as $\lambda/\#(p_{1010})^{1.1}$, where $\#(p_{1010})$ indicates the number of tokens in p_{1010} or, in other words, f.

Other types of dependence are structural: often, tokens might have to move from a submodel to another portion of the global model. With some care, lumping might still be possible [57].

Composition In this approach, the overall model is composed of a set of submodels. Construction and generation of a large model is avoided and the solution is obtained by interactions among the submodels. Interactions imply exchange of information between the submodels. Reward based performability analysis [44, 63] is an example of composition of reliability and performance models. The

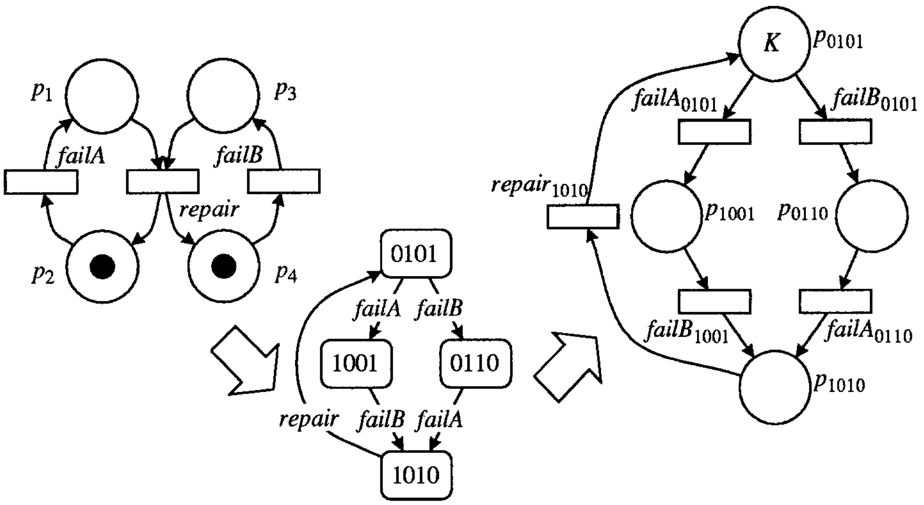


Fig. 14. Lumping of a model.

performance submodel is solved and its results are passed as reward rates to the reliability submodel. In general, quantities such as probability distributions, mean, variance, or numerical values of reliability and availability are exchanged among submodels.

Other examples of composition include flow-equivalent server approximation introduced by Chandy et al [10], behavioral decomposition used in the software tool HARP [21], composition of GSPNs and queuing networks proposed by Balbo et al [3], and hybrid hierarchical composition employed in the software tool SHARPE [56]. These approaches can be classified as ***hierarchical composition*** techniques. Hierarchical composition approaches differ not only in the way the model is constructed but also in the way the model is solved. The set of submodels can be solved iteratively using a fixed-point iteration scheme (a cyclic dependence exists among the submodels) [12, 14, 47, 61] or in a non-iterative fashion (a strict hierarchy exists among the submodels) [43, 56]. For a unified view of these seemingly different approaches to hierarchical composition, refer to [42].

Fluid Models As the number of tokens in a place or the number of jobs in a queue becomes large, the size of the underlying CTMC grows. It may be possible to approximate the number of tokens in the place, or the number of jobs in the queue, as a non-negative real number. It is then possible to write the differential equations for the dynamic behavior of the model and, in some cases, provide solution. Mitra has developed models along these lines [46]. More recently, Kulkarni and Trivedi have proposed fluid stochastic Petri nets (FSPNs) [33].

5.2 Stiffness

CTMC stiffness is a computational problem which adversely affects the stability, accuracy, and efficiency of a numerical solution method unless that method has been specially designed to handle it. CTMC stiffness is caused by extreme disparity between transition rates. In a reliability model, repair rates could be 10^6 times the failure rates. In a monolithic performability model, the job arrival rates could be 10^9 times the component failure rates. In this section, we discuss how stiffness can be overcome. To begin with, we describe how the extreme disparity between transition rates translates into a computational problem for numerical solution methods.

Let us consider the linear system of differential equations in Equation 2. This system is considered stiff if the solution has components whose rates of change (decay or gain) differ greatly. The rate of change of each solution component is governed by the magnitude of an eigenvalue of the generator matrix $\mathbf{Q}$. This system is considered stiff if for $i = 2, ..., m$, $Re(\lambda_i) < 0$ and

$$\max_i |Re(\lambda_i)| >> \min_i |Re(\lambda_i)| ,$$

where λ_i are the eigenvalues of $\mathbf{Q}$. The rate of change of a solution component is defined relative to the solution interval, hence Miranker [45] gave the following definition: "a system of differential equations is said to be stiff in the interval $[0, t)$ if there exists a solution component of the system which has variation in that interval that is large compared to $1/t$". However, the CTMC attains numerical steady-state at some finite time t_{ss}: within the specified accuracy (or error tolerance) the state probability vector does not change with increase in time. Hence we may redefine stiffness: "the system of differential equations in Equation 2 is said to be stiff in the interval $[0, t)$ if there exists a solution component of the system which has variation in that interval that is large compared to $1/\min\{t, t_{ss}\}$. The large difference in transition rates of the CTMC approximately translates into large difference in magnitude of the eigenvalues of the generator matrix.

Stiffness could cause numerical instability and make the solution methods inefficient if the methods are not designed to handle stiffness. Like largeness, two basic approaches to overcome stiffness are: stiffness avoidance or stiffness tolerance.

Stiffness Avoidance According to this approach, stiffness is eliminated from a model by applying some approximation scheme. This results in a set of non-stiff models which are then solved to obtain the overall solution. Bobbio and Trivedi [8] have designed one such technique based on aggregation. Most of these approaches avoid largeness as well, since some kind of model decomposition or aggregation is involved.

Stiffness Tolerance Special solution methods that are designed to handle stiffness are used in this approach. The two most commonly used methods for transient analysis of CTMCs are uniformization and numerical ODE solution methods. It has been shown [54, 40] that uniformization is inefficient for stiff CTMCs. A modified implementation of uniformization which incorporates steady-state detection of the underlying discrete-time Markov chain (DTMC) [50] was shown to be more efficient than the standard implementation when the solution interval was larger than t_{ss}. However, uniformization remains much more inefficient than L-stable ODE methods [38]. L-stable ODE methods [34] are recommended for stiff CTMCs. Among these, second-order TR-BDF2 [54] is efficient for low accuracy requirements and third order implicit Runge-Kutta method [40] is efficient for high accuracy requirements. Recently, more efficient methods based on stiffness detection [37] have been proposed.

5.3 Non-exponential distributions

Phase Approximations The basic methodology of phase approximations is to replace a non-exponential distribution in a model by a set of states and transitions between those states such that the holding time in each state is exponentially distributed. This follows from Cox [18], who showed that any non-exponential probability distribution with rational Laplace Steiltjes transform (LST) can be represented by a series of exponential stages with complex valued transition rates. Each stage is entered with some probability and exited (the process stops) with complementary probability. However, conditions to determine whether the resulting function is a proper cdf or not are not known. To overcome this problem, Neuts [52] restricted the Coxian representation by defining phase type distributions as absorbing-time distributions of a CTMC with at least one absorbing state. Non-exponential distributions can be approximated by phase type distributions (also known as phase approximations when used in this context). Distributions without rational LSTs can be approximated by distributions having rational LSTs, although, arbitrarily close approximations may require a CTMC with a large state space.

A complete approach to phase approximations is discussed in [39]. This approach consists of a few basic steps:

- *Selecting a phase approximation class for a given distribution.* One of the most commonly used phase approximation classes is a mixture of Erlang distributions [9]. It has been used in [26, 39, 60] and good fits to some commonly occurring distributions such as Weibull, deterministic, lognormal, and uniform have been obtained. Schmickler [58] has used mixtures of Erlang distributions to fit empirical functions. Bobbio et al. [6, 4, 5] have used a different kind of acyclic phase approximation and obtained good fits to several distributions.
- *Obtaining the parameters of phase approximations.* Once a suitable phase approximation has been chosen for a given distribution (which may be in

empirical form), the next step is to fit the parameters of this phase approximation. The choices include moment matching, function (cdf or pdf) fitting, maximum likelihood estimation (in case of empirical distributions), or a combination of these [9, 39]. Johnson and Taffe [27, 28] have considered matching the first three moments of mixtures of two Erlang distributions. For more references on this topic, refer to [7].

- *Generation of the overall CTMC.* After the parameters of phase approximations for all the non-exponential distributions have been fitted (or estimated), the overall CTMC is generated. This may require the cross-product of phase approximations [39].

A few software packages implementing this approach have been developed. Phase approximations were used in the SURF package [16], although SURF was intended only for a restricted class of reliability models. Cumani [19] has designed the software package ESP for evaluation of SPNs with phase-type distributed firing times. Phase approximations for a class of non-Markovian models have been implemented in GSHARPE [39]. GSHARPE is a front end for a general purpose performance and reliability modeling toolkit called SHARPE [56]. It accepts a non-Markovian model and converts it into a CTMC in SHARPE syntax after applying phase approximations.

Non-homogeneous CTMCs If transition rates in a CTMC are allowed to be time-dependent, where time is measured from the beginning of system operation, the model becomes a non-homogeneous CTMC. Such models are used in software reliability under the name of NHPP (Non-Homogeneous Poisson Process) [51] and in hardware reliability models of non-repairable systems [22]. Tools such as CARE III and HARP allow component failure distributions to be Weibull using this approach.

Markov regenerative processes (MRGPs) The use of non-homogeneous CTMC allows transition rates to be globally time-dependent while the use of SMPs allow the time dependence to be local (since the entry into the state). Both of these are often inadequate in practice. While, in principle, the phase approximations allow more general time dependence, their practical usefulness is limited by the increased size of the underlying stochastic process, which further exacerbates the largeness problem. MRGPs seem to provide a useful time-dependence that can capture many interesting practical scenarios. The basic idea is that not every state change is required to be a regeneration point. Thus, in a multi-component system with each component having exponential time-to-failure distribution and a generally distributed repair with a single repairperson (FCFS), the underlying stochastic process is a MRGP (but not a SMP or a CTMC). Recent work on this topic can be found in [11, 15].

References

1. M. Ajmone-Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems.* MIT Press, Cambridge, MA, 1986.
2. M. Ajmone-Marsan, G. Conte, and G. Balbo. A class of Generalized Stochastic Petri Nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.
3. G. Balbo, S. C. Bruell, and S. Ghanta. Combining queuing networks and GSPN's for the solution of complex models of system behavior. *IEEE Transactions on Computers*, 37:1251–1268, 1988.
4. A. Bobbio and A. Cumani. A Markov approach to wear-out modeling. *Microelectronics and Reliability*, 23(1):113–119, 1983.
5. A. Bobbio and A. Cumani. ML estimation of the parameters of a PH distribution in triangular canonical form. Technical Report R.T. 393, Istituto Elettrotecnico Nazionale Galileo Ferraris, Torino, Italy, 1990.
6. A. Bobbio, A. Cumani, A. Premoli, and O. Saracco. Modeling and identification of non-exponential distributions by homogeneous Markov processes. In *Proc. of the Sixth Advances in Reliability Symposium*, Apr. 1980.
7. A. Bobbio and M. Telek. Parameter estimation of phase type distributions. In *20th European Meeting of Statisticians*, Bath, UK, Sept. 1992.
8. A. Bobbio and K. Trivedi. An aggregation technique for the transient analysis of stiff Markov chains. *IEEE Transactions on Computers*, C-35(9):803–814, Sept. 1986.
9. W. Bux and U. Herzog. The phase concept: approximation of measured data and performance analysis. In K. Chandy and M. Reiser, editors, *Computer Performance*, pp. 23–38. North-Holland, Amsterdam, 1977.
10. K. M. Chandy, U. Herzog, and L. S. Woo. Parametric analysis of queuing networks. *IBM Journal of Research and Development*, 19:43–49, 1975.
11. H. Choi, V. G. Kulkarni, and K. S. Trivedi. Markov Regenerative Stochastic Petri Nets. In *16th IFIP W.G. 7.3 Int'l Sym. on Computer Performance Modelling, Measurement and Evaluation (Performance'93)*, Rome, Italy, Sept. 1993.
12. H. Choi and K. S. Trivedi. Approximate performance models of polling systems using stochastic Petri nets. In *Proc. of IEEE Infocom 92*, pp. 2306–2314, Florence Italy, May 1992.
13. G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. S. Trivedi. Automated generation and analysis of Markov reward models using Stochastic Reward Nets. In C. Meyer and R. J. Plemmons, editors, *Linear Algebra, Markov Chains, and Queueing Models, IMA Volumes in Mathematics and its Applications*, volume 48. Springer-Verlag, Heidelberg, Germany, 1993.
14. G. Ciardo and K. Trivedi. A decomposition approach for stochastic reward net models. *To appear in Performance Evaluation.*
15. G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. In *Proc. of the Fifth Int. Workshop on Petri Nets and Performance Models (PNPM93)*, Toulouse, France, Oct. 1993.
16. A. Costes, J. Doucet, C. Landrault, and J. C. Laprie. SURF: A program for dependability evaluation of complex fault-tolerant computing systems. In *Proc. 11th Intl. Symposium on Fault-Tolerant Computing*, pp. 72–78, 1981.
17. P. Courtois. Computable bounds for conditional steady-state probabilities in large Markov chain and queueing models. *IEEE J. Sel. Areas in Comm.*, SAC-4(6):926–937, 1986.

18. D. Cox. A use of complex probabilities in the theory of stochastic processes. *Proc. of the Cambridge Philosophical Society*, 51:313–319, 1955.
19. A. Cumani. ESP – A package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In *Proc. of International Workshop on Timed Petri Nets*, pp. 144–151, Torino, Italy, July 1985.
20. E. de Souza e Silva and H. R. Gail. Performability analysis of computer systems: from model specification to solution. *Performance Evaluation*, 14:157–196, 1992.
21. J. Dugan, K. Trivedi, M. Smotherman, and R. Geist. The hybrid automated reliability predictor. *AIAA Journal of Guidance, Control and Dynamics*, pp. 319–331, May-June 1986.
22. R. Geist and K. Trivedi. Ultra-high reliability prediction for fault-tolerant computer systems. *IEEE Transactions on Computers*, C-32(12):1118–1127, Dec. 1983.
23. A. Goyal, W. Carter, E. de Souza e Silva, S.S, Lavenberg, and K. Trivedi. The system availability estimator. In *Proc. of IEEE 16th Fault-Tolerant Computing Symposium*, pp. 84–89, July 1986.
24. B. Haverkort and K. Trivedi. Specification and generation of Markov reward models. *Discrete-Event Dynamic Systems: Theory and Applications* 3 , pp.219–247, 1993.
25. R. A. Howard. *Dynamic Probabilistic Systems, Vol.II: Semi-Markov and Decision Processes.* John Wiley & Sons, New York, 1971.
26. M. Johnson. Selecting parameters of phase distributions: combining nonlinear programming, heuristics, and Erlang distributions. To appear in ORSA JOC.
27. M. Johnson and M. Taffe. Matching moments to phase distributions: mixtures of Erlang distribution of common order. *Stochastic Models*, 5:711–743, 1989.
28. M. Johnson and M. Taffe. Matching moments to phase distributions: density function shapes. *Stochastic Models*, 6:283–306, 1990.
29. H. Kantz and K. Trivedi. Reliability modeling of MARS system : A case study in the use of different tools and techniques. In *International Workshop on Petri Nets and Performance Models*, Melbourne, Australia, 1991.
30. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains.* Springer-Verlag, 1976.
31. W. Kleinoder. Evaluation of task structures for hierarchical multiprocessor. In D. Potier, editor, *Modeling Techniques and Tools for Performance Analysis.* North-Holland, 1985.
32. V. Kulkarni, V. Nicola, R. Smith, and K. Trivedi. Numerical evaluation of performability measures and job completion time in repairable fault-tolerant systems. In *Proc. 16th Intl. Symp. on Fault Tolerant Computing,* Vienna, Austria, July 1986. IEEE.
33. K.S. Trivedi and V. G. Kulkarni, "Fluid Stochastic Petri Nets," *Proc. 14th International Conference on Applications and Theory of Petri Nets,* Chicago, June 1993.
34. J. Lambert. *Numerical Methods for Ordinary Differential Systems.* John Wiley and Sons, 1991.
35. S. Lavenberg. *Computer Performance Modeling Handbook.* Academic Press, 1983.
36. E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1984.
37. M. Malhotra. A computationally efficient technique for transient analysis of repairable Markovian systems. To appear in *Performance Evaluation* subject to revision, 1993.

38. M. Malhotra, J. K. Muppala, and K. S. Trivedi. Stiffness-tolerant methods for transient analysis of stiff Markov chains. Technical Report DUKE-CCSR-92-003, Center for Computer Systems Research, Duke University, 1992.
39. M. Malhotra and A. Reibman. Selecting and implementing phase approximations for semi-Markov models. To appear in *Stochastic Models*, 1993.
40. M. Malhotra and K. Trivedi. Higher-order methods for transient analysis of stiff Markov chains. In *Third international conference on Performance of Distributed Systems and Integrated Communication Networks*, Kyoto, Japan, 1991.
41. M. Malhotra and K. Trivedi. Dependability modeling using Petri-net based models. Technical Report DUKE-CCSR-92-012, Center for Computer Systems Research, Duke University, 1992.
42. M. Malhotra and K. S. Trivedi. A methodology for formal expression of hierarchy in model specification and solution. In *Proceedings of Fifth Intl. Workshop on Petri Nets and Performance Models*, 1993.
43. M. Malhotra and K. S. Trivedi. Reliability analysis of redundant arrays of inexpensive disks. *Journal of Parallel and Distributed Computing*, 17:146–151, Jan. 1993.
44. J. Meyer. On evaluating the performability of degradable computer systems. *IEEE Transactions on Computers*, C-29:720–731, Aug. 1980.
45. W. Miranker. *Numerical Methods for Stiff Equations and Singular Perturbation Problems*. D. Reidel, Dordrecht, Holland, 1981.
46. D. Mitra, "Stochastic Theory of Fluid Models of Multiple Failure-Susceptible Producers and Consumers Coupled by a Buffer," *Advances in Applied Probability*, Vol. 20, pp. 646-676, 1988.
47. I. Mitrani. Fixed-point approximations for distributed systems. In G. Iazeolla, P. J. Courtois, and A. Hordijk, editors, *Mathematical Computer Performance and Reliability*, pp. 245–258. North-Holland, 1984.
48. M. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, C-31(9):913–917, Sept. 1982.
49. R. R. Muntz, E. de Souza e Silva, and A. Goyal. Bounding availability of repairable computer systems. *IEEE Transactions on Computers*, C-38(12):1714–1723, Dec. 1989.
50. J. Muppala and K. Trivedi. Numerical transient analysis of finite Markovian queueing systems. In U. Bhat and I. Basawa, editors, *Queueing and Related Models*, pp. 262–284. Oxford University Press, 1992.
51. J. Musa, A. Iannino, and K. Okumoto, *Software Reliability; Measurement, Prediction, Application*, McGraw-Hill, 1987.
52. M. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, Baltimore, MD, 1981.
53. V. Nicola. Lumping in Markov reward processes. In W. Stewart, editor, *Numerical Solution of Markov Chains*, pp. 663–666. Marcel Dekker Inc, New York, 1991.
54. A. Reibman and K. Trivedi. Numerical transient analysis of Markov models. *Computers and Operations Research*, 15(1):19–36, 1988.
55. R. Sahner and K. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, 14(10):1105–1114, Oct. 1987.
56. R. Sahner and K. Trivedi. A software tool for learning about stochastic models. *IEEE Transactions on Education*, 36(1):56–61, Feb. 1993.

57. W. Sanders and J. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Selected Areas of Communications*, pp. 25–36, Jan. 1991.
58. L. Schmickler. MEDA – mixed Erlang distributions as phase-type representation of empirical functions. *Stochastic Models*, 8(1):131–156, Aug. 1992.
59. M. L. Shooman. *Probabilistic Reliability: An Engineering Approach.* McGraw-Hill, New York, 1968.
60. C. Singh, R. Billinton, and S. Lee. The method of stages for non-Markovian models. *IEEE Transactions on Reliability*, R-26(1):135–137, June 1977.
61. L. A. Tomek and K. S. Trivedi. Fixed point iteration in availability modeling. In M. D. Cin and W. Hohl, editors, *Proc. of the 5th International GI/ITG/GMA Conference on Fault-Tolerant Computing Systems*, pp. 229–240, Berlin, Sept. 1991. Springer-Verlag.
62. K. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, Englewood-Cliffs, NJ, 1982.
63. K. Trivedi, J. Muppala, S. Woolet, and B. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14:197–215, 1992.