From *Performance Evaluation Journal*

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

Muhammad A. Qureshi and William H. Sanders
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721 USA

qureshi@ece.arizona.edu and whs@ece.arizona.edu

ABSTRACT

Reward models have become an important method for specifying performability models for many types of systems. Many methods have been proposed for solving 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 uniformization. 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, we propose a new technique for discarding paths in the uniformized process whose contribution to the reward variable is minimal, which greatly reduces the time and space required for a solution. A bound is calculated on the error introduced by this discarding, and its effectiveness is illustrated through the study of the performability and availability of a degradable multi-processor system.

Keywords: Markov reward model, performability, stochastic activity networks, stochastic Petri nets, uniformization.

Short Title: Reward Model Solution Methods.

This work has been supported, in part, by a contract from US West Advanced Technologies and the Digital Faculty Program: Incentives for Excellence.

I Introduction

In recent years, growing dependence on computing systems has caused combined performance and dependability evaluation to become an increasingly important activity. Often called "performability" evaluation [12], this activity strives to predict the "bottom-line" performance of a computing system, accounting for possible degradations of performance due to faults. Many methods have been proposed to evaluate system performability, but one of the most popular has been through the use of "reward models."

Informally, a reward model consists of a stochastic process, a reward structure, and a performance variable defined on the structure [10]. Typically, the stochastic process is taken to be a discrete-state, continuous-time Markov process. The reward structure defines the "benefit" derived by operating the system for some period of time, through the use of functions. Generally speaking, these functions take two forms: *rates* of reward that are accumulated while in particular states of the process, and *impulses* of reward that are accumulated when particular transitions in the process occur.

The reward structure quantifies the behavior of the system, but does not specify the utilization interval over which the reward is to be accumulated. The type of variable specified does this. Broadly speaking, variables can be categorized as: instant-of-time variables, interval-of-time variables and time-averaged interval-of-time variables [18]. Instant-of-time variables represent the status of a process at either a particular time or in steady state. Interval-of-time variables represent the total or time-averaged reward (relative to a particular reward structure) accumulated during some fixed (non-zero length) interval.

This paper focuses on determining the probability distribution function of interval-of-time variables. Much work has been done in this regard, but has been limited either in the classes of stochastic processes or the types of reward structures considered. Most early work, in the context of computer system evaluation, has been limited to acyclic Markov reward models. For example, Meyer [13] considered a specific 2-processor multiprocessor without repair, and derived an explicit expression for the PDF of reward accumulated over a fixed utilization interval. Subsequently, Furchtgott and Meyer [7] developed a method to obtain the PDF of an interval-of-time variable, where rate rewards are associated to states in a manner such that the rate of accumulated reward does not increase during a utilization interval (called "non-recoverable" in [7]). This method was very general, in the sense that it could handle non-exponential holding times in states, but computationally very expensive,

since it relied heavily on numerical integration.

Later work restricted itself to the case of exponential holding times, and acyclic processes, but was computationally less demanding. In particular, Donatiello and Iyer [4] derived a closed-form expression for acyclic systems with distinct rate rewards using Laplace-Stieltjes transforms. Goyal and Tantawi [8] derived a similar expression, but used time-domain arguments. Ciciani and Grassi [1] derived a third result and applied the result to the performability evaluation of satellite networks. These results were significant in that they provided a computationally efficient solution, but were limited to certain classes of rate rewards (e.g., non-recoverable, or distinct) and to acyclic Markov processes.

Relatively little work has been done to obtain the PDF of interval-of-time variables defined on possibly cyclic Markov processes. Cycles in the process make the solution much more difficult, since the transform techniques yield expressions that do not have a simple inverse transform, and the time domain techniques require a finite number of paths in the process. Notable, however, is the work of Kulkarni et al. [11] which considers possibly cyclic Markov reward models with rate rewards, and numerically inverts the expression obtained in the transform domain to obtain a solution. Later Smith et al. [20] presented an improved version of the algorithm in [11], with a computational complexity of $O(n^3)$, where n is the number of states.

Also notable is the work of de Souza e Silva and Gail [5, 6] who developed a method, based on uniformization, that is suited for possibly cyclic processes and more general reward structures, through the use of the concept of "coloring." Uniformization, which has previously been exploited to obtain transient state occupancy solutions for Markov processes [9], permits exploration of only finitely many of the possible paths of the process when computing a solution. de Souza e Silva and Gail extended this method to Markov reward models and interval-of-time variables. Their reward structure is different than considered previously, in that it makes uses of distinguished events in the process, which are "colored." A vector is associated with each path considered, where each element in the path corresponds to the number of events of a certain color. The interpretation given to the events, relative to the performance variable, can vary. Specifically, de Souza e Silva and Gail consider two general types of variables: those which depend on the number of events of each color (impulse rewards, in the terminology introduced earlier), and those which depend on the time spent in states resulting from events of each color (rate rewards). This work both represented a significant step forward in the description and solution of Markov

reward models, and suggested areas that needed further study.

In particular, the use of a single coloring vector allowed the consideration of either impulse or rate rewards, but not both in a single model. Furthermore, the rewards were defined on the uniformized process, not the original, so self-transitions in the original process (if they existed) could not be distinguished from those added by the uniformization procedure with respect to impulse rewards. Additionally, the method required exploration of a very large graph of paths, since truncation was only done based on the length of the path. While the authors suggested that the space required to do this may be prohibitive for realistically-sized systems, no experimental results were given showing the space and time complexity of the approach. Finally, description of the systems and reward structures considered was done at the state level, which can be difficult for models of realistically-sized systems.

We have addressed each of these issues in this paper. In particular, we have used SAN-based reward models [17] as our description method, which allow specification of both impulse and rate reward at the network level, rather than uniformized process level. Furthermore, we have extended the development in [5] to make use of two coloring vectors, and hence permit the solution of impulse and rate rewards in a single model. (Note that others have considered solution of models with impulse and rate rewards in a single model for interval-of-time variables, but to the best of our knowledge, only to obtain the expected reward. See, for example, [15]). In addition, we have given an explicit algorithm for exploration of the path graph, and proposed a method for discarding specific sets of paths based on their contribution to the performance variable. Bounds are given on the error introduced by discarding such paths, and experimental results are presented that show that the discarding policy can dramatically reduce the space and time required for solution while achieving acceptable accuracy. The time and space complexity of an implementation of the developed algorithm is then investigated through the solution of a degradable multiprocessor system with repair.

II SAN-Based Reward Models

While the solution method could be used with any model description formalism, we use SAN-based reward models, [16, 17, 18], due to their ability to compactly represent Markov reward models, and the availability of software to automatically perform this conversion.

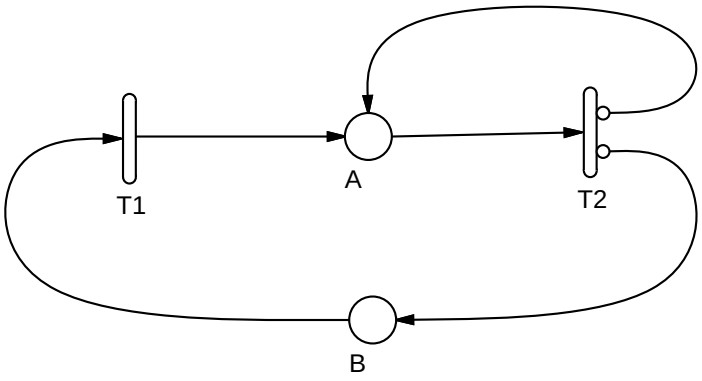


Figure 1: Example SAN

Table 1: Activity Time Distributions of Example SAN

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

A SAN-based reward model (see above references for more details) consists of two parts: 1) a stochastic activity network representation of the system in question, and 2) a reward structure defined on the SAN. The reward structure is defined directly in terms of events ("activity completions") that occur in the SAN and time spent in states that result from such events (numbers of "tokens" in places).

An example stochastic activity network is illustrated Figure 1. A and B are places and with their collection of tokens represent the system's states. *T1* and *T2* are timed activities, *T1* with a single case and *T2* with two cases. Cases represent choices about what happens when an activity completes. Time in a stochastic activity network is represented by associating activity time distributions with *T1* and *T2* as depicted in Table 1.

Performance and dependability variables are represented using a reward structure. The reward structure is similar to that of a Markov reward model, but is defined on the SAN, rather than the underlying stochastic process [18]. The reward structure defined on the SAN consists of two functions: a function $\mathcal{C} : A \rightarrow \mathbb{R}$ which assigns to each activity in the SAN an impulse reward that is obtained when the activity completes, and a function $\mathcal{R} : \mathcal{P}(P, \mathbb{N}) \rightarrow \mathbb{R}$ which assigns rates of rewards to particular numbers of tokens in some

set of places. The notation $\mathcal{P}(P, I\!N)$ refers to the power set of the set of places P of the SAN cross the natural numbers. Hence, reward rates can be associated with particular markings of each subset of the places of the SAN. Interval and instant-of-time variables are then defined in term of this reward structure, just as they are for Markov reward models.

Different stochastic processes can be (automatically) constructed from a stochastic activity network, depending on the type of variables considered and the solution method employed [17]. The most common behavior for SANs and other stochastic Petri nets is the "marking behavior," or *m-behavior*, which can be taken to be a stochastic process $\{R_n, T_n : n \in N\}$ where R_n is the marking of the SAN after the *nth* timed activity completion and T_n is the time of the *nth* activity completion. This behavior suffices for many variables, but not all, since impulse rewards depend on activity completions, and a transition between particular markings may be caused by several activity completions. Another behavior, known as the "activity-marking behavior," or *am-behavior*, can distinguish between such activity completions, by using a notion of state that includes the name of the most recently completed activity as well as the marking. In every way, the am-behavior is analogous to the m-behavior, except that a more refined notion of state is used.

Both notions of state (marking and activity-marking) often lead to extremely large state-space representations, and ways to directly generate smaller (lumped) processes have been devised. These "reduced" base models form the starting point for the developed Markov reward model solution technique, and can be automatically generated from a SAN-based reward model, using the method described in [17]. The procedure produces a Markov renewal process, where the time between state changes is exponential, and a state-level reward structure, where rate rewards are assigned to states, and impulse rewards are assigned to entrances to states.

Before a method to solve the reduced base models and their associated reward structures using uniformization can be devised, a transformation must be made to the process. The transformation is required due to the uniformization process itself, which adds self-transitions to many states in the stochastic process. The generated reduced base models also can have self-transitions, since activities may exist whose completions may not result in a state change. These transitions need to be distinguished from the transitions that are an artifact of uniformization, since they may result in impulse rewards.

This can be done by converting the constructed Markov renewal process to a (continuous-time) Markov process, without any self-transitions. Any self-transitions added in the uni-

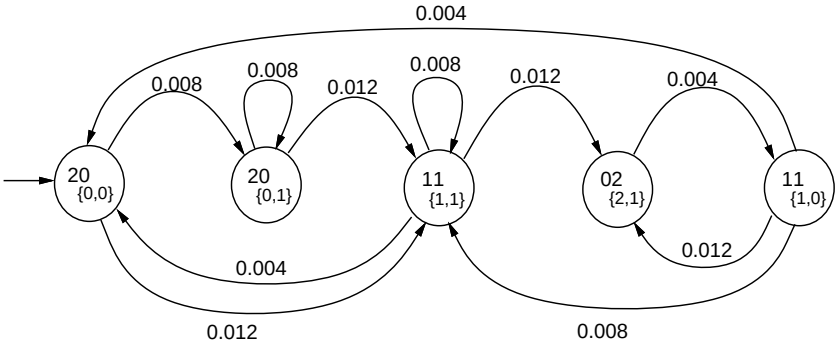
formization process can then be distinguished, and assigned an impulse reward of zero. The procedure to convert the process is as follows:

1. Replace each state with state-transitions by two states, which inherit the reward structure and all the outgoing transitions from the original state.
2. Assign to one of the new states the incoming transitions of the original state.
3. Assign the self-transition rate of the original state to a pair of transitions between the new states.

To illustrate this procedure, consider the SAN of Figure 1 with an initial marking of 2 tokens in place A and 0 tokens in place B. Furthermore, define a reward structure by associating an impulse reward of 1 with the completion of activity $T2$ and some marking dependent rate reward with the place B, such that the rate is equal to the number of tokens in the place. Figure 2 depicts the reduced base model (Markov renewal process) for the SAN, and the continuous-time Markov process constructed using the above rules. Each transition is labeled with its transition rate and each state is labeled as $AB_{\{r,i\}}$ where AB is the marking of places A and B respectively, in the state, and r and i are the rate and impulse rewards associated with the state. The resulting representation is simply a Markov reward model, and hence can be used as input to our algorithm to compute the distribution of reward accumulated during some interval of time.

III Solution by Uniformization

We thus seek methods for determining the distribution of reward accumulated during some interval of time, given a Markov reward model with reward rates that depend on states, and impulses of reward that occur on entrances to states. (As will be seen below, the solution method works equally well for reward models where impulse rewards depend on transitions between states, but the construction method defined above produces models where the dependence is only on the state entered.) The proposed algorithm is based on the well-known method of uniformization, which is briefly described below. In particular, consider a continuous-time discrete state Markov process $\{X_t : t \geq 0\}$, with generator matrix A and initial state distribution π_0, where one is interested in the state-occupancy probabilities at a particular time t. Let λ be greater than or equal to the maximum departure



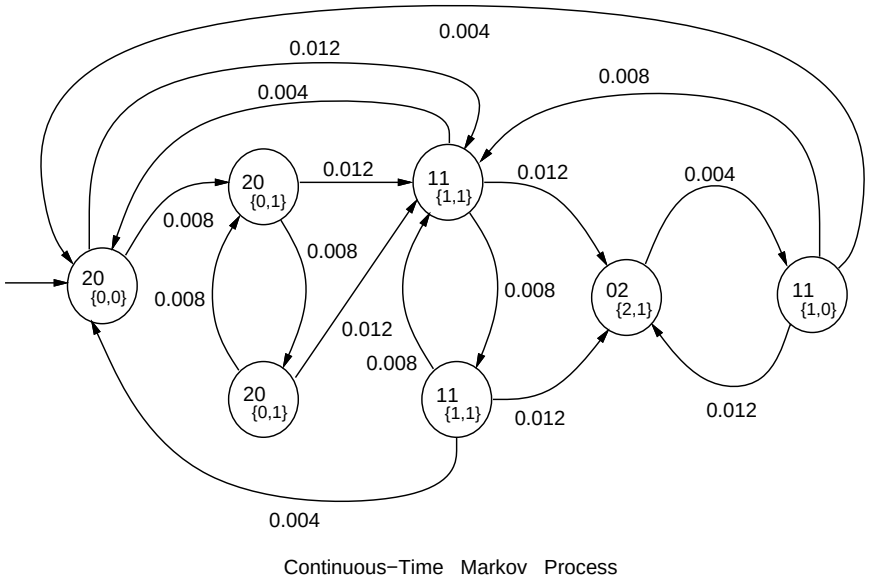


Figure 2: State Expansion

rate from any state in the process, and $\{Z_n : n \in \mathrm{N}\}$ be a discrete-time Markov chain defined on the same state space as $\{X_t : t \geq 0\}$ and with single step transition matrix $P = \frac{A}{\lambda} + I$. Furthermore, let $\{N_t : t \geq 0\}$ be a Poisson process with rate λ. Then, π_t, the row vector of state occupancy probabilities at time t can be expressed, as in [9], for instance, as

$$\pi_t = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \pi_0 P^n$$

where π_0 is the initial state occupancy probability vector.

The solution for the distribution of accumulated reward can be formulated in a similar manner, but is expressed in terms of the PDF conditioned on the number of transitions with particular impulse rewards and the number of entrances to states with particular rate rewards that can occur during the time interval. In particular, the continuous-time Markov process is first uniformized, using the above method. Impulse rewards are assigned to transitions between distinct states equal to the impulse awarded for entering the destination state in the continuous-time Markov process. Self-transitions (created by the uniformization process) are assigned an impulse reward of zero. Rate rewards are assigned to states which are identical to corresponding states in the continuous-time Markov process.

For example, the uniformized process corresponding to the Markov reward model of Figure 2 is given in Figure 3. For matter of convenience, we rename states of the process as follows: $20_{\{0,0\}}$ 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. The process consists of seven states with three distinct rate rewards and two distinct impulse rewards (including transitions with zero impulse reward). The rate (λ) of the Poisson process is taken to be 0.024. The transitions in Figure 3 are labeled according to their transition probability and, in parenthesis, the impulse reward that is accumulated when the transition occurs. Note that, as prescribed by the rule above, all self-transitions added by the uniformization procedure are assigned a zero impulse reward.

A Path Vectors

Consider now a process constructed in the above manner, with $K+1$ distinct rate rewards and J distinct impulse rewards. Consider a path of the process during some interval of time. The intervals between the transitions correspond to sojourn times in states of the Markov chain Z randomized by the Poisson process N. Each path thus consists of $n+1$ sojourn

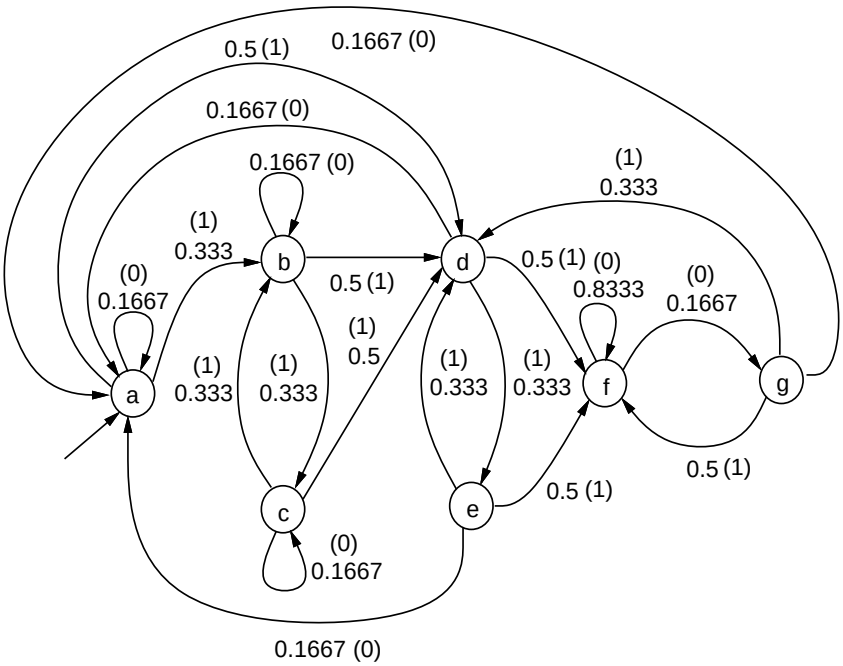


Figure 3: Uniformized Markov Process

times, and n state transitions, for some natural number n. Two vectors are assigned to each path, 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

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

$$\mathbf{j} =< j_1, j_2, \ldots, j_J >$$

are assigned to each path, where for each rate reward r_i, $i = 1, 2, \ldots, K+1$, there are k_i entrances to states of rate r_i in the path. Similarly, for each impulse reward s_i, $i = 1, 2, \ldots, J$, there are j_i transitions with impulse reward s_i in the path. 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 $i = 1, 2, \ldots, J$, and

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

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

The PDF of the total reward $Y_{[0,t]}$ accumulated during some period of time $[0,t]$ can thus be expressed by first conditioning on the number of transitions that occur in the randomized process during $[0,t]$ and, further on $\mathbf{k}$ and $\mathbf{j}$, as follows:

$$P[Y_{[0,t]} \leq y] = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k}} \sum_{\forall \mathbf{j}}, \ [n, \mathbf{k}, \mathbf{j}] P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}], \tag{1}$$

where , $[n, \mathbf{k}, \mathbf{j}]$ is the probability of $\mathbf{k}$ intervals and $\mathbf{j}$ transitions, given there are n transitions in the uniformized process. In order to solve this equation, it is necessary to calculate $P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}]$ and , $[n, \mathbf{k}, \mathbf{j}]$. These calculations are considered in the subsequent sections.

B Calculation of the Conditional Distribution

Consider first 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 [5], but complicated by the fact that we condition on two reward vectors (rate and impulse) instead of one. To begin, we define l_i to be the sum of the lengths of intervals with rate reward r_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 $s_1, s_2, \ldots, s_J$ are the impulse rewards. Then $Y(t, n, \mathbf{k}, \mathbf{j})$, $Y_{[0,t]}$, given n, $\mathbf{k}$ and $\mathbf{j}$, can be expressed as

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

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

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

de Souza e Silva and Gail [5] evaluate a similar 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 $\mathbf{k}$ alone. They exploit the independence of 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 are of rate r_2, and so on. We can use the same approach, since $\sum_{i=1}^{J} s_i j_i$ is deterministic, given $\mathbf{j}$. In particular, define $U_{(k)}$ to be the k^{th} order statistic of a set of n independent and identically distributed uniform random variables T_a. The sum of the lengths of intervals of rate reward r_i, where $i = 1, 2, \ldots, K+1$, can then 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,

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

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

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

We can then use the result by Weisberg [21] to evaluate the conditional distribution of $Y_{[0,t]}$. The result of [21] is applicable if the variable parameter is shifted to be within the range described by Dempster and Klyle [3], 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 first condition, we shift the axis such that

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

Now, the shifted parameter is in the range

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

To simplify the notation, define

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

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

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

Furthermore, 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 [21] yields

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

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

$$g_i^{(k)}(c_i) = \sum_{j=0}^{k-1} \left(\begin{array}{c} k-1 \\ j \end{array} \right) g_i^{(j)}(c_i) h_i^{(k-1-j)}(c_i), \text{ where} \tag{2}$$

$$h_i^{(j)}(c_i) = (-1)^j j! \left[\frac{n}{(c_i - x)^{j+1}} - \sum_{l=1, l\neq i}^{K+1} \frac{k_l}{(c_i - c_l)^{j+1}} \right]. \tag{3}$$

Note that $k_i = 0$ if no state with rate reward r_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 $\mathbf{j}$, and hence must be excluded from the calculation. A simple renumbering is used to disregard such elements of $\mathbf{k}$. In particular, 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

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

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

$$P[Y_{[0,t]} \leq y] \;=\; \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k}} \sum_{\forall \mathbf{j}}, \; [n,\mathbf{k},\mathbf{j}] \left[1 - \sum_{i=1}^{m(\mathbf{k},\mathbf{j})} \frac{g_{l(\mathbf{k},i)}^{(k_{l(\mathbf{k},i)}-1)}(c_{l(\mathbf{k},i)})}{(k_{l(\mathbf{k},i)}-1)!} \right]. \qquad (4)$$

To solve (4), we still need to compute , $[n,\mathbf{k},\mathbf{j}]$. Computation of this quantity is discussed in the next subsection, in which we present a recursive procedure for calculating , $[n,\mathbf{k},\mathbf{j}]$, for increasing n, $\mathbf{k}$, and $\mathbf{j}$.

C Path Graph Generation

In order to compute , $[n,\mathbf{k},\mathbf{j}]$, we need to enumerate possible paths of the process that result in particular $\mathbf{k}$ and $\mathbf{j}$ and sum their probabilities. Two observations help us do this in an efficient manner. First, note that there can be many $\mathbf{j}$ for a particular $\mathbf{k}$, so the $\mathbf{j}$ can be grouped according to the $\mathbf{k}$ they are associated with. Second, paths of length $n+1$ can be determined from paths of length n using only their final state and probability. Hence, if a more refined notion of path is taken, which records the most recently visited state, as well as $\mathbf{k}$ and $\mathbf{j}$, path probabilities can be determined recursively. As will be seen in the next section, this notion of a path provides a mean to selectively discard paths whose contribution to the reward variable is small.

A generic path graph is illustrated in Figure 4. Each level $n \geq 0$ of the graph corresponds to the n^{th} transition of the uniformized process, and consists of a set of nodes η_n. 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 a state of the discrete time Markov chain, and p is the sum of probabilities of all paths from the initial state to state s, resulting in $\mathbf{k}$ and $\mathbf{j}$.

Based on the definition of a node, an algorithm to compute η_n given η_{n-1} can be formulated. Such an algorithm is given below, where $p(s,s')$ is the single-step transition probability from state s to s' in the discrete-time Markov chain.

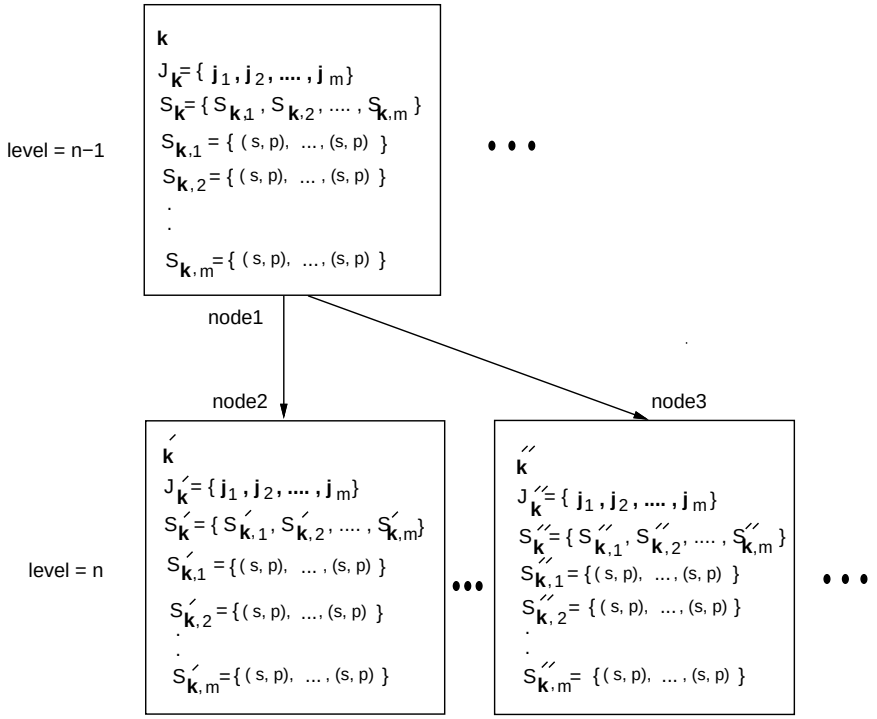


Figure 4: Generic Path Graph

Algorithm III.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
            $J_{\mathbf{k}'} = J_{\mathbf{k}'} \cup \{\mathbf{j}^{'}\}$.
            $S_{\mathbf{k}',\mathbf{j}'} = \{(s^{'}, p \times p(s, s^{'}))\}$.
        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 $\mathbf{k} \in \eta_{n-1}$.

With the help of the above algorithm, we can generate the complete path graph (up to some number of transitions N). The graph then gives us the probability, given n of particular $\mathbf{k}$'s, $\mathbf{j}$'s, and final states s. Thus, for solution purposes, it is appropriate to rewrite equation (4) in terms of this more refined summation. In particular,

$$P[Y_{[0,t]} \leq y] = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\mathbf{k} \in \eta_n} \sum_{\mathbf{j} \in J_{\mathbf{k}}} P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] \sum_{(s,p) \in S_{\mathbf{k},\mathbf{j}}} , [n, \mathbf{k}, \mathbf{j}, s], \qquad (5)$$

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 (the "p's", in the constructed graph), and $P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}]$ is as defined earlier.

To help understand the path graph generation, consider the uniformized Markov chain of Figure 3, corresponding to the SAN example introduced earlier. According to the definition of $\mathbf{k}$ and $\mathbf{j}$, 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, respectively, and $\mathbf{j} =< j_1, j_2 >$ where j_1, and j_2 correspond to transitions with impulse rewards 1, and 0, respectively.

Figure 5 gives an 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 have been 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 $\mathbf{k}$, $\mathbf{j}$ pair, 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 paths to the state s.

To illustrate the computation of entries in the nodes, we trace a path of the process of Figure 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 d with the probability of 0.5, resulting in

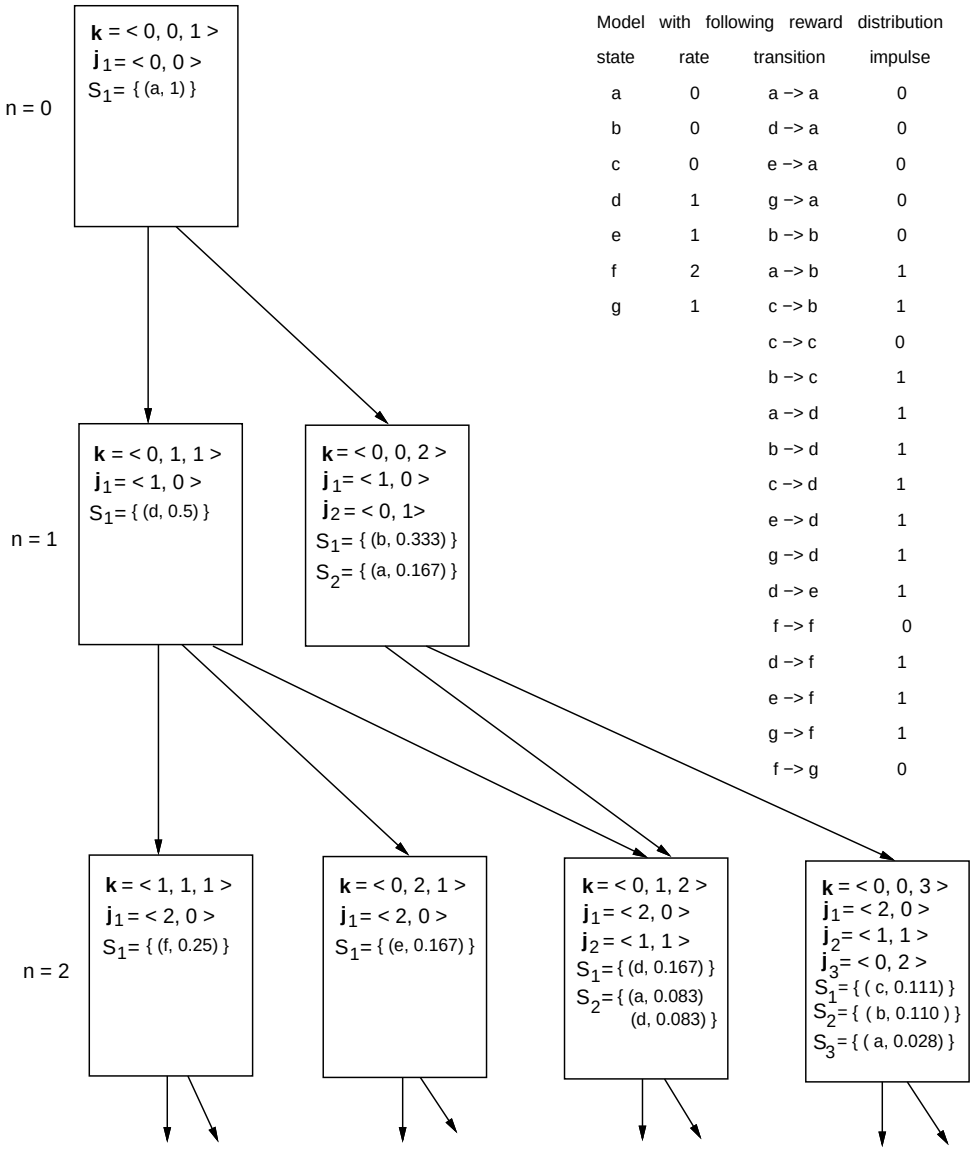


Figure 5: Path Graph for Example

$\mathbf{k}=<0,1,1>$, $\mathbf{j}=<1,0>$, and $(s,p) = (d, 0.5)$. Similarly, a transition from state d 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.

Thus we can compute the required conditional probability, via construction of a path graph. However, in order to obtain a value for equation (5), we need to determine the domain over which to actually perform the summations. The first summation is infinite, while the second and third summations can grow very fast. These summations quickly limit the size of the process that may be solved, if a method for limiting the growth of the tree is not employed. In the next section, we propose methods for discarding sets of paths whose contribution to the reward variable is negligible.

IV Error Bounds

Because the first summation in (5) is over an infinite set and the others can be over very large sets, the exact solution for $P[Y_{[0,t]} \leq y]$ cannot generally be computed. However, a solution with bounds can be computed, as will be shown in this section. The first summation must obviously be truncated, to make it finite. Bounds on the error induced by doing this are known [9] and, hence, are only briefly reviewed. Even with such "depth truncation", however, the space required to determine the conditional probabilities grows extremely rapidly with increasing n. In this section, we propose a new method for selectively discarding particular sets of paths in the path graph, based on their relative contribution to the performance variable.

A Error Due to Depth Truncation

Before describing path truncation, we briefly review calculation of the bound on the error due to truncating the first summation in (5) to a particular step N which we call *depth truncation*. The error, E_N, is

$$E_N = \sum_{n=N+1}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\mathbf{k} \in \eta_n} \sum_{\mathbf{j} \in J_{\mathbf{k}}} P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] \sum_{(s,p) \in S_{\mathbf{k},\mathbf{j}}} , [n, \mathbf{k}, \mathbf{j}, s]. \tag{6}$$

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

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

or, equivalently,

$$E_N \leq 1 - \sum_{n=0}^{N} \frac{e^{-\lambda t}(\lambda t)^n}{n!}. \tag{7}$$

Equation (7) can be used to calculate a truncation point N, to achieve a desired error bound.

B Error Due to Path Truncation

Even with depth truncation, the solution of equation (5) is immensely complex, since the size of the set $S_{\mathbf{k}}$ for a realization $\mathbf{k}$ increases exponentially with n. We propose a new method in which certain $(s,p) \in S_{\mathbf{k},\mathbf{j}} \in S_{\mathbf{k}}$ with small p are discarded, and compute a bound on the error induced by this discarding. As will be seen in the next section, the memory explosion associated with the sole use of depth truncation can largely be avoided, while still obtaining reasonably accurate estimates of the PDF of $Y_{[0,t]}$. We call this method *path truncation*, since it effectively truncates certain paths whose final state is s. Using this method, a particular (s,p) in a node of the path graph is ignored if p is less than a defined variable weight. As will be seen in the next section, the selection of the weight value determines the memory consumed and the error produced in the solution.

To determine a bound on the error induced by path truncation, let E_P be the error induced by discarding paths for which

$$, [n, \mathbf{k}, \mathbf{j}, s] \ \leq \ weight, \qquad 0 \leq weight \leq 1.$$

In order 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

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

Since the conditional distribution $0 \leq P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}] \leq 1 \ \forall y$, we can bound the error by assuming it equal to 1. This implies that

$$E_P(n, \mathbf{k}, \mathbf{j}, s) \ \leq \ \frac{e^{-\lambda t}(\lambda t)^n}{n!}, \ [n, \mathbf{k}, \mathbf{j}, s]. \tag{8}$$

Equation (8) is thus a bound on the error due to discarding a single (s, p). We now consider the error induced by discarding all (s, p)'s at the n^{th} transition in a set $G_{n,\mathbf{k},\mathbf{j}}$ of (s, p) associated with some $\mathbf{k}$ and $\mathbf{j}$ at the n^{th} transition. The bound for the error produced by such paths is then

$$\frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\mathbf{k} \in \eta_n} \sum_{\mathbf{j} \in J_{\mathbf{k}}} \sum_{(s,p) \in G_{n,\mathbf{k},\mathbf{j}}} , \ [n, \mathbf{k}, \mathbf{j}, s]. \tag{9}$$

This gives us a bound on the error due to discarding all $(s, p) \in G_{n,\mathbf{k},\mathbf{j}}$ at a given transition. Since the probability calculation is recursive, the total error due to discarding paths for a given n is determined by the (s, p) discarded at the n^{th} transition as well as by the (s, p) discarded at preceding transitions. The contribution to the error, for a given n, made by discarding (s, p) with a weight less than *weight* at the $(n-1)^{th}$ transition can be understood by considering the fact that the entries in a row of a transition matrix sum to 1. This implies that the sum of the probabilities of the missing paths at the n^{th} transition due to a discarded (s, p) at the $(n-1)^{th}$ transition is p. Let $, \ _D[n]$ be the sum of all paths missing at the n^{th} transition. Then, as per the above argument,

$$, \ _D[n] = \sum_{\mathbf{k} \in \eta_n} \sum_{\mathbf{j} \in J_{\mathbf{k}}} \sum_{(s,p) \in G_{n,\mathbf{k},\mathbf{j}}} , \ [n, \mathbf{k}, \mathbf{j}, s] + , \ _D[n-1], \qquad n \geq 1, \tag{10}$$

where $, \ _D[0] = 0$.

Therefore, using (9) and (10), a bound on the total error induced by discarding paths for a given transition n is

$$E_P(n) \leq \frac{e^{-\lambda t}(\lambda t)^n}{n!}, \ _D[n].$$

The error bound over all n, for some truncation depth N, is then

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

Since paths are truncated based on their probabilities, the error induced by choice of a particular *weight* can not be determined prior to generation of the paths for the model. In our implementation (as discussed in Section VI) we compute this error "on-the-fly," as the path space is explored. After E_P is computed, a lower bound on the PDF of $Y_{[0,t]}$ can be computed as

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

where $S'_{\mathbf{k},\mathbf{j}} = S_{\mathbf{k},\mathbf{j}} - G_{n,\mathbf{k},\mathbf{j}}$ is the set of (s,p) not discarded for particular $\mathbf{k}$ and $\mathbf{j}$. An upper bound on $P[Y_{[0,t]} \leq y]$ can be computed by adding E_P and E_N to the lower bound.

V Computational Complexity

This section investigates the time and space complexity of the computation of the distribution of $Y_{[0,t]}$. The exact computation of these complexities depends on the nature of the stochastic process and reward structure of the model being considered, and hence is not possible to compute, in general. Furthermore, the path truncation technique introduced in the previous section reduces the space and number of computations necessary in a very problem-specific way, and it is difficult to incorporate this effect in a generic complexity analysis. We thus will first, in this section, discuss the computational complexity of determining $P[Y_{[0,t]} \leq y]$ without assuming path truncation to give the reader an idea about the maximum space and time complexity of the algorithm, relative to the algorithm in [5]. Then, in the next section, we will illustrate the space and time complexity of the algorithm (as implemented) for a particular problem, showing the effectiveness of path truncation in reducing both space and computation time.

We first consider the space complexity of the algorithm. The algorithm is based on recursion by row (see subsection III.C), where the set η_n is needed to compute the set η_{n+1}. Each node (element) of the set η_n corresponds to a distinct vector k with number of transitions n. There can be, at most, $\begin{pmatrix} K+1+n \\ n+1 \end{pmatrix}$ such vectors (as with the algorithm in [5]), where n is the number of transitions and $K+1$ is number of distinct rate rewards. Each node keeps, in addition to the vector $\mathbf{k}$ itself, information about the elements $S_{\mathbf{k},\mathbf{j}}$ of

set $S_{\mathbf{k}}$. Recall that each $S_{\mathbf{k},\mathbf{j}}$ is a set of pairs (s, p), where s is a state of the discrete time Markov chain, and p is the sum of probabilities of all paths from the initial state to state s.

As the recursion is in row order, the set η_n can be discarded after η_{n+1} is computed. For a given truncation depth N, the process would thus grow to its maximum size when it has set η_N stored in the memory. Considering the worst case, we need to compute the maximum number of vectors j ($S_{\mathbf{k},\mathbf{j}}$) and pairs (s, p) distributed over all the nodes of η_N. To do this, let a and b be the number of possible initial states and total number of states of the uniformized process. Furthermore, suppose (as in the worst case) that the uniformized process is fully connected and each transition has a distinct impulse reward associated with it. Then the maximum number of vectors j and pairs (s, p) distributed over all the nodes of η_N is $a \times (b)^n$. ($a \times (b)^n$ correspond to the maximum number of paths that can be generated for N transitions). In the worst case, there are as many $S_{\mathbf{k},\mathbf{j}}$ sets as number of paths and each set has only one element (s, p).

Thus more space is needed (in the worst case, if path truncation is not used), compared to the algorithm in [5], since [5] does not consider rate and impulse rewards together, and hence can make use of a single vector $\mathbf{k}$, rather than both $\mathbf{k}$ and $\mathbf{j}$. In this case, each set $S_{\mathbf{k}}$ will have one element. The maximum number of pairs (s, p) for a row remains the same but now they are collected in fewer sets. The increased number of elements of $S_{\mathbf{k},\mathbf{j}}$ is thus the major contribution to the larger process size for our algorithm, relative to [5], if path truncation is not used. Note that, in practice, the (s, p) pairs within a set $S_{\mathbf{k},\mathbf{j}}$ with same state s can be combined into a single (s, p) pair, where state s is the common state and probability p is the sum of probabilities of the combined (s, p) pairs.

Regarding the time complexity, we must compute one conditional distribution for each element $S_{\mathbf{k},\mathbf{j}} \in S_{\mathbf{k}}$, $\forall \mathbf{k}$. The number of these elements is as described previously in this section. In contrast, the algorithm in [5] must to compute one conditional distribution for each element of the set η_n since set $S_{\mathbf{k}}$ contains only a single element.

If no impulse rewards are defined on the uniformized process (which corresponds to 0 impulse reward for each state) our algorithm (without path truncation) reduces in time and space complexity to the algorithm described in [5]. Furthermore, as will be seen in the next section, in practice, the algorithm as implemented (with path truncation), can take significantly less space and time than suggested here, while still obtaining reasonable accuracy.

VI Implementation and Results

The method has been implemented as a part of the modeling package *UltraSAN* [2], which is based on the SAN modeling framework. In *UltraSAN*, a model of a system consists of a hierarchical (or composed) SAN-based reward model, of the type described in Section II. The package automatically generates a reduced base model (Markov renewal process) and a state-level reward structure from the SAN-level description. This state-level description serves as input to a solver which implements the method developed in this paper. The solver is comprised of some 3700 lines of C language code. It provides options to specify all parameters which affect a solution, including: the interval of time $[0, t]$ desired (t), the accuracy for depth truncation (a), the weight used to discard paths (w), the range $[0, r]$ of the reward variable over which to calculate the distribution (r), and the number of points within the range for which the distribution should be computed (n). The specified accuracy is used to calculate a truncation point N using (7) such that $E_N \leq 10^{-a}$, i.e. there are a decimal digits of accuracy. w is also specified as a power of 10; the interpretation being that (s, p)'s with $p \leq 10^{-w}$ are discarded. The other parameters should be self explanatory, but more detail can be found in [14].

It is worth noting that a careful implementation is necessary to avoid numerical problems when implementing the algorithm. In particular, there are several places where care must be taken to avoid numerical overflow and underflow. In some cases, we have avoided underflow and overflow by changing the order of calculation within an expression, in others, we have used a multi-precision math package [19]. Specifically, the conditional PDF of $P[Y_{[0,t]} \leq y \mid n, \mathbf{k}, \mathbf{j}]$ is difficult to calculate using normal double precision arithmetic for some input parameter values. To avoid this problem, we use multi-precision arithmetic with a variable (user specified) accuracy for this calculation.

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 [15]. All the results presented are for a IBM RS6000 Model 530H with 64 Megabytes of RAM, and all multi-precision arithmetic was done at an accuracy of 100 decimal digits.

Figure 6 depicts the SAN model of the multiprocessor system. Here, the places *processor* and *buffer* represent the number of fault-free processors and the number of buffer stages, respectively, while the place *repair* indicates the number of processors queued for repair. The activities *processor_failure* and *buffer_failure* represent the occurrence of faults in the

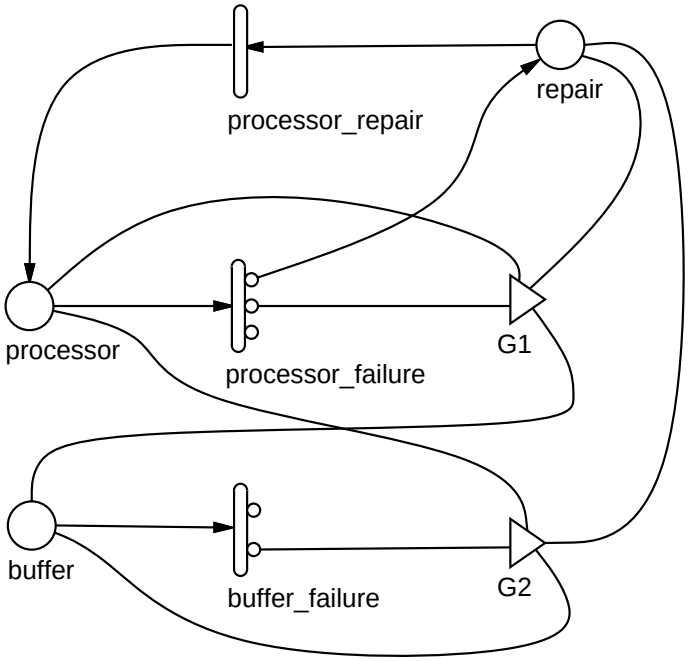


Figure 6: Multiprocessor Model

Table 2: Activity Time Distributions for Multiprocessor Model

Activity	**Distribution**	**Case Probabilities**		
		1	2	3
$buffer\_failure$	expon(0.001 * MARK(buffer))	0.99	0.01	0
$processor\_failure$	expon(0.002 * MARK(processor))	0.80	0.19	0.01
$processor\_repair$	expon(0.1)	1	0	0

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-repairable 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 and 3 present the associated details about the marking dependent rates of the activities, case probabilities, and gate functions. The unit of time for the example is days. It should be noted that the system considered is typically difficult to solve by uniformization, because the minimum

Table 3: Output Gate Functions for Multiprocessor Model

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

Table 4: Throughput as Determined from Performance Submodel

# 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

holding time is very small relative to the interval of time considered. Hence, a large number of steps (large N) need to be generated when computing the result to obtain a reasonable error bound.

Two performance variables are considered: the total "benefit" obtained from operating the multiprocessor for some interval of time, and the availability of the multiprocessor. The first variable is a measure of the performability of the system, and depends both on costs associated with processor repair (an impulse reward) and benefit gained from operating the multiprocessor. We employ behavioral decomposition in the solution, using as reward rates throughputs determined from a performance submodel. In particular, table 4, from [15], shows the throughput for different configurations of the multiprocessor 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 benefit associated

with operating the system, for some utilization period $[0, t]$, can be found by using the reward structure

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

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

where $\mathcal{C}$ and $\mathcal{R}$ are as defined in Section II.

Before formulating the measure of availability, 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

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

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

The remainder of this section presents results for the availability and performability of the multiprocessor system to illustrate the effect various parameter values have on the time and space complexity of the developed method. The first four sets of results were computed at a high level of accuracy, using $a = 6$ and $w = 8$.

The results of the first set of experiments are presented in Figure 7, and show the space and time required to calculate the distribution of total benefit (performability) obtained from operating the system for 10 days ($t = 10$), when the number of buffers in the system is varied. An increase in the number of processors and buffers corresponds to an increase in the number of different performances the system can exhibit. This implies an increase in number of different rate rewards in the reward model, which increases the number of elements in the vector $\mathbf{k}$. Therefore, at each level in the path graph, there are more nodes to compute, which increases the computation time and memory required to store a level and to compute the next level in the graph. The error bound on the resulting PDF's varied as the number of distinct rate rewards was varied, but was always within the range given on the caption of Figure 7.

The second set of experiments investigates the computation of the distribution of availability of the modeled system. In the computation of the distribution of availability, an

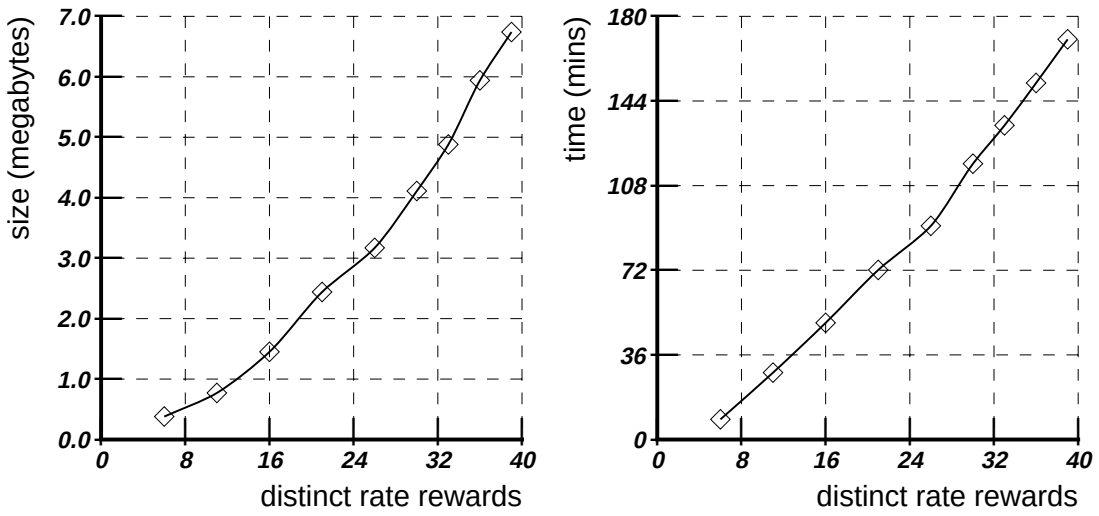


Figure 7: Effect of Increasing the Number of Distinct Rate Rewards ($t = 10$, $a = 6$, $w = 8$, $r = 50$, $n = 50$); $(2.3e-07 \leq (E_N + E_P) \leq 4.5e-07)$

increase in the number of combinations of processors and buffers only increases the number of states having a particular rate reward, not the number of distinct rate rewards. The number of distinct rate rewards thus remains fixed at two, corresponding to the availability and non-availability of the system. In terms of the path graph, the number of nodes at each level remains the same while the number $(s, p) \in S_{\mathbf{k},\mathbf{j}} \in S_{\mathbf{k}}$ increases greatly. Figure 8 shows that increasing the number of states by increasing number of buffers, while keeping the number of distinct rate rewards fixed, significantly increases the memory required for solution, but that the time required for solution remains very small. As with the previous sets of experiments the error bound achieved is given in the caption of Figure 8.

The third set of experiments 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 of the solution. The experiment computes the distribution of performability when the interval of time is varied from 5 days to 50 days. Low minimum holding times and large observation intervals require generation of more transition steps of the uniformized process for a specified error bound. An increase in the number of required transitions increases the time required to calculate a solution. In addition, as the number of transition steps required increases, the number of nodes explored in the path graph

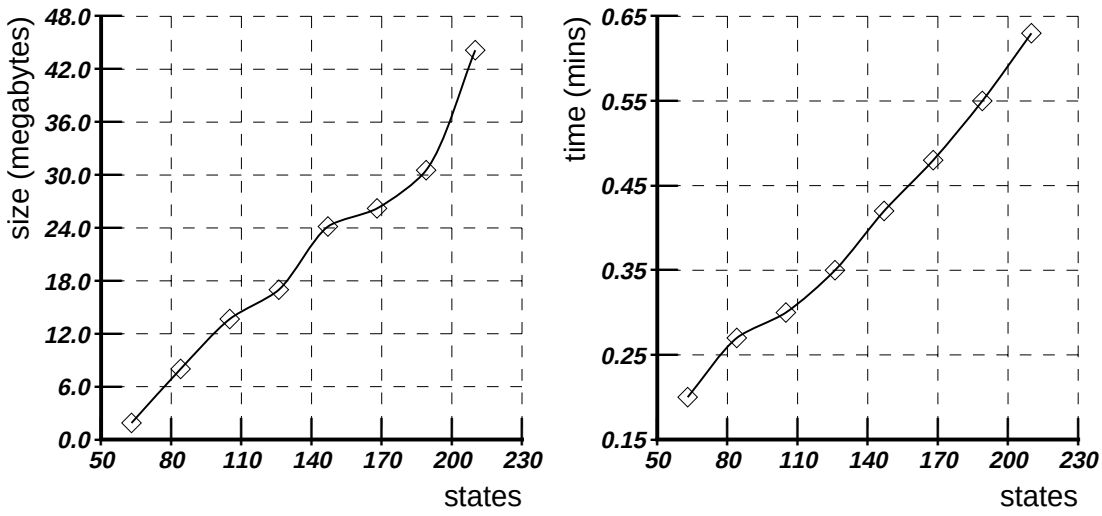


Figure 8: Evaluation of the Distribution of Availability ($t = 10$, $a = 6$, $w = 8$, $r = 20$, $n = 20$); $(2.9e-07 \leq (E_N + E_P) \leq 4.8e-07)$

will increase and result in an increase in process size. Figure 9 illustrates the relation of an increase in the observation time interval to the required time for computation and the process size. This shows the sensitivity of the calculation to the minimum holding time in the process, or the interval of time considered. In should be recalled that the minimum holding time is small in the multiprocessor model: this growth would not be near as fast for a system which has a greater minimum holding time relative to the interval of time considered.

Another parameter that affects the time required to compute a solution is the number of points on the distribution curve that are computed. A single path graph and set of Poisson probabilities can be used for all time points, necessitating only the additional computation of the conditional probability distribution for each new time point. Hence the time overhead of additional points is linear and space overhead is barely measurable. Figure 10 illustrates this result, for the performability variable, for an interval of time $[0, 10]$ when an accuracy of 6 ($a = 6$) and a weight of 8 ($w = 8$) for discarding paths was used.

All the above results are evaluated to a very high accuracy. In particular, in all cases, an accuracy of 6 ($a = 6$), and a weight of 8 ($w = 8$) for discarding paths was used. The required accuracy for the probability distribution depends upon the underlying nature of

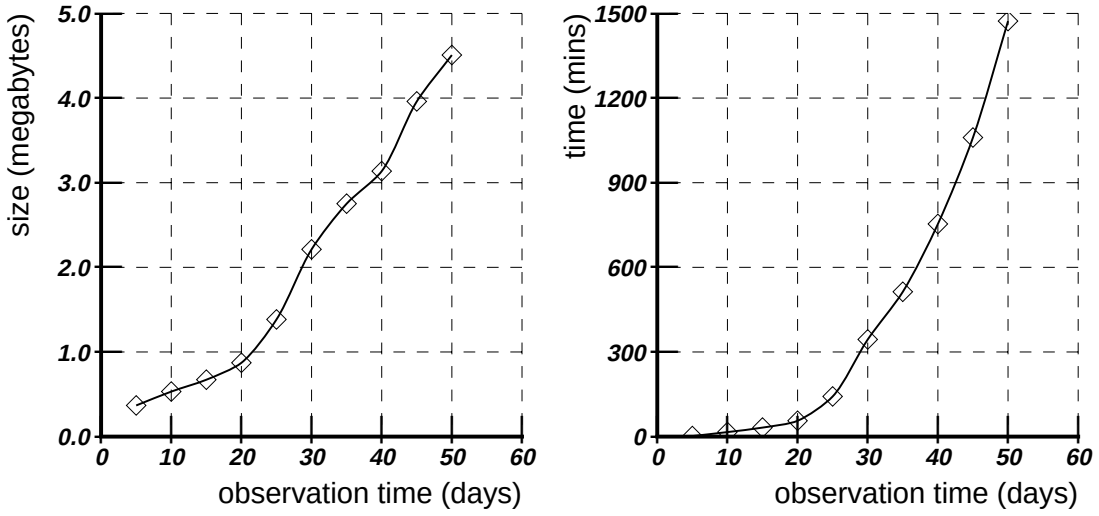


Figure 9: Effect of Increasing the Observation Interval ($a = 6$, $w = 8$, $r = 20$, $n = 20$); $(8.4e-08 \leq (E_N + E_P) \leq 5.4e-06)$

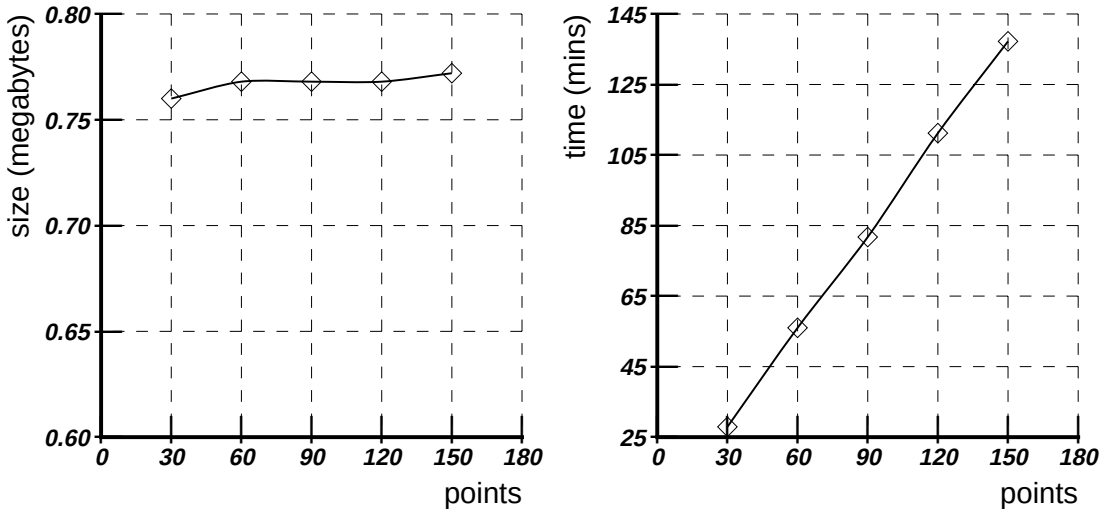


Figure 10: Effect of Increasing the Number of Points Calculated ($t = 10$, $a = 6$, $w = 8$); $((E_N + E_P) = 2.071955e-07)$

the model and the variable that is evaluated. In many cases, this high degree of accuracy is not needed, and a dramatic reduction in the space and time required for solution can be realized if a larger weight for discarding paths is used. Figure 11 illustrates this, by considering computation of the distribution of availability for interval of time $[0, 10]$ when $r = 20$, $n = 20$, and $a = 6$. Figure 11 gives the the error bound $(E_N + E_P)$, process size, and CPU time required as a function of the discarding weight used. The results show that a reduction in weight increases the number of discarded paths, drastically reducing the size and the CPU time required.

To illustrate the size of processes that can be handled if a large discarding weight is used, we consider a version of the multiprocessor model and the calculation of the distribution of availability as the number of processors is varied between 15 and 30 while the number of buffers is held constant at 40. The accuracy and weight were chosen to obtain two decimal digits of accuracy ($a = 2$, $w = 4$). As can be seen from Figure 12, very large problems can be solved if a large number of significant digits of accuracy is not required.

VII Conclusions

In summary, we have presented an algorithm and implementation for the calculation of the probability distribution function of reward accumulated over some interval of time. The method is based on uniformization of a continuous-time Markov process generated from a stochastic activity network. This allows both impulse and rate rewards to be specified at the network, rather than at the uniformized process level. We have also proposed a method for discarding specific sets of paths in the uniformized process, based on their contribution to the reward variable, and computed a bound on the error introduced by discarding such paths. Finally, we presented experimental results that show that the discarding policy can dramatically reduce the time and space required to obtain a solution, while achieving acceptable accuracy.

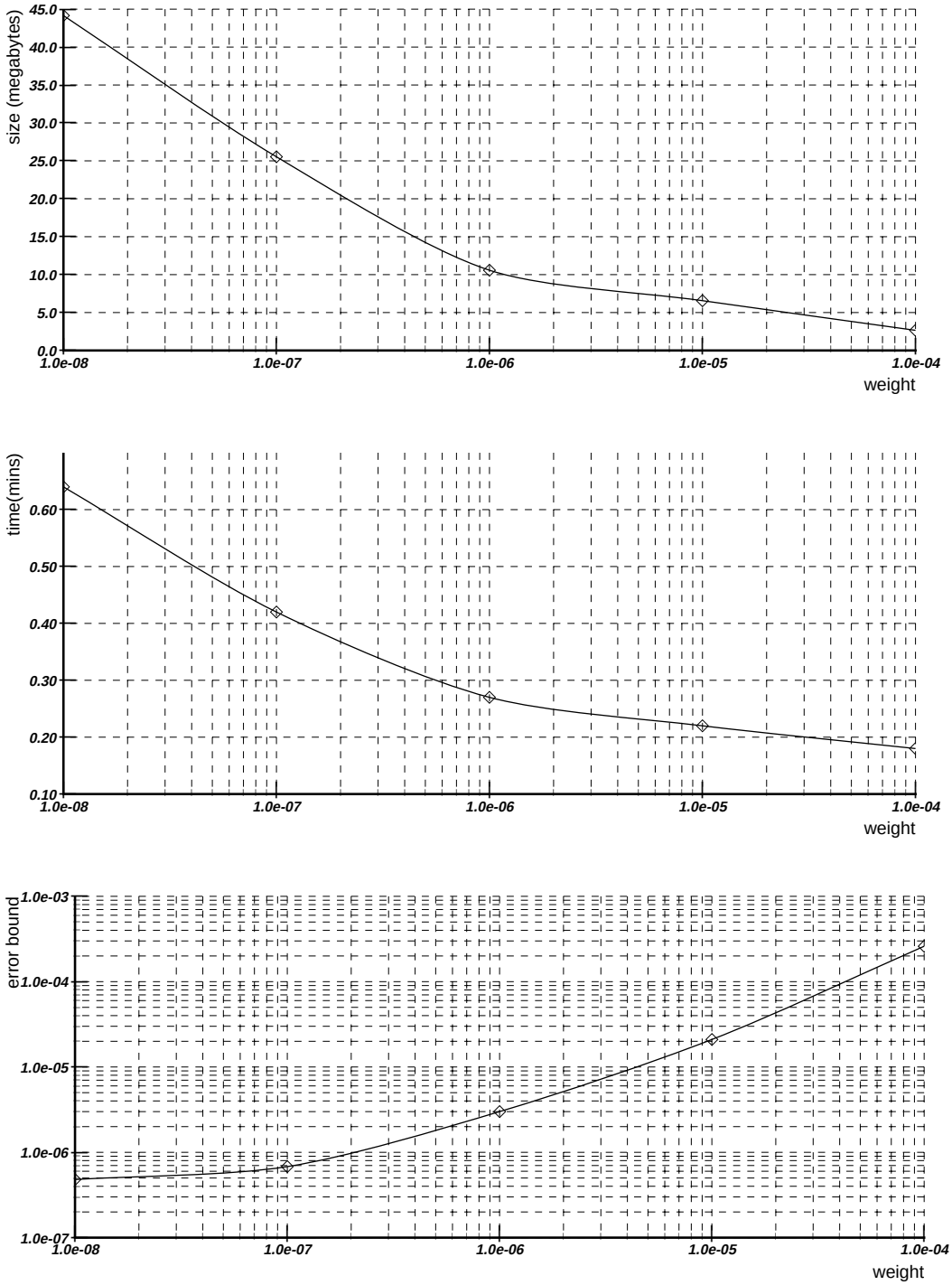


Figure 11: Effect of Discarding of Paths ($t = 10$, $a = 6$, $r = 20$, $n = 20$)

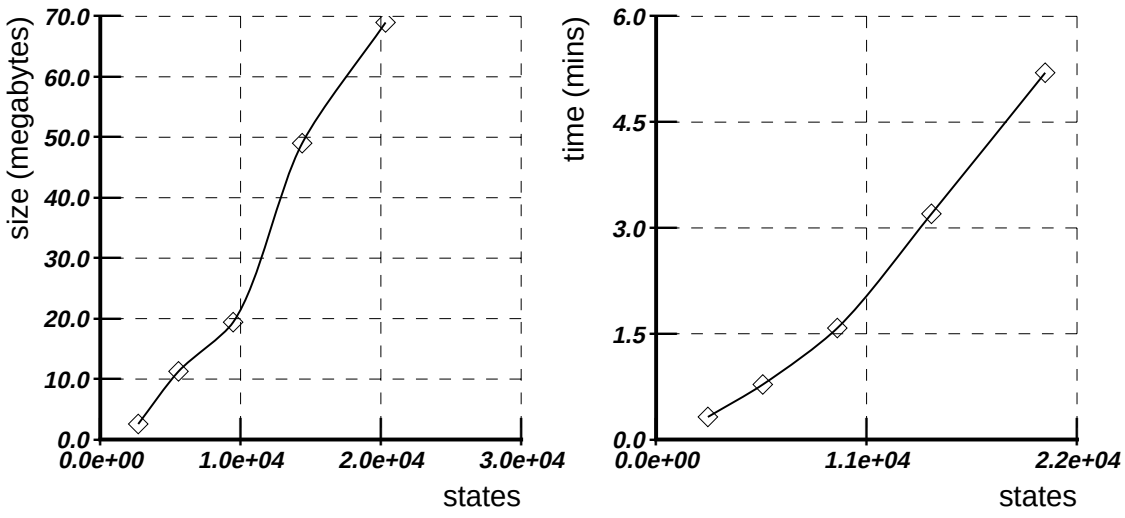


Figure 12: Results for Large State Spaces ($t = 10$, $a = 2$, $w = 4$, $r = 20$, $n = 20$); $(4.5e-03 \leq (E_N + E_P) \leq 1.1e-02)$

REFERENCES

[1] B. Ciciani, and V. Grassi, *"Performability evaluation of fault-tolerant satellite systems,"* IEEE Trans. Commun. 35 (4) (Apr. 1987) 403-409.

[2] 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 8 (5) (Sept. 1991) 69-80.

[3] A. P. Dempster, and R. M. Kleyle, *"Distributions determined by cutting a simplex with hyperplanes,"* Ann. Math. Stat. 39 (5) (1968) 1473-1478.

[4] L. Donatiello, and B. R. Iyer, *"Analysis of a composite performance reliability measure for fault-tolerant systems,"* JACM. 34 (1) (Jan. 1987) 179-199.

[5] E. de Souza e Silva, and H. R. Gail, *"Calculating availability and performability measures of repairable computer systems using randomization,"* JACM. 36 (1) (Jan. 1989) 171-193.

[6] E. de Souza e Silva, and H. R. Gail, *"Calculating cumulative operational time distributions of repairable computer systems,"* IEEE Trans. Comput. 35 (4) (Apr. 1986) 322-332.

[7] D. G. Furchgott, and J. F. Meyer, *"A performability solution method for degradable non-repairable systems,"* IEEE Trans. Comput. 33 (6) (Jun. 1984) 550-554.

[8] A. Goyal, and A. N. Tantawi, *"Evaluation of performability for degradable computer systems,"* IEEE Trans. Comput. 36 (6) (Jun. 1987) 738-744.

[9] D. Gross, and D. R. Miller, *The randomization technique as a modelling tool and solution procedure for transient Markov processes,"* Op. Res. 32 (2) (Mar-Apr. 1984) 343-361.

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

[11] 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 (Vienna, Austria, Jul. 1985) 252-257.

[12] J. F. Meyer, *"On evaluating the performability of degradable computing systems,"* IEEE Trans. Comput 22 (Aug. 1980) 720-731.

[13] J. F. Meyer, *"Closed-form solutions of performability,"* IEEE Trans. Comput. 31 (7) (Jul. 1982) 648-657.

[14] M. A. Qureshi, *Reward model solution methods with impulse and rate rewards: an algorithm and numerical results,"* M.S. thesis, University of Arizona (1992).

[15] W. H. Sanders, and J. F. Meyer, *"Performability evaluation of distributed systems using stochastic activity networks,"* Petri nets and performance models (Madison, WI, Aug. 1987).

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

[17] W. H. Sanders, and J. F. Meyer, *"Reduced based model construction for Stochastic Activity Networks,"* IEEE Journal in selected areas in Communications 9 (1) (Jan. 1991) 273-283.

[18] 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).

[19] D. M. Smith, *"A FORTRAN package for floating-point multiple-precision arithmetic,"* ACM transactions on mathematical software 17 (2) (Jun. 1991) 273-283.

[20] R. M. Smith, K. S. Trivedi, and A. V. Ramesh, *"Performability analysis: measures, an algorithm, and a case study,"* IEEE Trans. Comput. 37 (4) (Apr. 1987) 406-417.

[21] H. Weisberg, *"The distribution of linear combinations of order statistics from the uniform distribution,"* Ann. Math. Stat. 42 (2) (1971) 704-709.