REWARD MODEL SOLUTION METHODS WITH IMPULSE AND RATE REWARDS: AN ALGORITHM AND NUMERICAL RESULTS

by

Muhammad Akber Qureshi

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 2

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED:_____

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

_____ _____

William H. Sanders
Assistant Professor
Electrical and Computer Engineering

Date

To my parents

ACKNOWLEDGMENTS

There are numerous people I would like to thank for their help and support during this endeavor. First of all, I thank my graduate committee members Dr. Pam Nielsen and Dr. Jo Dale Carothers for reviewing my thesis, and my advisor Dr. William Sanders for his guidance and support throughout this project.

Most of all, I want to thank my family and friends for their morale support and for their lending assistance where they could. I express my sincere appreciation to Gerry Jones and Zahir Khurshid for reading many early versions of this thesis for grammer and style. I especially want to thank Doug Obal and Hemal Shah for their help and morale support throughout this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Reward models have become an important method for specifying performability models for many types of systems. Many methods have been proposed for solving these reward models, but no method has proven itself to be applicable over all system classes and sizes. Furthermore, specification of reward models has usually been done at the state level, which can be extremely cumbersome for realistic models. We describe a method to specify reward models as stochastic activity networks (SANs) with impulse and rate rewards, and a method by which to solve these models via randomization. The method is an extension of one proposed by de Souza e Silva and Gail in which impulse and rate rewards are specified at the SAN level, and solved in a single model. Furthermore, a novel method of discarding trajectories of low probabilities with algorithms to compute bounds on the injected error is proposed. The methodology is presented, together with the results on the time and space efficiency of a particular implementation.

CHAPTER 1

Introduction

In the past few years, the growing dependence on computing systems has caused computer performance and computer reliability evaluation to emerge as important technical disciplines. Traditionally, performance and reliability issues have been distinguished by regarding "performance" as "how well the system performs provided it is failure free" and "reliability" as "how long the system is failure free". Meyer [18] has introduced another approach to quantify behavior of systems with parallel resources that exhibit degradable performance. The approach introduced by Meyer defines a general modeling framework that formulates and evaluates a unified performance-reliability measure, formally introduced as "performability" measure. Reward models, as defined by Howard [14], seem well suited to quantify the performability of systems. In this thesis, we propose a reward model solution method based on randomization to evaluate the probability distribution of the performability measure.

1.1 Performability

Performability and its associated concepts are defined by Meyer [18], as follows. Let S denote a degradable system in question, wherein A defines the set of all possible performance outcomes the system S can accomplish. Then, the performance of S over a specified time

period T can be represented by a random variable Y_s taking values in the set A. Accordingly, the *performability* of S for a set of outcomes B is the probability measure induced by Y_s, provided $B \subseteq A$

$$p_S(B) = \begin{array}{l} probability\ \ that\ \ S\ \ has\ \ a\ \ performance\ \ outcome \\ from\ \ set\ \ B \end{array}$$

Solution of performability is based on the underlying stochastic process $\{X_t : t \leq 0\}$ which represents the dynamics of the system structure and the performance variable Y_s. Performability model construction is the process of identifying a performance variable Y and determining the base model stochastic process X that permits the solution of performability. This process, if used with traditional construction techniques, will lead to models which are stiff. Stiffness is associated with the large differences in magnitude between various rates. Meyer [19] has studied this problem and presented a solution in terms of behavioral decomposition.

1.2 Behavioral Decomposition

Behavioral decomposition, based on time-scale differences, has become an accepted modeling technique for predicting the combined performance and dependability of computer systems. The basic idea behind the behavioral decomposition can be easily understood by considering a gracefully degradable system, where performance-oriented events occur more often than dependability-related events. Therefore, it is reasonable to assume that a degradable system performs in steady state between the structural changes. Based on the above idea of behavorial decomposition, a performability model can be decomposed into a structural model and a number of performance models equal to the number of structural

states. Each performance model provides an estimate of the steady state performance for the particular structural configuration. Howard [14] has defined the performance as a real valued "reward function", $r: \quad M \rightarrow \Re$, defined on the structure state space M.

1.3 Reward Models

Reward models [14] allow us to compute the behavior of a system by analyzing the state occupancies and the transitions of the associated stochastic process. Informally, a *reward model* consists of two parts: a stochastic process $X = \{X_t : t \geq 0\}$ representing the structural behavior of the system and a reward function defined on the state space of the process X. On the basis of the nature of stochastic process, reward models are characterized as Markov reward models and semi-Markov reward models. And on the basis of the reward assignment, *reward functions* are classified into "rate reward functions", $r(i,t)$, and "impulse reward functions", $r(i,j)$. *Rate reward functions* are time dependent rewards associated with occupancy of state i of the structure state space M. *Impulse reward functions* are time independent rewards associated with the transition from state i to state j where $i, j \in M$.

The reward structure quantifies the behavior of the system, but does not distinguish a variable by the utilization interval it is measured for. Typically one might be interested in an instant-of-time variable, interval-of-time variable or time-averaged-interval of time variable, Sanders *et al.* [27]. This thesis focuses on the interval-of-time behavior, which calls for transient analysis of the system. Much work has been done in an attempt to calculate the distribution of reward accumulated over an interval-of-time. In particular, Furchtgott and Meyer [9] define a reward structure to evaluate the performability distribution function

(PDF) of the performability variable by associating non-increasing (fixed) rate rewards to the performance (accomplishment) levels of degradable non-repairable systems. They derive an integral expression for performability by enumerating all the state trajectories of the queueing model. Donatiello and Iyer [6] propose a method based on the Laplace-Stieljes transform while Goyal and Tantawi [10] suggest a recursive solution for analyzing the discrete rate reward model of degradable non-repairable systems.

Relatively little work has been done in obtaining the performability solutions of repairable fault-tolerant systems. Kulkarni et al. [17] consider a Markov process model of repairable fault-tolerant system and investigate a transform solution. de Souza e Silva and Gail [7] present an elegant solution based on randomization and linear combinations of the order statistics of state occupancies in the randomized Markov chain. However, they evaluate the performance variable by defining the reward structure either on the transitions or the state occupancies of the randomized Markov chain rather than the underlying Markov process. There is a significant difference between the two assignments as after the randomization the reward structure based on transitions may lose information due to the resulting self-transition arcs in the subordinated Markov chain.

1.4 SAN-Based Reward Models

In the past few years, the enormous increase in the versatility and complexity of computer systems has placed significant constraints on the types of modeling techniques that can be used. Specifically, it is difficult to model the parallelism, fault tolerance and degradable performance exhibited by complex systems by using queueing networks, since one is typically limited to asking questions regarding service utilization, queue lengths, waiting

times and service times. Stochastic Petri nets (SPNs) [20, 23] are better suited to exhibit the above mentioned characteristics because of their representation at a lower level, but they rapidly run into problems due to the large number of states that need to be considered using traditional methods. Balbo *et al.* [1] has addressed the problem of exponential growth of reachability set as a function of the number of places and tokens in the generalized stochastic Petri net (GSPN) Ciardo *et al.* [2], while modeling the machine scheduling policy of flexible manufacturing systems. Couvillion and Sanders *et al.* [3] and Sanders *et al.* [26] have introduced a model construction technique which significantly reduce the state space in modeling hierarchical systems.

In all existing solution techniques known to the author for computing the reward variables, the repairable or non-repairable systems are modeled by queueing networks which describe the state level structure. The rate rewards associated with the states could be a measure of the throughput of the system and the impulse rewards associated with the transitions could be a loss of performance due to different types of failures. Sanders [25] and Ciadro *et al.* [2] have claimed that this state level reward structure is inadequate to quantify the performance behavior at the network level. Stochastic activity networks (SANs) define reward structures by associating impulse rewards and rate rewards with the activity completions and the number of tokens in the distinguished places, respectively. The network level structure of SANs allows us to define the reward structure at a lower level where the modeler thinks rather than at the state level, which is there after use in the base model construction. Therefore, the assignment of rewards and the interpretation of solutions is more natural.

1.5 Research Objectives

The reward model solution techniques known to the author either for repairable or non-repairable systems consider state level assignment of rewards, which limits the scope of the systems that can be evaluated. Also, none of the solution method make use of reward structure types that are as general as those considered by Howard [14]. Most of the methods use inverse Laplace transforms to compute the transient probabilities. The large time complexity of the transform solution restricts the methods to evaluate only small non-repairable systems. Although some solution methods have used Fourier series approximation of the Laplace transform inversion to minimize time-complexity, they run into numerical instability. This thesis addresses the above mentioned issues and proposes a method to evaluate the PDF of the performability measure based on randomization and linear combination of order statistics.

The initial work for evaluating the performability measure of Markov reward models by using randomization and linear combination of order statistics was presented by de Souza e Silva and Gail [7]. They did avoid the problems with methods that make use of of the complexity of Laplace transforms but their reward structure lacked generality (they either used rate rewards or impulse rewards but not both) and the reward assignment was at the state level. Another drawback was the assignment of rewards to the state occupancies or the transitions of the randomized Markov chain rather than the underlying Markov process, which makes it difficult to capture some system behaviors (explained in detail in the third chapter). Also, their algorithm has inherent problem of memory explosion which they

acknowledge, but do not address. In view of these shortcomings, the goals of this study can be summarized as:

1) The development of a solution method based on a SAN-level assignment impulse and rate rewards.

2) The development of an expansion mechanism which distinguishes between transitions due to activity completions and fictitious self-transitions in the randomized process.

3) The development of a scheme to reduce the memory required by the algorithm, by neglecting certain transition paths depending on the desired accuracy.

4) The development of a program to implement the algorithm.

5) An illustration of the usefulness of the tool through its application to a multiprocessor model.

The remainder of the thesis is organized as follows. Chapter 2 presents an overview of the SANs and discuss their use in performability evaluation. In particular, the construction of the SAN-based reward models (SBRMs) are discussed.

Chapter 3 presents an algorithm for solving SBRMs via randomization. A method for expanding a SBRM, which is based on expanding a Markov renewal process, to a Markov process is presented. From the resulting Markov process, the PDF of the performability measure is computed using randomization and the order statistics of particular paths in the process.

Chapter 4 defines two kinds of errors and computes bounds on these errors. First, the error due to truncation of the states of the randomized process is considered. Second,

the error due to the throwing away of low probability paths to minimize computational complexity is characterized

Chapter 5 presents some numerical results while chapter 6 summarizes the results of this research and suggests future research in this area.

CHAPTER 2

Review of SAN-Based Reward Models

SANs [21, 22] provide a modeling framework to capture the complex behaviors inherent to large distributed systems. The modeling framework is general enough to easily represent realistic system designs and formal enough to permit derivation of analytical results. A model of a system consists of two parts: 1) a representation as a SAN, 2) a reward structure defined on the SAN. Representing systems at the SAN level provides extensibility in the range of questions which can be asked about the system, while the reward structure specifies the particular question (variable) in terms of network behavior. A stochastic activity network along with its reward structure is called a "SAN-based reward model", Sanders [25].

"Reduced based model construction" (RBMC) methods can then be used to derive a stochastic process from a SBRM, that supports the solution of the variable in question. This chapter briefly explains the basic concepts necessary to understand stochastic activity networks.

2.1 Network Level Representation

Structurally, SANs consist of activities, places, gates, and arcs. Activities represent delays and events of the modeled system that affect its dependability and performance.

There are two types of activities, timed and instantaneous. A non-zero completion time is associated with each timed activity, while instantaneous activities complete as soon as they are enabled. Several actions may be possible upon an activity's completion. Cases represent these alternative actions. Each case has a probability which represents the likelihood of the cases occurance. (Note: the sum of probabilities associated to the cases of an activity is always one). Graphically, timed activities are represented by elongated ovals, instantaneous activities by solid bars, and cases by small circles on the output side of an activity.

Places are represented by circles, which can contain small dots called tokens. It is up to the modeler to give appropriate meaning to each place with its collection of tokens. Each place with its tokens define a partial marking or partial state of the system. The places collectively define the marking (state) of a SAN model.

Gates permit greater flexibility in defining enabling and completion rules of activities. They reduce the complexity of a SAN by replacing places, activities and arcs, which could make the graphical representation of the model unmanageable. There are two types of gates; input gates and output gates.

Each *input gate* is depicted by a triangle with a finite set of inputs and one output. Inputs are connected to places while the output is connected to an activity. Each gate has a predicate and a function. The predicate is defined over the marking of the input places. If the predicate is true, the gate is enabled. The predicate activates the activity and, upon completion, executes the gate function. The function may change the markings of the places connected to the gate.

Similarly, each *output gate* is depicted by a triangle but with one input and a finite set of outputs. The input is connected to an activity while the outputs are connected to

places. Each gate has a function, but no predicate. When the activity connected to the gate completes, the gate function is executed which may change the markings of the connected places.

An arc is used to connect activities, places, and gates. The direction of the flow of execution of SAN is indicated by an arrow on the arc.

To illustrate the construction and execution of SANs, figure 2.1 illustrates a SAN representing a multiprocessor system with a finite number of buffer stages. Places *processor* and *buffer* represent the number of fault-free processors and buffer stages, respectively, while the place *repair* indicates the number of processors queued for repair. Activities *processor-failure* and *buffer-failure* represent the occurrence of faults in the processors and buffer stages respectively. Activity *processor-failure* has three associated cases, representing three possible types of processor faults: 1) a repairable fault, 2) a fault causing total system failure, 3) non-repairable processor failure. Activity *buffer-failure* has two associated cases: 1) a non-repiarable buffer failure, 2) a fault causing total system failure. Processor repairs are represented by activity *processor-repair*. The two output gates *G1* and *G2* implement the effect of a total system failure. Tables 2.1, 2.2, and 2.3 represent the associated details about the exponential marking dependent rates of the activities, the case probabilities, and the gate functions, respectively.

2.2 SAN-Based Reward Models

After defining the network level representation of a system, a method is needed to relate the possible behavior of the SAN model to a specified performance variable. Typically, this is done by associating rate rewards to occupation of states and impulse rewards to

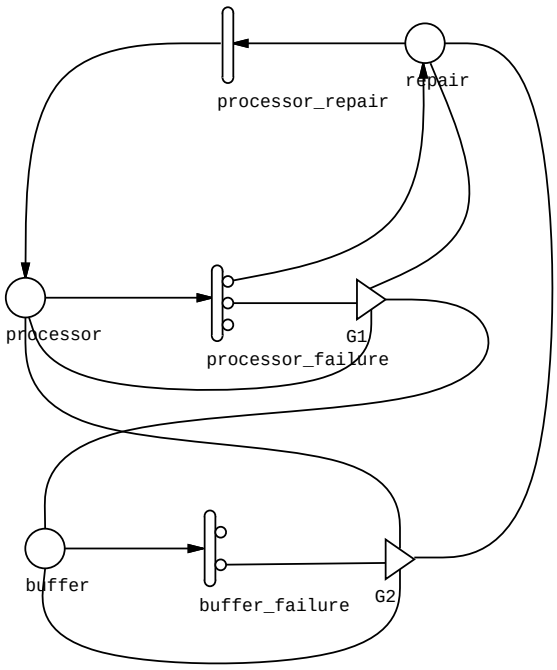


Figure 2.1: Example SAN

Table 2.1: Activity Time Distributions

Activity	**Distribution**
$buffer\_failure$	expon(0.0001 * MARK(buffer))
$processor\_failure$	expon(0.005 * MARK(processor))
$processor\_repair$	expon(0.05)

transition of states. The interval-of-time variables used herein can be evaluated by computing the reward accumulated over the utilization interval. Recall a reward structure at the network level quantifies the benefit obtained by associating impulse rewards with activity completions and rate rewards with the numbers of tokens in places.

Sanders and Meyer [27] define an activity-marking oriented reward structure of a stochastic activity network, with places P and activities A, as a pair of functions, C and R.

Table 2.2: Case Probabilities for Activities

Activity	**Case**	**Probability**
$buffer\_failure$	1	0.95
	2	0.05
$processor\_failure$	1	0.90
	2	0.05
	3	0.05

Table 2.3: Output Gate Functions

Gate	**Function**
$G2$	$MARK(processor) = MARK(buffer) = MARK(repair) = 0$
$G1$	$MARK(processor) = MARK(buffer) = MARK(repair) = 0;$

C: $A \rightarrow \mathbb{R}$ where for $a \in A$, $C(a)$ is the impulse reward obtained due to completion of activity a; and

R: $\mathcal{P}(P, \mathbb{N}) \rightarrow \mathbb{R}$ where for $v \in \mathcal{P}(P, \mathbb{N})$, $R(\nu)$ is the rate reward obtained when, for each $(p, n) \in \nu$, there are n tokens in place p,

where $\mathbb{N}$ is the set of natural numbers and $\mathcal{P}(P, \mathbb{N})$ is the set of all partial functions between P and $\mathbb{N}$.

The cumulative measure of the behavior of the system for the interval of time $[0, t]$ is

$$Y_{[0,t]} = \sum_{\nu \in \mathcal{P}(P, \mathbb{N})} \mathrm{R} \times J^{\nu}_{[0,t]} + \sum_{\mathrm{a} \in A} C(a) \times N^{a}_{[0,t]} \tag{2.1}$$

Here $J^{v}_{[0,t]}$ is the random variable representing the total time that the SAN is in a marking such that, for each $(p, n) \in v$, where there are n tokens in p during $[0, t]$, and $N^{a}_{[0,t]}$ is a random variable representing the number of completions of activity a during the time interval $[0, t]$.

The concept of SAN-based reward model can be illustrated by formulating the availability measure of the multiprocessor model of figure 2.1. Define the system to be available whenever there is at least one processor functioning. With the reward structure

$$C(a) = 0, \quad \forall a \in A$$

$$R(v) = \begin{cases} 0 & \text{if } v = \{(processor, 0)\} \\ 1 & otherwise \end{cases}$$

the availability measure is defined.

2.3 Reduced Base Model Construction

A problem with traditional analysis techniques is that they generate huge numbers of states for realistic sized systems. The state space representation of even very small systems can grow too large to solve in a reasonable period of time. Therefore, a state space reduction scheme is required which is solvable and can support performance, dependability, and performability variables. RBMC defines such a scheme for a restricted, but common, class of SANs. The restricted class includes stochastic activity networks that have replicated components with identical reward structures. For systems with some degree of replication the base model is considerably smaller than that obtained using traditional techniques. RBMC allows the solution of much larger systems with replicated components.

The RBMC method defines two primitive operations, "replicate" and "join", to construct a complete model of a system. The *replicate* operation replicates a SAN-based reward model and is extremely useful when a system has two or more identical components. The effect of the replicate operation depends upon a set of places which allow communication between

the replicated submodels. Such places are called *distinguished* (or common) places and are not replicated when the operation is carried out. Informally, the result of the replicate operation on a SAN-based reward model is another SAN-based reward model, where

- The SAN contains a set of common places and the number of copies of all other places (distinguished places) equal to the number of replicates.
- The reward structure assigns an impulse reward to each replicate activity equal to its reward in the original model and reward rates to each partial marking in the new model equal to the rate assigned to the corresponding partial marking in the original model.

The *join* operation permits the representation of systems that consist of several different components, each of which may be replicated. Like the replicate operation, the join operation acts on and produces SAN-based reward models. Again, common places are defined which allow communication between joined subnetworks. The reward structure is defined by assigning

- an impulse reward to each activity in the new model equal to the reward of the corresponding activity in the original model, and
- a reward rate for each partial marking in the new model equal to the rate assigned to the corresponding partial marking in the original model.

Algorithms to construct reduced base models are illustrated in Sanders [25]. Rai [24] implemented these algorithms to generate a transition-rate state diagram, which are considered as input to the solution method proposed herein. In the next chapter, an algorithm to solve for the probability distribution of an interval-of-time variable is discussed.

CHAPTER 3

Performability Solution

Most of the methods for computing PDF of the performability variable require a process to be Markov in order to be solved. To satisfy this requirement, the activity time distribution function of a SAN must be exponential and its activities must be reactivated often enough such that the activity times only depend on the current marking. But a problem with Markov processes is that they are unable to count the number of transitions to the same state (self-transitions). A Markov process representation which can capture the self-transition behavior of SANs is presented. The process representation with appropriate reward assignments is used to compute PDF via randomization.

3.1 State Expansion

Sanders [25] describes the resulting "activity marking behavior" (am-behavior) from the execution of SANs as a time homogeneous Markov renewal process $\{R_n, T_n : n \in N\}$ where T_n is the time of the n^{th} activity completion, and R_n is the "activity marking state" (am-state) reached after the n^{th} time activity completion. The minimal am-behavior for SANs which is a Markov process can be derived from the am-behavior by looking at the am-states as a function of time, [25]. The minimal am-behavior can support instant-of-time variables, but not interval-of-time variables, since activity completions that do not result

in an am-state change (note: similar to self-transitions) cannot be detected. To model the interval-of-time variables, the am-behavior must be expanded to a Markov process which holds both activity and marking related behavior. The expansion of an am-behavior to a Markov process has the following requirements:

1) Expand states which have self-transitions.

2) Replace each self-transition state by a combination of two states which inherit the reward structure and all the outgoing transitions from the replaced state.

3) Only one of the combination states inherits the incoming transitions.

4) Assignment of the self-transition rate of the replaced state to the transitions between the new states.

Figure 3.1 considers an example SAN with an initial marking of 2 tokens in place A and 0 tokens in place B. Tables 3.1 associates the completion rates with the activities and the probabilities with the cases of activities, respectively. The reward structure is defined by associating an impulse reward of 1 with the completion of activity $T2$ and some marking dependent rate reward with the place B. Figure 3.2 depicts the am-behavior (Markov renewal process) of the SAN, and derives the expanded Markov process by fulfilling the above mentioned requirements. Each transition is labeled with transition rate and each state in figure 3.2 is represented as $AB_{\{r,i\}}$ where AB is the marking of the SAN and, r and i define the related rate and impulse rewards.

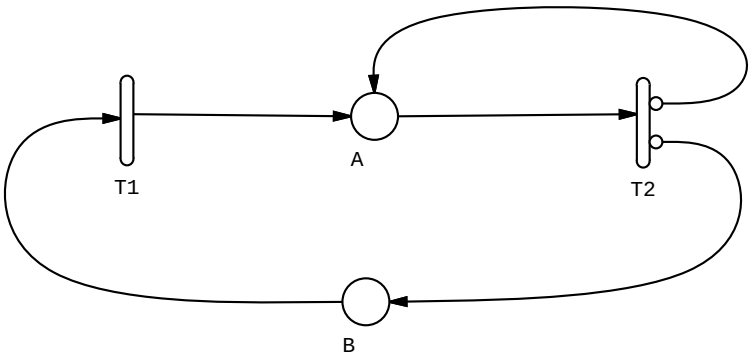


Figure 3.1: Example SAN

Table 3.1: Activity Time Distributions

Activity	**Distribution**	**Case Probabilities**	
		1	2
$T1$	expon(.004)	1	0
$T2$	expon(0.02)	0.4	0.6

3.2 Randomization

By using randomization, a Markov process $\{X(t) : t \geq 0\}$ defined on a countable state space $S = \{a_i : i = 1, 2, \ldots, M\}$ can be represented by a Markov chain $\{Z_n : n \in \mathrm{N}\}$ with the same state space S and a Poisson process $\{N_t : t \geq 0\}$. This approach is based on the subordination of a Markov chain to a Poisson process. The Poisson process counts the number of transitions of the Markov chain with a rate defined by the minimum holding time (minimum diagonal element of the generator matrix) of the Markov process.The general form of the randomization equation is, according Gross and Miller [12],

$$P\{X_{t+s} = j \mid X_t = i\} = \sum_{n=0}^{\infty} \frac{e^{-\lambda s}(\lambda s)^n}{n!} k^n(i,j) \tag{3.1}$$

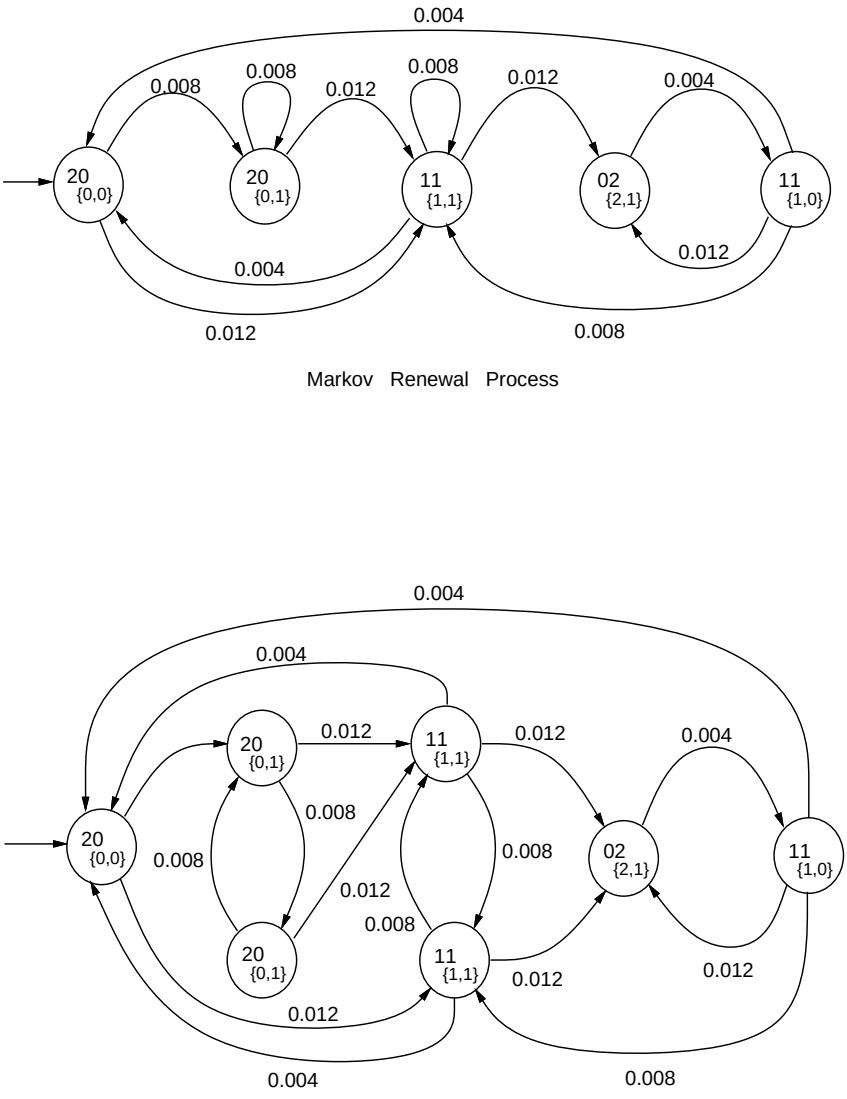


Figure 3.2: State Expansion

where λ is the rate of the Poisson process and $k^n(i,j)$ is the probability that Markov chain Z moves from the state $X_t = i$ to $X_{t+s} = j$ in n-steps.

In the proposed solution method, the expanded Markov process is first randomized. Then, the distribution of the reward variable is calculated by conditioning on the trajectories of the randomized process. The paths are enumerated with respect to reward accumulated. To keep track of the reward accumulated for each transition of the randomized Markov chain, in the next section, we define two vectors corresponding to each type of reward: rate and impulse.

3.3 Trajectory Vectors

Let M and Q be the total number of states and transition arcs, respectively, in a randomized Markov chain Z with $K+1 \leq M$ distinct rate rewards and $J+1 \leq Q$ distinct impulse rewards. Consider a trajectory of the process during some interval of time. The intervals between the transitions correspond to sojourn times in states of the Markov chain Z. Each trajectory then consists of $n+1$ sojourn times, and n state transitions. Two vectors are assigned to each trajectory, one for each type of reward, such that the number of elements in each vector is equal to the number of different possible rewards of the particular type. Specifically, the vectors

$$\mathbf{k} = < k_1, k_2, \ldots, k_{K+1} >$$

$$\mathbf{j} = < j_1, j_2, \ldots, j_{J+1} >$$

are assigned to each trajectory. Where for each rate reward $i = 1, 2, \ldots, K+1$, there are k_i entrances to states of rate i. Similarly, for each impulse reward $i = 1, 2, \ldots, J+1$,

there are j_i entrances to states with impulse i. These vectors are constrained such that $0 \leq k_i \leq n+1$, for $i = 1, 2, \ldots, K+1$, $0 \leq j_i \leq n$, for $i = 1, 2, \ldots, J+1$, and

$$\sum_{i=1}^{K+1} k_i = n+1,$$

$$\sum_{i=1}^{J+1} j_i = n.$$

The total reward accumulated during some period of time [0, t] can be expressed, using randomization, by conditioning first on the number of transitions during [0, t] and, further on $\mathbf{k}$ and $\mathbf{j}$, as follows

$$Y_{[0,t]} = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k}} \sum_{\forall \mathbf{j}}, \ [n, \mathbf{k}, \mathbf{j}] Y(t, n, \mathbf{k}, \mathbf{j}), \tag{3.2}$$

where , $[n, \mathbf{k}, \mathbf{j}] = P[\mathbf{j}$ *transition arcs and* $\mathbf{k}$ *intervals* $\mid n$ *transitions*] and

$Y(t, n, \mathbf{k}, \mathbf{j})$ is $Y_{[0,t]}$, given t, n, $\mathbf{k}$ and $\mathbf{j}$. Given (3.2), the PDF of $Y_{[0,t]}$ can be expressed as

$$P[Y_{[0,t]} \leq y] = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k}} \sum_{\forall \mathbf{j}}, \ [n, \mathbf{k}, \mathbf{j}] P[Y_{[0,t]} \leq y \ \mid \ n, \mathbf{k}, \mathbf{j}]. \tag{3.3}$$

In order to solve this equation, it is necessary to be able to calculate $P[Y_{[0,t]} \leq y \ \mid \ n, \mathbf{k}, \mathbf{j}]$ and , $[n, \mathbf{k}, \mathbf{j}]$.

3.4 Calculation of Conditional Distribution

Consider the calculation of $P[Y_{[0,t]} \leq y \ \mid \ n, \mathbf{k}, \mathbf{j}]$. Our calculation is similar to that of de Souza e Silva and Gail [7], but complicated by the fact that we condition on two reward vectors (rate and impulse) instead of one. Define l_i to be the sum of the lengths of intervals with rate reward i where $i = 1, 2, \ldots, K+1$. Suppose the rate rewards are ordered such

that

$$r_1 > r_2 > \ldots > r_{K+1} \geq 0,$$

and $s_1, s_2, \ldots, s_{J+1}$ are the impulse rewards. Then

$$Y(t, n, \mathbf{k}, \mathbf{j}) = \sum_{i=1}^{K+1} r_i l_i + \sum_{i=1}^{J+1} s_i j_i, \tag{3.4}$$

where $\sum_{i=1}^{J+1} s_i j_i$ is a deterministic value, given n, $\mathbf{k}$, $\mathbf{j}$. Thus $Y(t, n, \mathbf{k}, \mathbf{j})$ is in the range

$$r_{K+1} t + \sum_{i=1}^{J+1} s_i j_i \leq Y(t, n, \mathbf{k}, \mathbf{j}) \leq r_1 t + \sum_{i=1}^{J+1} s_i j_i. \tag{3.5}$$

de Souza e Silva and Gail [7] evaluate such an expression by using a linear combination of the order statistics of the length of k_i intervals for the performability measure $Y[0, t]$ conditioned on $\mathbf{k}$. They exploit the independence between the Poisson process N and the Markov chain Z to arrange the state occupancy times T_a, where $a = 1, 2, \ldots, n+1$, such that first k_1 intervals are of rate r_1, the next k_2 intervals of rate r_2, and so on. Then U_k can be defined to be the k^{th} order statistic of a set of n independent and identically distributed random variables T_a, and sum of the lengths of intervals of rate reward r_i, where $i = 1, 2, \ldots, K+1$, can be expressed as

$$\begin{aligned} l_1 &= U(k_1) \\ l_2 &= U(k_1 + k_2) - U(k_1) \\ &\vdots \\ l_{K+1} &= t - U_{(k_1 + \ldots + k_K)}. \end{aligned}$$

Given this,

$$Y(t, n, \mathbf{k}, \mathbf{j}) = \sum_{h=1}^{K} [(r_h - r_{h+1}) U_{(n_h)}] + r_{K+1} t + \sum_{i=1}^{J+1} s_i j_i,$$

where $n_h = \sum_{l=1}^{h} k_l$. This implies that

$$P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] = P\left[\sum_{h=1}^{K} [(r_h - r_{h+1}) U_{(n_h)}] + r_{K+1} t + \sum_{i=1}^{J+1} s_i j_i \leq y \right].$$

de Souza e Silva and Gail [7] then use the result by Weisberg [29] to evaluate the conditional distribution of performability measure.

The result of [29] is applicable here if the variable parameter is shifted to be within the range described by Dempster and Klyle [5], according to which $0 \leq y \leq r_1 t$ and $r_1 t \geq r_2 t \geq \ldots \geq r_{K+1} t$. To fulfill the condition, a shifting of axis can be done such that

$$P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] = P\left[\sum_{h=1}^{K} (r_h t - r_{h+1} t) \frac{U_{(n_h)}}{t} \leq y - r_{K+1} t - \sum_{i=1}^{J+1} s_i j_i \right]. \tag{3.6}$$

Now, the shifted parameter is in the range

$$0 \leq y - r_{K+1} t - \sum_{i=1}^{J+1} s_i j_i \leq r_1 t.$$

To simplify the notation, define

$$x = y - r_{K+1} t - \sum_{i=1}^{J+1} s_i j_i,$$

and for $1 \leq i \leq K$,

$$c_i = \sum_{h=i}^{K} (r_h t - r_{h+1} t) = r_i t - r_{K+1} t.$$

Further, let

$$c_{K+1} = r_{K+1} t - r_{K+1} t = 0$$

then, $c_1 > c_2 > \ldots > c_K > c_{K+1} = 0$. Applying the results of Weisberg [29], yields

$$P\left[\sum_{h=1}^{K} (r_h t - r_{h+1} t) \frac{U_{(n_h)}}{t} \leq x \right] = 1 - \sum_{i=1}^{m} \frac{g_i^{(k_i - 1)}(c_i)}{(k_i - 1)!} \tag{3.7}$$

where $g_i^k(c_i)$ is the k^{th} derivative of function $g_i(c_i)$ and m is the largest index i such that $x \leq c_i$ and x as defined earlier. Weisberg also shows that derivatives of functions $g_i(c_i)$ can be calculated as follows

$$g_i^{(k)}(c_i) = \sum_{j=0}^{k-1} \begin{pmatrix} k-1 \\ j \end{pmatrix} g_i^{(j)}(c_i) h_i^{(k-1-j)}(c_i),$$

$$h_i^{(j)}(c_i) = (-1)^j j! \left[\frac{n}{(c_i - x)^{j+1}} - \sum_{l=1,\ l \neq i}^{K+1} \frac{k_l}{(c_i - c_l)^{j+1}} \right].$$

Note that $k_i = 0$ if no state with rate reward i is visited. This implies that the element corresponding to k_i in the realization of $\mathbf{k}$ has a value of 0. Such k_i's do not contribute to the conditional distribution given n, $\mathbf{k}$, and $\mathbf{j}$. A simple renumbering is used to disregard such elements of $\mathbf{k}$. Let $l(\mathbf{k}, 1)$ be the index of the first nonzero k_i, $l(\mathbf{k}, 2)$ be the index of the second nonzero k_i, and so on. Then the conditional distribution of $Y_{[0,t]}$ can be expressed as follows

$$P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] = 1 - \sum_{i=1}^{m(\mathbf{k},\mathbf{j})} \frac{g_{l(\mathbf{k},i)}^{(k_{l(\mathbf{k},i)}-1)}(c_{l(\mathbf{k},i)})}{(k_{l(\mathbf{k},i)}-1)!}, \tag{3.8}$$

where $m(\mathbf{k}, \mathbf{j})$ defines the selection of m, as described above, given $\mathbf{k}$, and $\mathbf{j}$. Using the above results, the distribution of accumulated reward can be expressed as

$$P[Y_{[0,t]} \leq y] = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k}} \sum_{\forall \mathbf{j}} , [n, \mathbf{k}, \mathbf{j}] \left[1 - \sum_{i=1}^{m(\mathbf{k},\mathbf{j})} \frac{g_{l(\mathbf{k},i)}^{(k_{l(\mathbf{k},i)}-1)}(c_{l(\mathbf{k},i)})}{(k_{l(\mathbf{k},i)}-1)!} \right]. \tag{3.9}$$

To solve (3.9), we still need to compute , $[n, \mathbf{k}, \mathbf{j}]$ and the summations. The first summation is infinite, while the second and third summations can grow very fast. The second summation has a combitorial growth rate of $\begin{pmatrix} K+n+1 \\ n+1 \end{pmatrix}$ while the growth rate of the

third summation depends upon the number of distinct impulse rewards assigned to the set of states with a distinct rate reward. These summations quickly limit the size of process that may be solved, if some truncation method is not employed. In the next chapter first, based on the generation of a trajectory graph, an algorithm for recursively calculating , $[n, \mathbf{k}, \mathbf{j}]$, for increasing n, $\mathbf{k}$, and $\mathbf{j}$ is proposed. Then methods for truncating each of the summations are proposed.

CHAPTER 4

Algorithms and Error bounds

The previous chapter used randomization to derive an infinite sum representing the distribution of accumulated reward over a finite interval. In this chapter, algorithms and error bounds required to efficiently solve equation (3.9) are defined. First, an algorithm for generating the trajectory graph with respect to the three summations: 1) over all realizations of vector $\mathbf{j}$ for a particular realization of vector $\mathbf{k}$, 2) over all realizations of vector $\mathbf{k}$ for a particular transition n, 3) over all number of transitions n, is presented. Second, an example is introduced which helps to better understand the relation between n, $\mathbf{k}$, $\mathbf{j}$ and the generated graph. Third, two types of errors are defined which arise in the practical implementations of the method. Algorithms to compute bounds on these errors are presented. Finally, computational equations with bounds are presented.

4.1 Trajectory Graph Generation

Trajectory graph keeps track of all the possible trajectories generated with each randomized transition of the Markov chain. It represent trajectories with realizations of vectors $\mathbf{k}$ and $\mathbf{j}$. Each trajectory is defined with one element from each of the three summations and the last state reached. Trajectory graph is used to compute the conditional probabilities

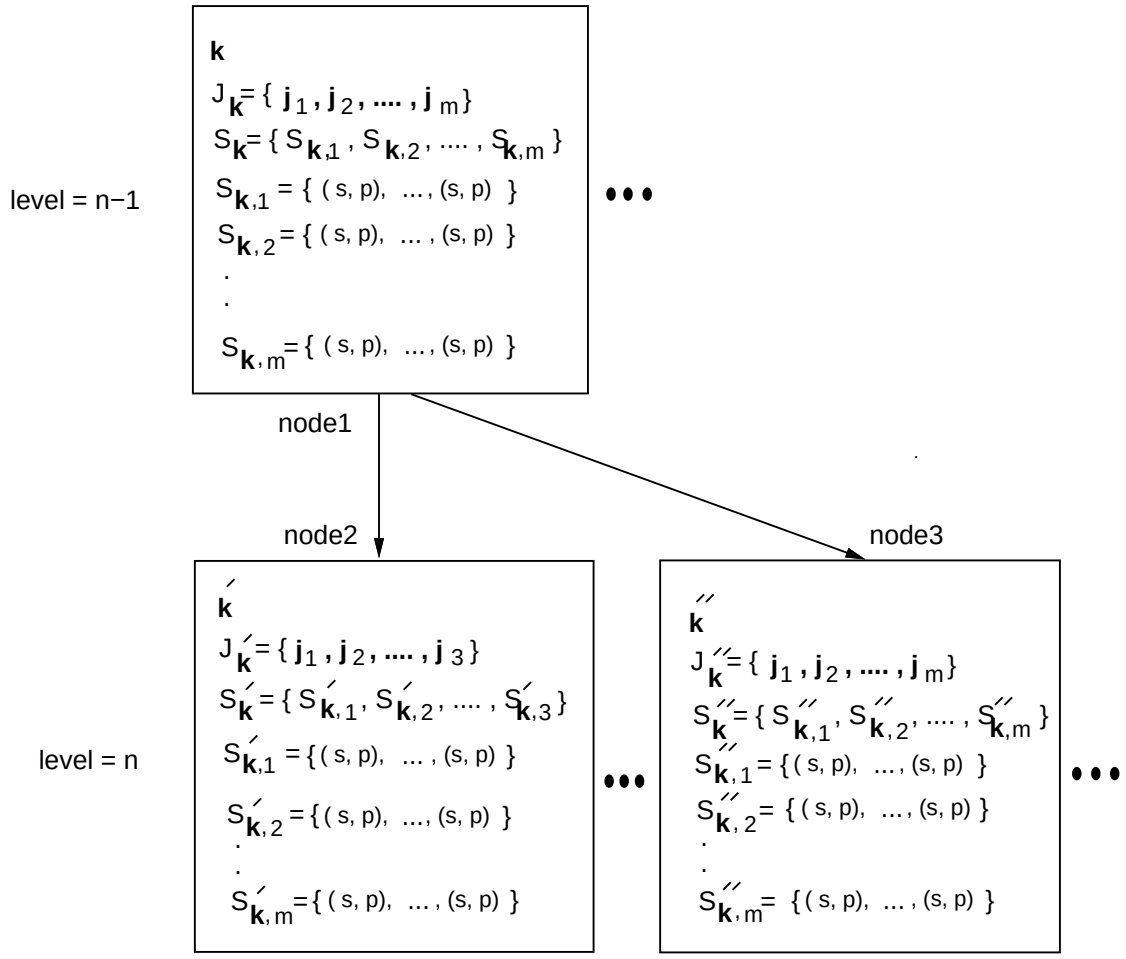


Figure 4.1: Acyclic Graph

, $[n, \mathbf{k}, \mathbf{j}]$ and also, provides a way to truncate the summation which is explained in the error bound section.

The generation of a trajectory graph, with respect to the three summations in equation (3.9), is illustrated in figure 4.1. Each level n of the graph, corresponding to the n^{th} transition of the randomized process, consists of a set of nodes η_n. The number of nodes at a level n can grow combitorially by $\begin{pmatrix} K+n+1 \\ n+1 \end{pmatrix}$, where $K+1$ is the number of distinct rate rewards. Each node is defined by a tuple $(\mathbf{k}, J_{\mathbf{k}}, S_{\mathbf{k}})$, as follows:

$\mathbf{k}$, a particular realization of vector $\mathbf{k}$,

$J_{\mathbf{k}}$, a set of possible $\mathbf{j}$, when $\mathbf{k}$ is the vector of rate rewards,

$S_{\mathbf{k}}$, a set of sets $S_{\mathbf{k},\mathbf{j}}$ where each set $S_{\mathbf{k},\mathbf{j}}$ corresponds to a distinct $\mathbf{k}$ and $\mathbf{j} \in J_{\mathbf{k}}$, and

$S_{\mathbf{k},\mathbf{j}}$, a set of pairs (s,p), where s is the state of the expanded Markov process, and p is the sum of probabilities of all the transitions to state s resulting in $\mathbf{k}$ and $\mathbf{j}$.

Based on the above definitions, an algorithm to compute η_n given η_{n-1} is:

Note: $p(s,s^{'})$ is single-step transition probability from state s to $s^{'}$.

Algorithm 4.1.1 *(Compute η_n given η_{n-1})*

$\eta_n = \phi$

for each $\mathbf{k} \in \eta_{n-1}$*:*

for each $\mathbf{j} \in J_{\mathbf{k}}$*:*

for each $(s,p) \in S_{\mathbf{k},\mathbf{j}}$*:*

for each $s^{'}$ *reachable from* s*:*

compute $\mathbf{k}^{'}$, $\mathbf{j}^{'}$.

if $\mathbf{k}^{'} \in \eta_n$

if $\mathbf{j}^{'} \in J_{\mathbf{k}^{'}}$

if $(s^{'},p^{'}) \in S_{\mathbf{k}^{'},\mathbf{j}^{'}}$ *for some* $p^{'}$

$p^{'} = p^{'} + p \times p(s,s^{'})$.

else

$$S_{\mathbf{k}',\mathbf{j}'} = S_{\mathbf{k}',\mathbf{j}'} \cup (s', p \times p(s, s')).$$

else

$$J_{\mathbf{k}'} = JJ_{\mathbf{k}'} \cup \{\mathbf{j}'\}.$$

$$S_{\mathbf{k}',\mathbf{j}'} = (s', p \times p(s, s')).$$

else

$$\eta_n = \eta_n \cup \{\mathbf{k}'\}.$$

$$J_i' = \{\mathbf{j}'\}.$$

$$S_{\mathbf{k}',\mathbf{j}'} = \{(s', p \times p(s, s')\}.$$

Next s'.

Next $(s, p) \in S_{\mathbf{k},\mathbf{j}}$.

Next $\mathbf{j} \in J_{\mathbf{k}}$.

Next $\mathbf{k} \in \eta_{n-1}$.

For a better understanding of equation (3.9), it would be appropriate to rewrite it in terms of the definitions of node and their elements. According to the above mentioned definitions, equation (3.9) can be precisely written as

$$P[Y_{[0,t]} \leq y] = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k} \in \eta_n} \sum_{\forall \mathbf{j} \in J_{\mathbf{k}}} P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] \sum_{\forall (s,p) \in S_{\mathbf{k},\mathbf{j}}} , [n, \mathbf{k}, \mathbf{j}, s], \quad (4.1)$$

where , $[n, \mathbf{k}, \mathbf{j}, s]$ is the probability of generating $\mathbf{k}$ and $\mathbf{j}$ such that the last state reached is s, given n, and $P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}]$ as defined earlier.

4.1.1 Example

To help understand the generation of a trajectory graph, we consider the uniformed Markov chain of figure 4.2, corresponding to the SAN example considered earlier. The

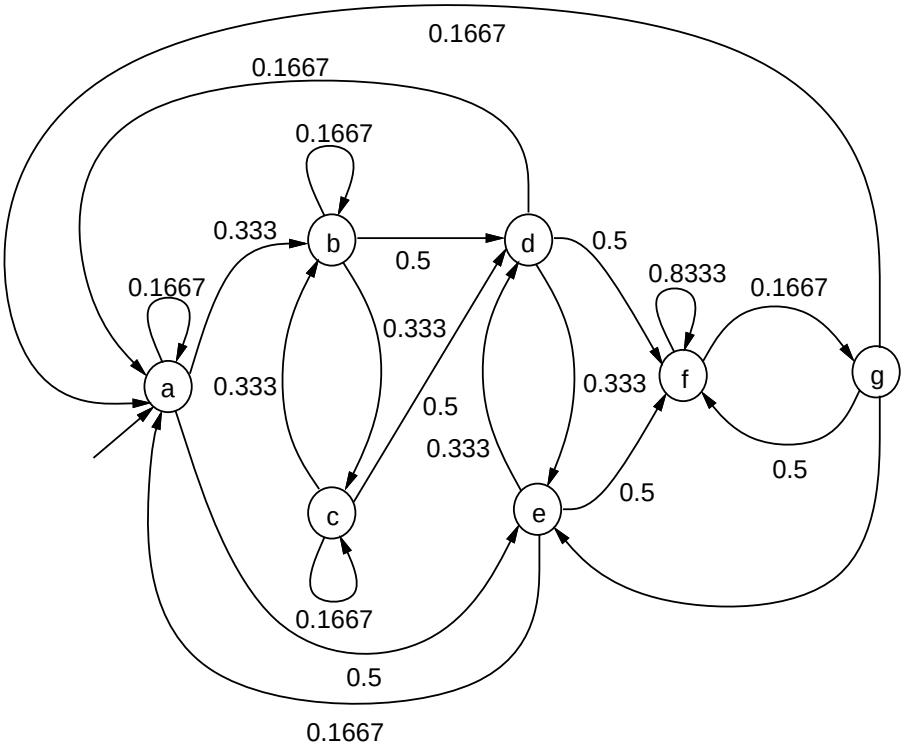


Figure 4.2: Randomized Markov Process

process consists of seven states with three distinct rate rewards and two distinct impulse rewards. According to the definition of $\mathbf{k}$ and $\mathbf{j}$, we let $\mathbf{k} = < k_1, k_2, k_3 >$ where k_1, k_2, and k_3 correspond to intervals of rate rewards 2, 1, and 0 and $\mathbf{j} = < j_1, j_2 >$ where j_1, and j_2 correspond to transitions to states with impulse rewards 1, and 0, respectively. For matter of convenience, we rename states of the process as: $20_{\{0,0\}}$ by a, $20_{\{0,1\}}$ by b, $20_{\{0,1\}}$ by c, $11_{\{1,1\}}$ by d, $11_{\{1,1\}}$ by e, $02_{\{2,1\}}$ by f, and $11_{\{1,0\}}$ by g.

Figure 4.3 gives a lexicographic illustration of possible realizations $\mathbf{k}$ and $\mathbf{j}$ for 0, 1, and 2 transitions of the process. Each node represents a possible realization $\mathbf{k}$, corresponding to a possible assignment of rate rewards. Each element of $\mathbf{k}$ indicates the number of transitions to states with a particular rate reward. The elements are arranged such that the left-most

element corresponds to the highest rate reward, the following element corresponds to the next highest rate reward and so on. Each node contains a set of realizations of $\mathbf{j}$, which represent the possible impulse rewards accumulated in reaching a particular $\mathbf{k}$. Also, each node contains sets of (s, p) corresponding to each $\mathbf{j}$ where s is the state to which the last transition is made resulting in $\mathbf{k}$ and $\mathbf{j}$, and p is the sum of probabilities of all the trajectories to the state s.

To illustrate the computation of entries in the nodes, we trace a trajectory of the process in figure 4.3. Initially, the process is in state a with a probability of 1. According to the reward assignments, the realization of vectors are $\mathbf{k}=< 0, 0, 1 >$ and $\mathbf{j}=< 0, 0 >$. The process will then make a transition to state e with the probability of 0.5, resulting in $\mathbf{k}=< 0, 1, 1 >$, $\mathbf{j}=< 1, 0 >$, and $(s, p) = (e, 0.5)$. Similarly, a transition from state e to state f with probability of 0.5 would generate $\mathbf{k}=< 1, 1, 1 >$ and $\mathbf{j}=< 2, 0 >$, and $(s, p) = (f, 0.5 \times 0.5)$. Other entries in the figure are generated in a similar manner.

To solve (3.9), now we only need to compute the summations. The first summation is infinite, while the second and third summations can grow very fast. The second summation has a combitorial growth rate of $\begin{pmatrix} K+n+1 \\ n+1 \end{pmatrix}$ while the growth rate of the third summation depends upon the number of distinct impulse rewards assigned to the set of states with a distinct rate reward. These summations quickly limit the size of process that may be solved, if some truncation method is not employed. In the next section, propose methods for truncating each of the summations.

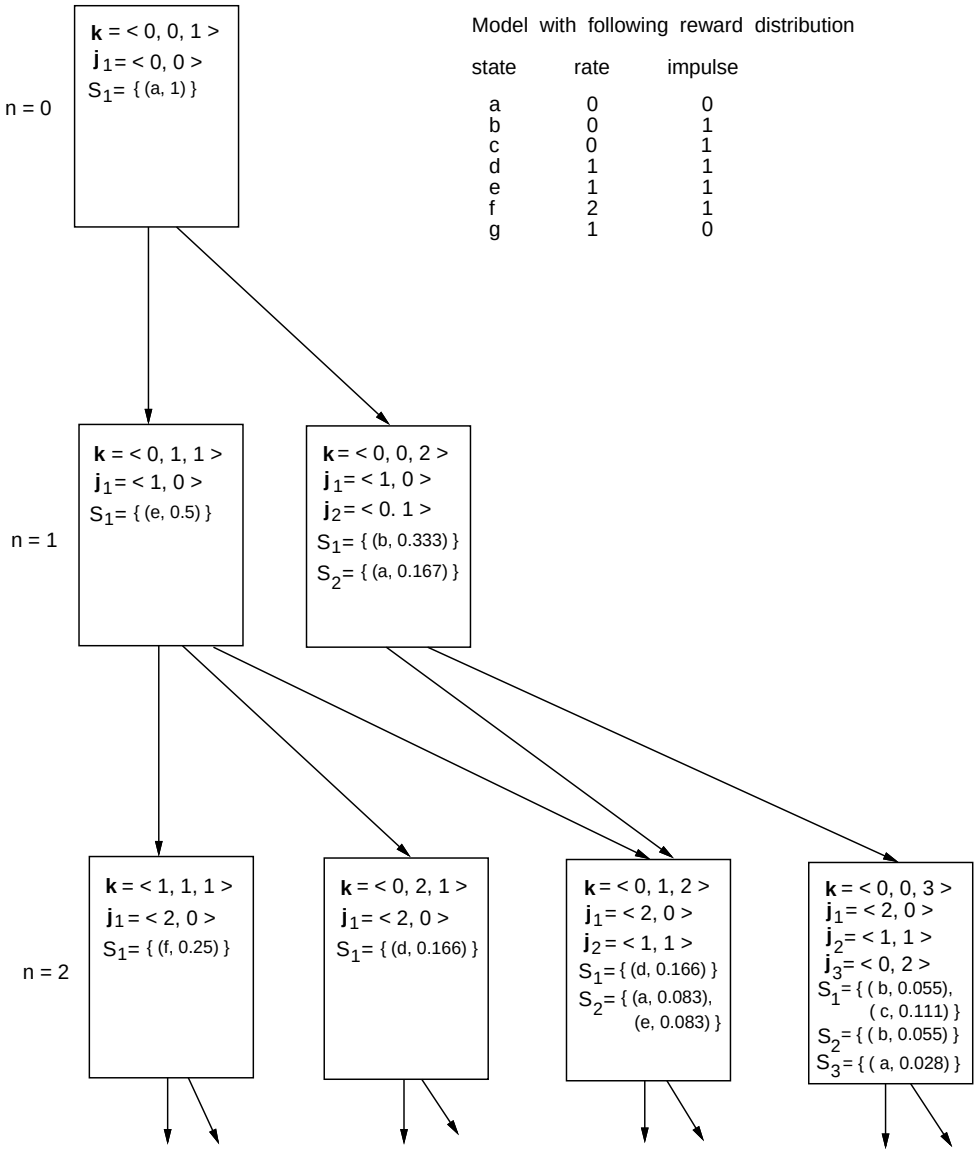


Figure 4.3: Trajectory Graph

4.2 Error Bounds

Trajectory graph gives us a way to truncate the summations in two ways: 1) limiting the depth of the graph, and 2) discarding particular (s, p), depending upon the probability p.

4.2.1 Error Due to Depth Truncation

Truncation based on the depth of the graph is simple and not new. Gross and Miller [12] and de Souza e Silva and Gail [7] have pointed out how to pick a truncation point on the first summation to achieve a given error bound. In particular, the error induced due to the truncation of the infinite summation to the N^{th} step, E_N, is

$$E_N = \sum_{n=N+1}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k} \in \eta_n} \sum_{\forall \mathbf{j} \in J_{\mathbf{k}}} P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] \sum_{\forall (s,p) \in S_{\mathbf{k},\mathbf{j}}}, [n, \mathbf{k}, \mathbf{j}, s]. \tag{4.2}$$

The error defined in equation (4.2) can be bounded by observing that $P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] \leq 1$ for all y and $\sum_{\forall \mathbf{k} \in \eta_n} \sum_{\forall \mathbf{j} \in J_{\mathbf{k}}} \sum_{\forall (s,p) \in S_{\mathbf{k},\mathbf{j}}}, [n, \mathbf{k}, \mathbf{j}, s] = 1$. Therefore

$$E_N \leq \sum_{n=N+1}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!}, \tag{4.3}$$

or, equivalently,

$$E_N \leq 1 - \sum_{n=0}^{N} \frac{e^{-\lambda t}(\lambda t)^n}{n!}. \tag{4.4}$$

Equation (4.4) can be used to calculate a truncation point N, to achieve a desired error.

4.2.2 Error Due to Trajectory Truncation

Even with depth truncation, the solution of equation (3.9) is immensely complex, since the size of the set $S_{\mathbf{k}}$ for a realization $\mathbf{k}$ increases exponentially with the number of randomization steps, which can cause memory explosion. We propose a new method, in which certain $(s,p) \in S_{\mathbf{k},\mathbf{j}} \in (\mathbf{k}, J_{\mathbf{k}}, S_{\mathbf{j}})$, with small p, are discarded, and a bound on the error induced by this discarding is computed. To a large extent, the memory explosion can be avoided. We call this trajectory truncation, since this action effectively truncates certain trajectories whose final state is s. A particular (s,p) in the node of the acyclic graph can be ignored if p is less than a defined variable weight. As will be seen in the next section, the selection of the weight value is a trade off between memory consumed and the error produced. Let E_P be the error induced by discarding certain trajectories for which

$$, [n, \mathbf{k}, \mathbf{j}, s] \leq \ weight. \tag{4.5}$$

To compute E_p, we first compute $E_P(n, \mathbf{k}, \mathbf{j}, s)$, the error produced by discarding a particular (s,p). The error will be

$$E_P(n, \mathbf{k}, \mathbf{j}, s) = \frac{e^{-\lambda t}(\lambda t)^n}{n!} P[Y_{[0,t]} \leq y \ \mid \ n, \mathbf{k}, \mathbf{j}], [n, \mathbf{k}, \mathbf{j}, s]. \tag{4.6}$$

Since the conditional distribution

$$P[Y_{[0,t]} \leq y \ \mid \ n, \mathbf{k}, \mathbf{j}] \ \leq \ 1 \qquad \forall \ y, \tag{4.7}$$

we can bound the error by assuming it equal to 1. This implies that

$$E_P(n, \mathbf{k}, \mathbf{j}, p) \ \leq \ \frac{e^{-\lambda t}(\lambda t)^n}{n!}, \ [n, \mathbf{k}, \mathbf{j}, s]. \tag{4.8}$$

We now consider the error induced by throwing away a set of (s,p)'s at the n^{th} transition. In particular, let G_n be the set of (s,p)'s discarded at n^{th} transition. The bound for the error produced by discarding trajectories defined by (s,p)'s at n^{th} transition is then

$$\frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{(s,p)\in G_n} , \ [n, \mathbf{k}, \mathbf{j}, s]. \tag{4.9}$$

This gives us a bound on the error due to discarding (s,p)'s at a given transition. Since the probability calculation is recursive, the total error due to discarding trajectories for a given number of transitions must also include the error induced by discarding trajectories at previous transitions. The effect of error induced by throwing away trajectories in any transition on the subsequent transition is calculated as follows.

Suppose, with probability $p(a,b)$, process makes the $(n+1)^{th}$ transition by changing from state a to b. Then

$$, \ [n+1, \mathbf{k}, \mathbf{j}, b] = , \ [n, \mathbf{k}, \mathbf{j}, a] \ \times \ p(a,b) \tag{4.10}$$

Now, let total number of states be m. Since $\sum_{i=1}^{m} p(a,q_i) = 1$, therefore

$$, \ [n, \mathbf{k}, \mathbf{j}, a] = \sum_{i=1}^{m}, \ [n+1, \mathbf{k}, \mathbf{j}, q_i]. \tag{4.11}$$

This implies that the error propagated to the $(n+1)^{th}$ transition is equal to the error induced due to throwing away new (s,p)'s in n^{th} transition and the error propagated from the $(n-1)^{th}$ transition. Therefore, the error induced by discarding trajectories at n^{th} transition can be expressed as

$$E_P(n) \leq \frac{e^{-\lambda t}(\lambda t)^n}{n!} \left[\sum_{(s,p)\in G_n} , \ [n, \mathbf{k}, \mathbf{j}, s] + E_P(n-1) \right]. \tag{4.12}$$

For error bound over all the transitions, we have

$$E_P \leq \sum_{i=1}^{N} E_P(i). \tag{4.13}$$

Thus a lower bound on the probability distribution function of $Y_{[0,t]}$ can be expressed as

$$P[Y_{[0,t]} \leq y] \geq \sum_{n=0}^{N} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k} \in \eta_n} \sum_{\forall \mathbf{j} \in J_{\mathbf{k}}} P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] \sum_{\forall (s,p) \in S'_{\mathbf{k},\mathbf{j}}}, [n, \mathbf{k}, \mathbf{j}, s] \tag{4.14}$$

where $S'_{\mathbf{k},\mathbf{j}}$ is the set of (s, p) not discarded. The upper bound on the PDF can be computed by adding E_p and E_N to the lower bound.

CHAPTER 5

Implementation and Results

The method has implemented as a part of the large modeling package *UltraSAN* [3], based on the SAN modeling framework. In *UltraSAN*, a model of a system consists of a SAN representation and a reward structure, collectively defined as "SAN-based reward model" (SBRM). The package automatically generates the am-behavior and state-level reward structure from SBRM. This state-level representation serves input to a solver which implements the method developed in this paper. The solver comprises of 3700 lines of c language code. It provides options to specify parameters such as: interval of time (t), desired accuracy for depth truncation (a), weight to discard trajectories (w), range of the distribution (r), and the number of points to be computed (n).

The computation of higher order derivatives to calculate conditional PDF $P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}]$ runs into numerical accuracy problems. Also, the probabilities of trajectories grow smaller with each randomized step and loose numerical accuracy. For the conditional PDF we use floating point accuracy of 100 decimal digits while we throw away the trajectories with probabilities lower than the user defined weight (default is $1.00e-12$) and calculate bound on the injected error. The bound for default weight is normally in the order of $1.00e-10$ which is acceptable for most of the applications. All the results presented in this section are for a IBM RS6000 model with 64 megabyte of RAM.

Table 5.1: Activity Time Distributions

Activity	**Distribution**	**Case Probabilities**		
		1	2	3
$buffer\_failure$	expon(0.0001 * MARK(buffer))	0.95	0.05	0
$processor\_failure$	expon(0.005 * MARK(processor))	0.90	0.05	0.05
$processor\_repair$	expon(0.05)	1	0	0

Table 5.2: Output Gate Functions

Gate	**Function**
$G2$	$MARK(processor) = MARK(buffer) = MARK(repair) = 0$
$G1$	$MARK(processor) = MARK(buffer) = MARK(repair) = 0;$

The applicability of the proposed solution method is illustrated by considering the performability and availability analysis of a multiprocessor system with finite buffers taken from Sanders [25]. Figure 5.1 depicts the SAN model of the multiprocessor system. Here, places *processor* and *buffer* represent the number of fault-free processors and buffer stages, respectively, while the place *repair* indicates the number of processors queued for repair. Activities *processor_failure* and *buffer_failure* represent the occurrence of faults in the processors and buffer stages respectively. Activity *processor_failure* has three associated cases, representing three possible types of processor faults: 1) a repairable fault, 2) a fault causing total system failure, 3) a non-repairable processor fault. Activity *buffer_failure* has two associated cases: 1) a non-repiarable buffer failure, 2) a fault causing total system failure. Processor repairs are represented by activity *processor_repair*. The two output gates *G1* and *G2* implement the effect of a total system failure. Tables 5.1, and 5.2 represent the associated details about the exponential marking dependent rates of the activities, the case probabilities, and the gate functions, respectively.

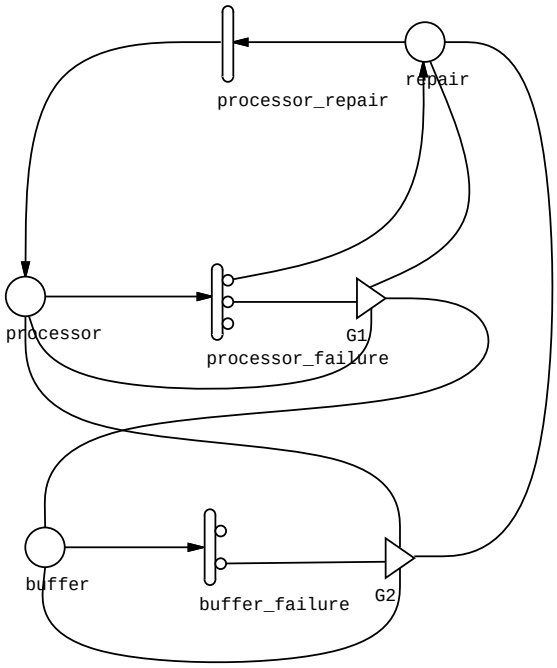


Figure 5.1: Example SAN

# Processors	*# Buffers*				
	1	2	3	4	5
0	.833	1.622	2.352	3.008	3.576
1	.968	1.859	2.656	3.338	3.891
2	.994	1.945	2.807	3.532	4.093
3	.999	1.978	2.888	3.658	4.232
4	1.000	1.997	2.934	3.744	4.334
5	1.000	1.999	2.961	3.805	4.412
6	1.000	1.999	2.977	3.850	4.474
7	1.000	1.999	2.986	3.883	4.524
8	1.000	1.999	2.992	3.909	4.566

Table 5.3: Throughput as Determined from Performance Submodel

Table 5.3, from [25], shows the throughput for different configurations of the multiprocessor system described above. The rate of accumulation of benefit due to completion of jobs in the system, for a given structure configuration, is obtained by multiplying the throughput in that state by a constant x. Costs associated with the processor repairs are presented in the reward structure by associating a reward of $-y$ with each completion of activity *processor_repair*. Under these assumptions, the probability distribution of the total profit associated with operating the system for some utilization period $[0, t]$ can be found by using the reward structure

$$C(a) = \begin{cases} -y & \text{if a} = \textit{processor\_repair} \\ 0 & \textit{otherwise} \end{cases}$$

$$R(\nu) = \begin{cases} x \times Thru(m,n) & \text{if } \nu = \{(B,n),(C,m)\} \\ 0 & \textit{otherwise} \end{cases}$$

Before formulating the availability measure, a definition of "system availability" must be given. In this regard, the system is defined to be available if there are at least m number of working processors and n working buffers in the system. Then availability can be formulated by using the reward structure as follows

$$C(\text{a}) = \begin{cases} 0, & \forall \quad a \in A \end{cases}$$

$$R(\text{v}) = \begin{cases} 1 & \textit{if there at least } m \textit{ processors and } n \textit{ buffers} \\ 0 & \textit{otherwise} \end{cases}$$

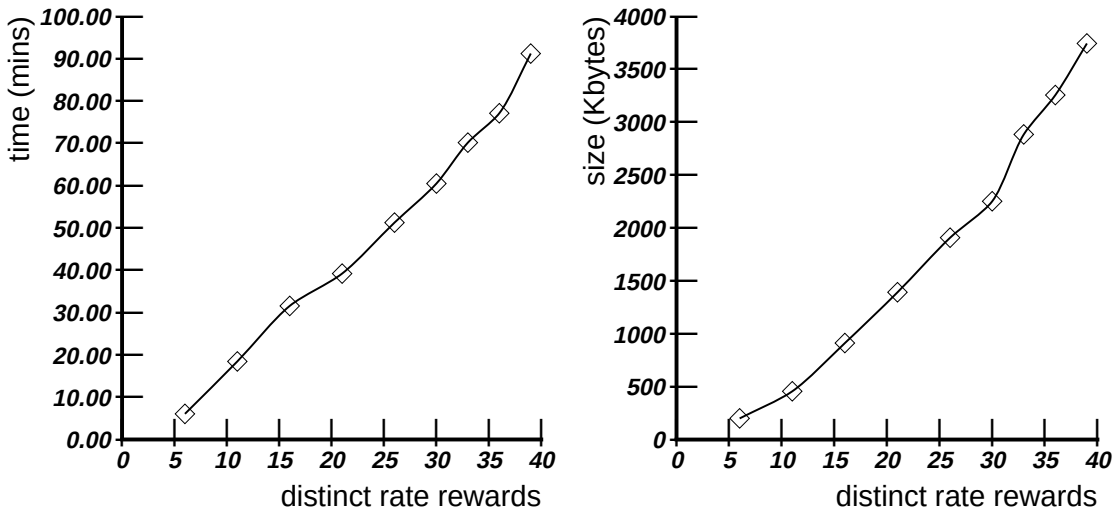


Table 5.4: Effect of Increase in Number of Distinct Rate Rewards (t = 10, a = 6, w = 12)

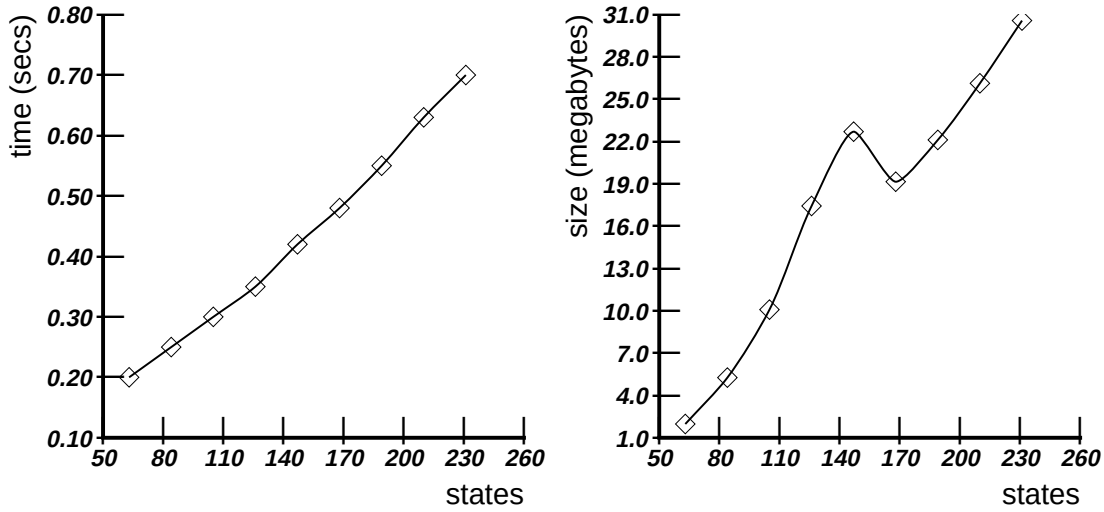
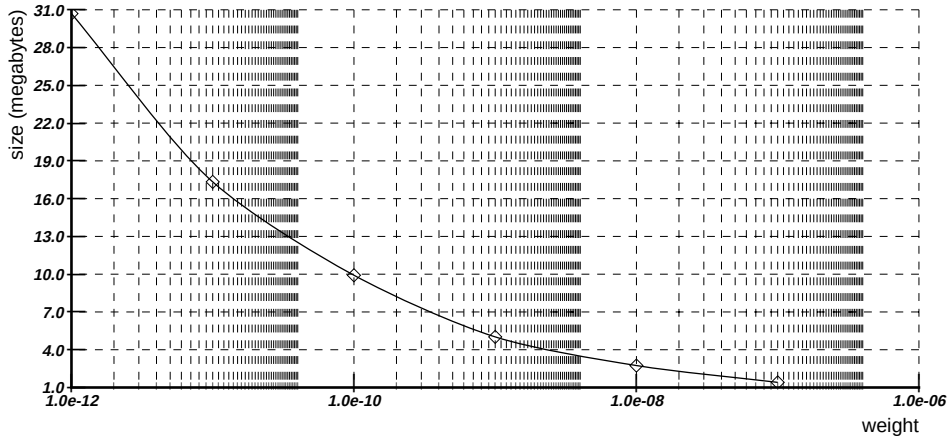
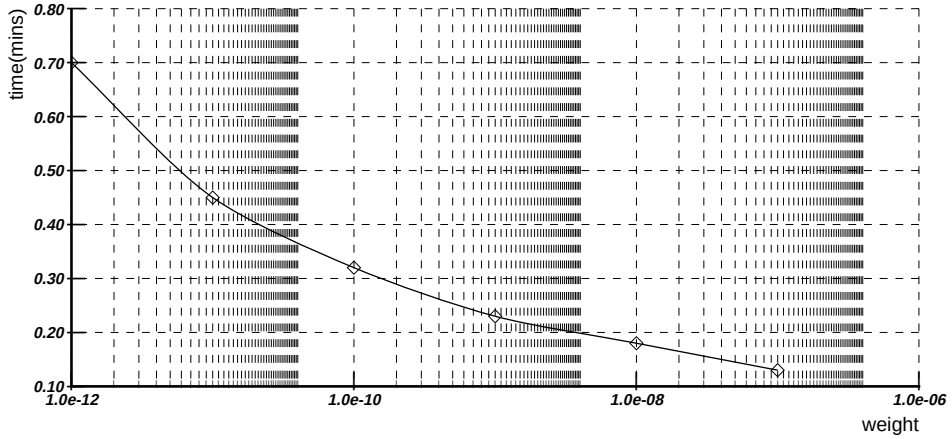


Table 5.5: Evaluation of the Availability Measure (t = 10, a = 6, w = 12)





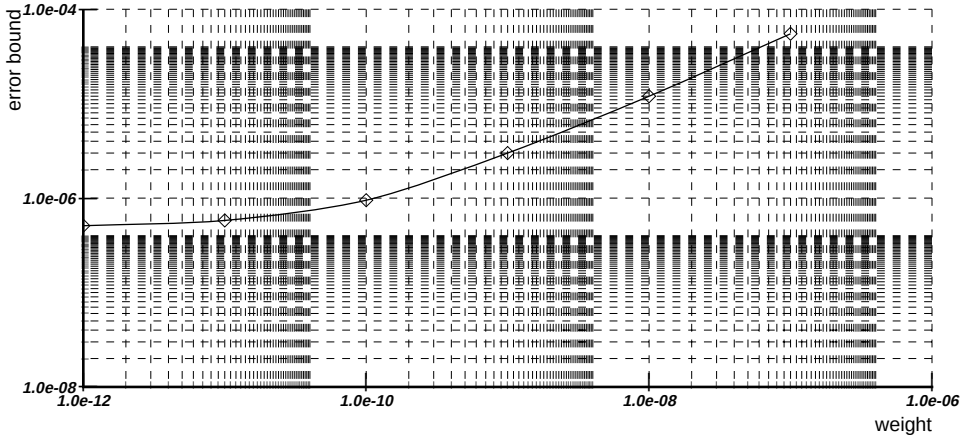


Table 5.6: Effect of Discarding Trajectories (t = 10, a = 6)

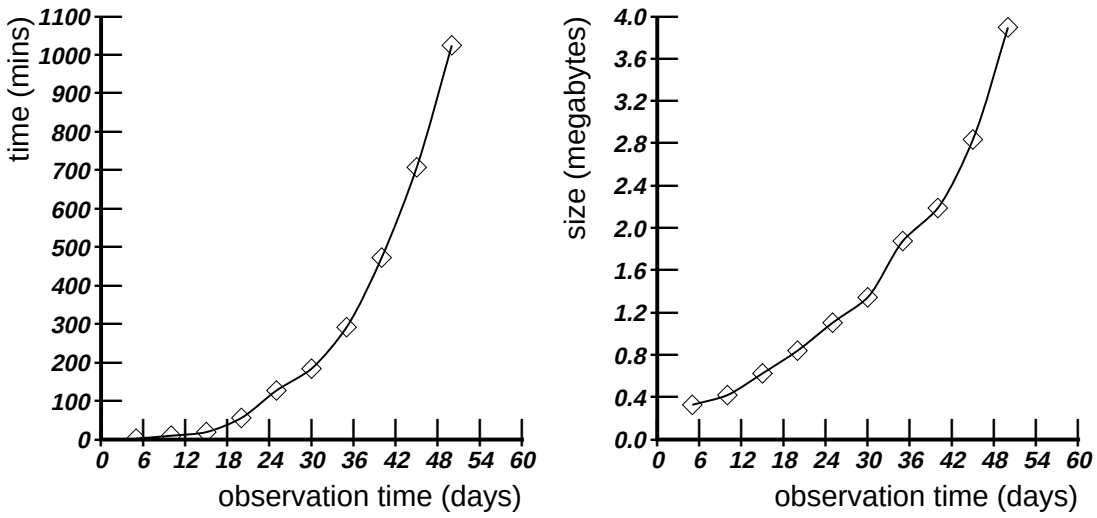


Table 5.7: Effect of Increasing the Time Observation (a = 6, w = 12)

The remainder of the section presents results for the availability and performability of the multiprocessor system to illustrate the effects of various parameters on the time and space complexity of the developed method.

The results presented in table 5.4 show the space and time required to calculate the distribution of total profit obtained from operating the system for 10 days when the number of buffers is varied. An increase in the number of processors and buffers corresponds to the increase in the number of different performances the system can exhibit. This implies the increase in number of different rate rewards in the reward model, which increases the number of elements of vector **k**. Therefore, at each level, there are more nodes to compute which increases the computation time and memory required to store a level to compute the next level in the trajectory graph.

The second set of experiments compute the distribution of availability for the modeled system. For availability measure, the increase in the number of combinations of processors

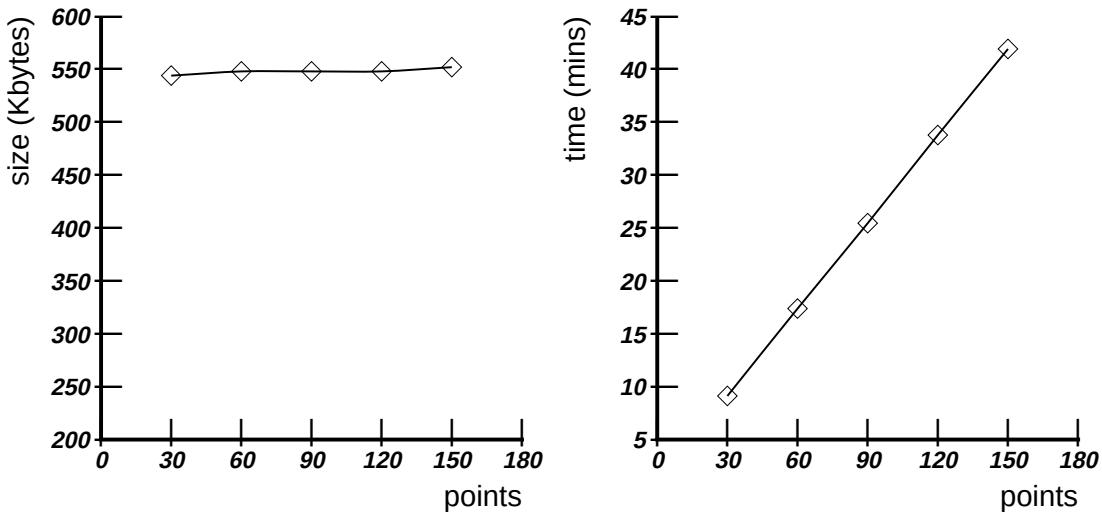


Table 5.8: Effect of Increase in Number of Points (t = 10, a = 6, w = 12)

and buffers only increases the number of states having a particular rate reward and not the number of distinct rate rewards. The number of distinct rate rewards remains fixed corresponding to availability and nonavailability of the system. In terms of trajectory graph, the number of nodes at each level remains the same while the number of elements (s, p)'s of set $S_{\mathbf{k}}$ increases. Table 5.5 shows that the increase of states keeping the distinct rate rewards fixed significantly increases the required memory for the process. The increase in time to evaluate the distribution is negligible as number of nodes remains same.

The third experiment shows how the scheme of discarding trajectories can reduce the space and time required to compute the distribution. The weight provided by the user is the criterion used for discarding trajectories. In table 5.6 we varied the weight and observed its impact on the error bound, process size, and cpu time while evaluating the distributions of interval availability. The results show that a reduction in weight increases the number of discarded trajectories, drastically reducing the size and the cpu time of the process.

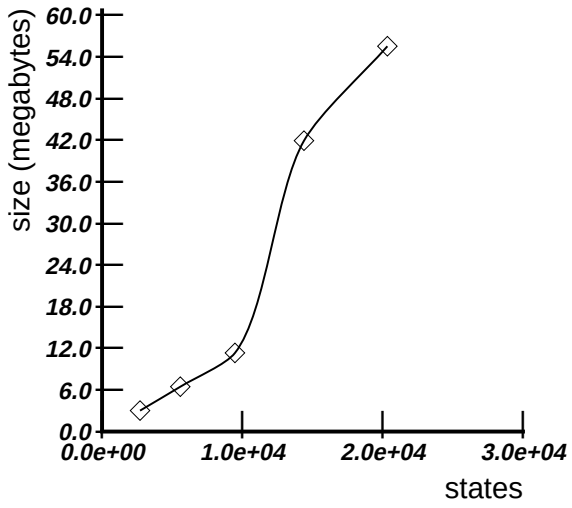


Table 5.9: Results for Large State Spaces (t = 10, a = 3, w = 5)

For different weights, the error bounds are calculated, illustrating the effectiveness of the scheme.

The fourth experiment investigates the effect of the value of the minimum holding time of the Markov process, or equivalently, the length of the observation interval, on the time and space complexity. Low values of minimum holding times and large observation intervals require generation of more transition steps of the randomized process for a specified error bound. An increase in the number of transitions required increases the time require to calculate the performability distribution. Also, at a higher level the number of generated nodes will increase and results in the increase of process size. Table 5.7 illustrates the relation of an increase in the observation time interval to the required time for computation and the process size.

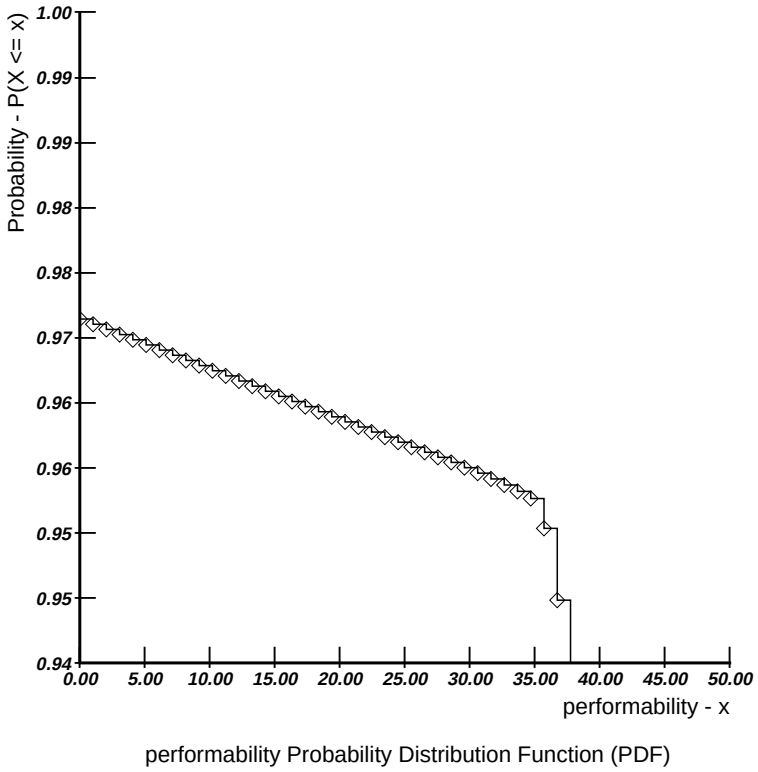


Figure 5.2: Probability Distribution (t = 10, a = 6, w = 12)

Sometimes, it is desired to compute the distribution for more than one reward value (point). A typical example may be a plot of a distribution graph, figure 5.2. Calculating distributions for more values increases the computation time, but not significantly. Table 5.8 illustrates the results.

All the above results are evaluated to a very high accuracy. In particular, except for table 5.6 a weight of 12 for discarding certain trajectories was used. The required accuracy for the probability distribution depends upon the underlying nature of the model and the variable that is evaluated. For certain systems an accuracy of two decimal points is sufficient. To show how big of a process the tool can estimate the PDF for, the distributions for large state spaces of the multiprocessor model is evaluated with an accuracy of two decimal points and an interval of 10 days. As can be seen from table 5.9, very large problems can be solved if many significant digits are not necessary.

A typical distribution for 5 *processors* and 2 *buffers* with the parameters defined in table 5.3 is illustrated in figure 5.2.

In summary, the performability solver can solve both repairable and non-repairable systems. It provides options to the user to overcome, to an extent, the inherent problems of space and time complexities. Of course the problems is dealt by reducing the accuracy of the solution, but the option is given to the user to define desired accuracy.

CHAPTER 6

Conclusions

The objective of the study was to develop an efficient method for solving reward models specifying performability for both repairable and non-repairable systems. The objective was achieved by:

- Using a generalized reward structure, where both impulse and rate rewards are considered.
- Assigning rewards at the network level rather than the state level, as the state assignment limits the scope of the systems that can be evaluated.
- Developing a mechanism to expand a Markov Renewal process to a Markov process, without loosing information concerning self-transitions.
- Truncating the infinite summation of the randomized equation to a certain step to avoid memory and time complexity, and calculating bound on the error produced.
- Discarding certain paths in the computation to avoid the memory explosion. Calculating bound on the error induced by discarding paths.
- Implementing the solution method in form of a software tool.

- Illustrating the usefulness of the tool by considering the evaluation of a simple multiprocessor system.

Much time was spent on developing the equation (3.9), based on randomization, to compute the specified performability measure of a system using the generalized reward structure. The most challenging problems related to use of randomization to solve generalized reward models were: 1) Developing a technique to transform the reward model into a Markov process which holds both impulse and rate rewards, 2) Keeping track of impulse reward accumulated for every possible generated path, 3) Deriving algorithms to compute bounds on the error produced by discarding certain path trajectories.

The implementation of the performability evaluation algorithm exposes to certain interesting computational problems. The traditional ones are the time and space complexities. Error algorithms are designed to overcome time and space complexities. Path probabilities decrease drastically with increase in number of transitions. This causes underflow problems which results in numerical inaccuracies. We used a multiprecision arithmetic tool to keep a precision of 100 decimal digits. This slows the tool performance but solves numerical inaccuracies.

Performability tool provides options to the users which help them to control parameters such as time interval, weight, accuracy, and the desired number of result points within the user specified range. These options enable users to control the time and space required to evaluate a system.

6.1 Area of Further Research

The theory of reward structure solution is under development and there are a number of interesting areas for investigation. One such area is the development of a reward assignment technique to specify the performability variable that would result in a reduction in the number of states which must be considered.

The tool can also be improved by implementing an application oriented memory allocator.

Appendix A

Implementation in *UltraSAN*

A.1 Data Structures

Following are the data structures utilized in the performability solver. The first data structure stores the matrix elements. Each structure represents a matrix row. The source code for this data type is represented as:

```
/* STRUCTURE TO CONTAIN ONE ROW OF THE MATRIX

   row_element - points to the first element in the row
   column_index - points to the column index of the first element
   length - the allocated length of row_element and column_index
   num_zeros - the number of unused elements in the row
   next_row - structure pointer to the next row in the matrix, will
     point to NULL for the last row
*/
struct matrix
{
   double *row_element;
```

```
   unsigned long *column_index;
   unsigned long length;
   unsigned long num_zeros;
   struct matrix *next_row;
};
```

The second data type stores the performance variable information. Each structure represents one performance variable. The source code for this data type is represented as:

```
/* REWARD STRUCTURE - One structure is needed per performance variable.
   pvname - points to the name of the performance variable
   impulse - points to an array of impulse rewards
   rate - points to an array of rate rewards
   next - points to another reward structure
   The length of the arrays is the total number of states
*/
struct reward
{
   char *pvname;
   double *impulse;
   double *rate;
   struct reward *next;
};
```

The third data type stores the states with the associated rewards, impulse and rate. These states are arranged in the descending order of rate rewards. The source code for this data type is represented as:

```
/* STATE INFORMATION - States are arranged according to the rate rewards.
   state - state number
   impulse - impulse reward assigned to the state
   rate - rate reward assigned to the state
   Structure contains the information of the perfvar in consideration
*/
struct arrangeRew
{
   unsigned long state;
   double impulse;
   double rate;
};
```

The fourth data type stores the expanded state space. Each structure contains its state number and its expanded state number if it has been expanded. The source code for this data type is represented as:

```
/* EXPANDED STATE SPACE STORAGE - structure is used to expand  the
   renewal process to Markov process without losing the information.
   state - state which has self transition in the renewal process.
```

```
   expstate - the new included state.
   rate - rate in or out from or to another state.
*/
typedef struct exp_storage
{
   unsigned long state;
   unsigned long expstate;
   double rate;
} expstorage;
```

The fifth data type stores the information about states which have same rate reward. Each structure stores the rate reward, the number of states with the rate reward, and a pointer to an array of the states. The source code for this data type is represented as:

```
/* REWARD DISTRIBUTION over the states.
   rate - different rates
   states - no. of states for each rate
   ptr - array containing states with same rate and diff. impulses.
*/
typedef struct stateRewDist {
   double rate;
   unsigned long states;
   impulseArrEle *ptr;
```

```
   } stateRewDist;
```

The sixth data type defines the element of the array pointed by the structure of type "stateRewDist" defined above. Each structure contains the state number, the impulse reward associated with entering the state, and the probability of reaching that state. The source code for this data type is represented as:

```
/* SAME RATE DIFFERENT IMPULSE - element of an array which contains
   the information about states with same rate rewards and different
   impulse rewards.
   state - state number
   impulse - impulse reward accumulated by reaching that state
   probability - probability of following the path
*/
typedef struct impulseArrEle {
   unsigned long state;
   double impulse;
   double probability;
   } impulseArrEle;
```

The seventh data type stores the information about each of the color sequence generated. Each structure contains a particular sequence, number of paths followed to the sequence, and a pointer to an array containing information about each of the path traversed. The source code for this data type is represented as:

```
/* SEQUENCE AND PATHS FOLLOWED - Each element is for a particular
   color sequence.
   sequence - particular sequence
   noPathDep - number of paths resulting in the sequence
   pathProbEle - array containing info about each of the path
typedef struct cellstruct {
   int *sequence;
   unsigned long noPathDep;
   pathProbEle *ptr;
   } cellStructEle;
```

The eighth data type stores the information about the paths followed to generate a particular sequence. Each structure contains the state last reached in following a particular path, and the probability of reaching that state and the impulse reward accumulated along the path. The source code for this data type is represented as:

```
/* PATH PROBABILITIES AND IMPULSE REWARDS - It contains the path
   probability and the last state reached following the path.
   state - last state reached in a particular path
   probabilities - probability of reaching the state
   impulse - impulse reward accumulated following the path
typedef struct pathProb {
   unsigned short state;
```

```
double probabilities;

double impulse;

} pathProbEle;
```

A.2 Process Flow

The following description of procedures suggests the process flow of the solver.

It also describes the call relationships between procedures.

Procedure: *main*

Read the command line arguments.

Calculate the points of distribution by computing -n[] points between the reward range -r1[] to -r2[].

call procedure *test_solv*.

Procedure: *test_solv*

Read in the transition matrix.

Subordinate the Markov process to a Poisson Process.

Compute the truncation point for the time interval of interest. The -t option specifies the time interval, and the -a option specifies the desired accuracy on the basis of which the number of iterations of the randomized process is calculated.

For each iteration

Call procedure *perfcalc*

Write results in the output file *struct.result*.

Procedure: *perfcalc*

Calculate the possible state trajectories (paths) and the resulting color sequences. Transition-rate matrix determines the probability of following a path. If the probability of a path is less than the weight

specified by -w option then ignore that path.

Compute impulse reward accumulated for each state trajectory.

For each color sequence
For each path
Call procedure *condDist*

Multiply the conditional distribution with the probability of the path.

Sum up for the paths in a given color sequence.

Sum up for the color sequences in the given iteration.

Procedure: *condDeriv*

For each color of the sequence greater than one.

Call procedure *derivative*

Procedure: *derivative*

Calculate the k^{th} derivative.

A.3 Reference Manual

NAME

pdf - probability distribution solver for Ultra*SAN*

SYNTAX

pdf -Pproject -ttime -r1range -r2range [options]

DESCRIPTION

pdf obtains the probability distribution of the specified performability variable(s) for the generated reduced base model (created previously using Gen(1))

using randomization (also known as uniformization). It calculates the probability distribution function of interval-of-time variables. The probability distribution files are given in a format compatible with splot(1). The splot files are found in the directory project/int, named with the extension ".splot". For technical informa- tion concerning the solver, see the reference below.

OPTIONS

-P project

Is a character string naming the project to be solved. A name must be specified. No default is assumed.

-t time

Is a double representing the desired interval of observation. An interval has to be specified. No default is assumed.

-aaccuracy

Is a short integer representing the number of digits to the right of the decimal point that are desired to be accurate. Six digits is the default. If the number of digits specified exceeds the machine accuracy, then the machine accuracy becomes the default.

-d filename

Is the file where debug information will be stored (i.e. project/results/-filename.debug). If -d is specified with no filename, then project/results/project.debug is assumed. If -d is not specified, no debug information is generated.

-w weight

Is the scale to instruct the solver to discard paths with probabilities less than the specified weight. Default does not discard any paths.

-r1 range

Is a double and specifies the lower range between which probability distribution is evaluated. Used only with option r2. Cannot be used alone or with option r.

-r2 range

Is a double and specifies the upper range between which probability distribution is evaluated. Used only with option r1. Cannot be used alone or with option r.

-r range

Is a double and specify the upper range between which probability distribution is evaluated. In this case lower limit is 0. Used alone. Either r or r1 and r2 are necessary to specify.

-n points

Is an integer and specifies the number of points between the range.

Default consider only two points, lower and upper limits.

M. A. Qureshi, Reward Model Solution Methods with Impulse and Rate Rewards: An Algorithm and Numerical Results, Masters Thesis, Dept. of Electrical and Computer Engineering, University of Arizona, May. 1992.

REFERENCES

[1] G. Balbo, G. Chiola, G. Franceschinis, and G. Molinar, *Generalized Stochastic Petri Nets for the performance evaluation for FMS.*

[2] G. Ciardo, J. Mupallo, and K. S. Trivedi, *"SPNP: Stochastic Petri Net Package,"* Proc of the third International Workshop on Petri Nets and Performance Models., 1989, pp 142-151.

[3] J. Couvillion, R. Friere, R. Johnson, W. D. Obal, M. A. Qureshi, M. Rai, W. H. Sanders, and J. E. Tvedt, *"Performability modelling with UltraSAN,"* IEEE Software, Vol. 8, No. 5, Sept. 1991.

[4] H. A. David, *"Order Statistics,"*. 2nd Wiley, New York, 1981.

[5] A. P. Dempster, and R. M. Kleyle, *"Distributions determined by cutting a simplex with hyperplanes,"* Ann. Math. Stat., Vol. 39, No. 5, 1968, pp. 1473-1478.

[6] L. Donatiello, and B. R. Iyer, *"Analysis of a composite performance reliability measure for fault-tolerant systems,"* JACM., Vol. 34, No. 1, Jan. 1987, pp 179-199.

[7] E. de Souza e Silva, and H. R. Gail, *"Calculating availability and performability measures of repairable computer systems using randomization,"* JACM, Vol. 36, No. 1, January 1989, pp. 171-193.

[8] E. de Souza e Silva, and H. R. Gail, *"Calculating cumulative operational time distributions of repairable computer systems,"* IEEE Trans. Comput. C-35, No. 4, April 1986, pp. 322-332.

[9] D. G. Furchgott, and J. F. Meyer, *"A performability solution method for degradable non-repairable systems,"* IEEE Trans. on Computers, Vol. C-33, No. 6, June 1984.

[10] A. Goyal, and A. N. Tantawi, *"Evaluation of performability for degradable computer systems,"* IEEE Trans. on Computers, Vol. C-36, No. 6, June 1987, pp 738-744.

[11] W. Grassman, *Transient solutions in Markovian queues,"* J. of Oper. Res. 1, 1977, pp 396-402.

[12] D. Gross, and D. R. Miller, *The randomization technique as a modelling tool and solution procedure for transient Markov processes,"* Op. Res., Vol. 32, No. 2, 1984, pp. 343-361.

[13] B. Haverkort, *Performability modelling tools, evaluation techniques and applications,"* PHD dissertation, Universiteit Twente, Netherland, Jan. 1990.

[14] R. A. Howard *Dynamic Probabilistic Systems.* Vol 11: Semi Markov and Decision Process, New York, Wiley, 1979.

[15] B. R. Iyer, L. Donatiello, and P. Heidelberger, *"Analysis of performability for stochastic models of fault-tolerant systems ,"* IEEE Trans. on Computers, Vol. C-35, No. 10, Oct. 1986, pp 902-906.

[16] L. Kleinrock, *Queueing systems.* Vol. 1, New York, John Wiley, 1975.

[17] V. G. Kulkarni, V. F. Nicola, R. M. Smith, and K. S. Trivedi, "*Numerical evaluation of performability and job completion time in repairable fault-tolerant systems,"* Proc. 16th International Symp. on fault-tolerant computing, pp. 252-257, Vienna, Austria, July 1985.

[18] J. F. Meyer, *"On evaluating the performability of degradable computing systems,"* IEEE Trans. Comput. Vol. C-22, Aug. 1980, pp. 720-731.

[19] J. F. Meyer, *"Closed-form solutions of performability,"* IEEE Trans. Comput. Vol. C-31, Jul. 1982, pp. 648-657.

[20] M. Molloy, *"On the integeration of delay and throughput measure in distributed processing model,"* PHD dissertation, Univ. Calif., Los Angeles, 1981.

[21] A. Movaghar, *"Performability modelling with stochastic activity networks,"* Dissertation, University of Michigan, 1985.

[22] A. Movaghar, and J. F. Meyer, *"Performability modelling with stochastic activity networks,"* In proceedings Real-Time Systems Symp., Austin, Tx, December 1984.

[23] S. Natkin, *"Reseaux de Petri Stochastiques,"* These de Docteur-Ingenieur, CNAN., Paris, June 1980.

[24] M. Rai, *"Design and implementation of a reduced base model construction technique for stochastic activity networks,"* M.S. thesis, University of Arizona, 1990.

[25] W. H. Sanders, *"Construction and solution of performability models based on stochastic activity networks,"* Dissertation, University of Michigan, 1988.

[26] W. H. Sanders, and J. F. Meyer, *"Reduced based model construction for Stochastic Activity Networks,"* IEEE Journal in selected areas in Communications, Vol. 9, No. 1, Jan. 1991.

[27] W. H. Sanders, and J. F. Meyer, *"A unified approach for specifying measures of performance, dependability, and performability,"* Dependable computing for critical applications, Vol. 4, Springer-Verlag, 1991.

[28] K. Trivedi, A. Reibman, and R. Smith, *"Transient analysis of Markov and Markov reward models,"* Computer Performance and Reliability, 1988, pp 535-544.

[29] H. Weisberg, *"The distribution of linear combinations of order statistics from the uniform distribution,"* Ann. Math. Stat., Vol. 42, No. 2, 1971, pp. 704-709.