A Structured Path-Based Approach for Computing Transient Rewards of Large CTMCs

Vinh V. Lam* Peter Buchholz† William H. Sanders*

*Dept. of Electrical and Computer Engineering
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL, U.S.A.
Email: {lam, whs}@crhc.uiuc.edu

†Informatik IV, Universität Dortmund
D-44221 Dortmund, Germany
Email: peter.buchholz@udo.edu

Abstract

Structured (a.k.a. symbolic) representation techniques of Markov models have, to a large extent, been used effectively for representing very large transition matrices and their associated state spaces. However, their success means that the largest space requirement encountered when analyzing these models is often the representation of their iteration and solution vectors. In this paper, we present a new approach for computing bounds on solutions of transient measures in large continuous-time Markov chains (CTMCs). The approach extends existing path- and uniformization-based methods by identifying sets of paths that are equivalent with respect to a reward measure and related to one another via a simple structural relationship. This relationship allows us to explore multiple paths at the same time, thus significantly increasing the number of paths that can be explored in a given amount of time. Furthermore, the use of a structured representation for the state space and the direct computation of the desired reward measure (without ever storing the solution vector) allow us to analyze very large models using a very small amount of storage. In addition to presenting the method itself, we illustrate its use to compute the reliability and the availability of a large distributed information service system in which faults may propagate across subsystems.

1 Introduction

In performance and dependability analysis, transient measures are often computed by discrete event simulation or by numerical analysis of the CTMCs underlying the models. Both approaches have their drawbacks: simulation can only estimate results up to a certain accuracy because it relies on the observation of a stochastic process, while numerical analysis suffers from the problems associated with state-space explosion. Moreover, when the intervals to be simulated are long and there is large variance in the transient measures being computed, many trajectories need to be analyzed, but no bounds can be placed on the accuracy of the obtained results. Even though statistical confidence intervals can be calculated in simulation, without bounds one cannot absolutely determine how close the obtained results are to the correct results. Numerical analysis, on the other hand, offers the potential to compute bounds when the exponential growth in state-space size, the associated transition matrix, and the solution vector can be managed efficiently.

Recently, a number of researchers have devised data structures and algorithms to manage the state space efficiently. One of the approaches represents the generator matrix of a CTMC in compact form as a *Kronecker* product of smaller component matrices [10, 12]. An interesting approach that successfully exploits data structures is based on *decision diagrams* and *matrix diagrams*, e.g., [4]. These techniques make it possible to analyze CTMCs that have state spaces that are several orders of magnitude larger than could previously be handled. In effect, they shift the bottleneck encountered during solution from the storage of the state space and transition matrix to the storage of the iteration and solution vectors (whose size is on the order of that of the state space).

In this paper, we present a new path-based approach for computing the solutions of transient measures in large CTMCs. The approach extends existing path- and uniformization-based methods (e.g., [11, 13, 9]) by identifying sets of paths that are equivalent with respect to a reward measure and related to one another via a simple structural relationship. This relationship allows us to explore multiple paths at the same time, and thus significantly increases the number of paths that can be explored. Furthermore, the use of a structured representation for the state space and the direct computation of the desired reward measure (without ever storing the solution vector) allow us to analyze very large models using a modest amount of storage. In addition, due to the combined aspects of path and uniformization analysis, our approach can compute both lower and upper bounds on the solution.

This paper proceeds as follows. In Section 2, we briefly review the transient analysis of reward measures for CTMCs via the uniformization method. Next, in Section 3, we describe the class of Markovian models that we intend to analyze and their underlying matrix structures. Then, in Section 4, we introduce our path-based approach utilizing the methods described in the previous two sections. In Section 5, we discuss issues related to the implementation of our approach. In Section 6, we present some preliminary results, and in Section 7, we discuss the algorithmic costs of the approach. Finally, in Section 8, we conclude our presentation with a discussion of future work.

2 Transient Analysis of CTMCs Via Uniformization

Uniformization [12] is an often-used method for transforming a CTMC into a DTMC for transient analysis. Briefly, given a CTMC with finite state space $\mathcal{RS}$, generator matrix $\mathbf{Q}$, initial distribution vector $\mathbf{p}_0$, and reward vector $\mathbf{r}$, the uniformization procedure may be applied when the diagonal elements of $\mathbf{Q}$ are finite. Applying uniformization, we can compute $\mathbf{p}_s$, the transient distribution at time s, as

$$\mathbf{p}_s = \mathbf{p}_0 e^{(\mathbf{Q}s)} = \mathbf{p}_0 \sum_{k=0}^{\infty} \mathbf{P}^k \beta(\alpha s, k) \tag{1}$$

where $\alpha = \max_x(|Q(x,x)|)$, $\mathbf{P} = \mathbf{Q}/\alpha + \mathbf{I}$, and $\beta(\alpha s, k) = e^{-\alpha s}(\alpha s)^k/k!$ is the probability that a Poisson process with rate αs will make k jumps in the interval (0,1]. For a finite truncation of the infinite sum, lower and upper bounds for the probability $\mathbf{p}_s(x)$, $x \in \mathcal{RS}$, can be computed easily using standard means.

The instantaneous reward at time s is computed as

$$E[R_s] = \mathbf{p}_0 \left(\sum_{k=0}^{\infty} \mathbf{P}^k \beta(\alpha s, k) \right) \mathbf{r}. \tag{2}$$

Finite truncation of the sum can be bounded above and below by

$$\mathbf{p}_0 \left(\sum_{k=0}^{K} \mathbf{P}^k \beta(\alpha s, k) \right) \mathbf{r} \;\le\; E[R_s] \le$$
$$\mathbf{p}_0 \left(\sum_{k=0}^{K} \mathbf{P}^k \beta(\alpha s, k) \right) \mathbf{r} + \left(1 - \sum_{k=0}^{K} \beta(\alpha s, k) \right) \max_x(\mathbf{r}(x)). \tag{3}$$

The expected value of the accumulated reward during the interval [0,s) is computed as

$$E[AR_s] = \int_0^s \mathbf{p}_s \mathbf{r} = \mathbf{p}_0 \left(\sum_{k=0}^{\infty} \mathbf{P}^k \frac{1 - \beta(\alpha s, k)}{\alpha} \right) \mathbf{r}, \tag{4}$$

and the bounds on the finite truncation of its sum can be computed in the same manner as in (3).

3 Analyzable Class of Markovian Models and Their Underlying Matrix Structure

In this section, we describe a class of structured Markovian models that are analyzable by our approach. Many large, non-trivial models in reliability and performance analysis are built up from smaller logical components or submodels. The submodels are composed together by means of *state sharing* (e.g. [6]) or *action synchronization* (also known as *action sharing*). In our approach, we consider action synchronization. The structure in these models can be exploited to make their analysis more efficient. In particular, we consider the class of models for which we can represent generator matrices and solution vectors in a structured manner by means of Kronecker products and sums. In the following paragraphs, we describe the structure of the matrices underlying this class of models.

Suppose a given model has state space $\mathcal{RS}$ (where $|\mathcal{RS}| = n$) and is decomposable into J components, which are numbered from 1 to J. We usually use parenthesized superscripts and subscripts, i,$j \in \{1, \ldots, J\}$, to denote particular components, unless the context is clear (in which case, we do not use parentheses). $\mathcal{RS}_i = \{0, \ldots, n_i - 1\}$ is the set of states of component i, and the relation $\mathcal{RS} \subseteq \times_{i=1}^{J} \mathcal{RS}_i$ holds. Because the global state space is described by the states of the individual components, each global state can be represented by a J-dimensional vector $\mathbf{x}$ such that $\mathbf{x}(i)$ is the state of component i in global state $\mathbf{x}$. This J-dimensional representation can be linearized via $x = \sum_{i=1}^{J} \mathbf{x}(i) \cdot n_{i+1}^{J}$, where $n_j^i = \prod_{k=j}^{i} n_k$ and empty product equals 1. Both linearized and J-dimensional representations can be used interchangeably. This basic compositional representation of $\mathcal{RS}$ is the structure used in [10] and many other papers to represent $\mathbf{Q}$ in compact form. For numerical analysis, it is generally not sufficient to consider the cross product of component state spaces because of the unreachable states. However, for the bounding approach presented here, the representation is appropriate, because the path-based computation of rewards implicitly considers only reachable states. We will expand upon this feature further in Section 4, when we describe our path-based approach.

Components in a model interact via events or transitions from a set of synchronized transitions, $\mathcal{T}_S$. Furthermore, a component has local behavior described by local transitions. Let $\mathbf{Q}_l^{(i)}$ be the matrix of local transition rates of component i. $\mathbf{Q}_l^{(i)}$ is the generator matrix of a CTMC with state space $\mathcal{RS}_i$. For each component i and synchronized transition $t \in \mathcal{T}_S$, let $\mathbf{E}_t^{(i)}$ be a matrix describing the transitions due to the occurrence of t in the state space of $\mathcal{RS}_i$. Thus, $\mathbf{E}_t^{(i)}(x,y)$ is the weight of the firing of t in local state x to transition to local state y in component i. If component i is not involved in the firing of t, then $\mathbf{E}_t^{(i)} = \mathbf{I}_{n_i}$, the identity matrix of order n_i. The global firing rate of transition $t \in \mathcal{T}_S$ starting in state $\mathbf{x}$ and ending in state $\mathbf{y}$ is given by $\lambda_t \prod_{i=1}^{J} \mathbf{E}_t^{(i)}(\mathbf{x}(i), \mathbf{y}(i))$, where λ_t is the rate of the transition. We assume that the weight of any transition $t \in \mathcal{T}_S$ is scaled such that $\max_{x \in \mathcal{RS}_i}(\sum_{y \in \mathcal{RS}_i} \mathbf{E}_t^{(i)}(x,y)) \le 1.0$. In other words, matrices $\mathbf{E}_t^{(i)}$ have row sums between 0 and 1. This is to account for marking-dependent transition rates [2]; for constant transition rates, each row sum is either 0 or 1. The descriptor $\hat{\mathbf{Q}}$ of the CTMC, which contains the generator matrix $\mathbf{Q}$ as a submatrix, can be represented as the composition of component matrices:

$$\hat{\mathbf{Q}} = \bigoplus_{j=1}^{J} \mathbf{Q}_l^{(i)} + \sum_{t \in \mathcal{T}_S} \lambda_t \left(\bigotimes_{j=1}^{J} \mathbf{E}_t^{(i)} - \bigotimes_{j=1}^{J} \mathbf{D}_t^{(i)} \right), \tag{5}$$

where $\mathbf{D}_t^{(i)} = \text{diag}(\mathbf{E}_t^{(i)} \mathbf{e}^T)$ and $\mathbf{e}$ is a n_i-dimensional vector of ones.

The representation of $\hat{\mathbf{Q}}$ in (5) is very compact, since all of the component matrices used in that equation have dimensions $n_i \times n_i$ rather than $n \times n$ (where n can be much larger than n_i). We assume that the initial distribution vector can also be composed from initial vectors of the components. Let $\mathbf{p}_0^{(i)}$ be the initial vector of component i ($\mathbf{p}_0^{(i)}$ can be obtained according to the decomposition of $\mathcal{RS}$ into $\mathcal{RS}_i$). The initial distribution of the complete model is a row vector that is given by $\mathbf{p}_0 = \bigotimes_{i=1}^{J} \mathbf{p}_0^{(i)}$ Similarly, let column vector $\mathbf{r}^{(i)}$ be the reward vector for component i; then $\mathbf{r} = \bigoplus_{i=1}^{J} \mathbf{r}^{(i)}$ or $\mathbf{r} = \bigotimes_{i=1}^{J} \mathbf{r}^{(i)}$ is the reward vector of the complete model. Whether the reward vector is built by the Kronecker sum or product depends on the type of reward. We can extend this beyond sums or products and use other associative and commutative operations like maximum or minimum.

Uniformization can be applied at the level of the components to yield a compact representation of matrix $\mathbf{P} = \mathbf{Q}/\Lambda + \mathbf{I}$, where $\Lambda = \sum_{i=1}^{J} \lambda_{l_i} + \sum_{t\in\mathcal{T}_S} \lambda_t$ and $\lambda_{l_i} = \max_{x\in\mathcal{RS}_i}(|\mathbf{Q}_l^{(i)}(x,x)|)$. Application of uniformization to a component matrix yields the transition matrix

$$\mathbf{P}_l^{(i)} = \mathbf{Q}_l^{(i)}/\lambda_{l_i} + \mathbf{I}, \tag{6}$$

which describes the transition probabilities of component i.

For each synchronized transition, $t \in \mathcal{T}_S$, we define matrices

$$\overline{\mathbf{E}}_t^{(i)} = \mathbf{I} - \mathbf{D}_t^{(i)}. \tag{7}$$

Matrix $\overline{\mathbf{E}}_t^{(i)}$ is a diagonal matrix that contains in position (x,x), $x \in \mathcal{RS}_i$, the probability that component i will not enable synchronized transition t in state x with respect to the events of a Poisson process with rate $\lambda_t s$. Due to the definition of the elements in $\mathbf{E}_t^{(i)}$, matrix $\mathbf{E}_t^{(i)} + \overline{\mathbf{E}}_t^{(i)}$ is a stochastic matrix. For each synchronized transition t, we have to distinguish between components that *may* disable the transition and those that *cannot* disable it. The latter include those components that do not synchronize at t. Thus, define $\mathcal{E}(t) = \{i | \exists 0 \leq x < n_i : \mathbf{e}_x \mathbf{E}_t^{(i)} \mathbf{e}^T < 1\}$ to be the set of components that may disable transition t. Set $\mathcal{E}(t)$ contains between 0 and J components. If all communication in the model is asynchronous, then $\mathcal{E}(t)$ contains at most one component. Furthermore, define set $\mathcal{S}(t)$, $t \in \mathcal{T}_S$, to contain all components i such that $\mathbf{E}_t^{(i)} \neq \mathbf{I}_{n_i}$. For local transition l_i, define $\mathcal{S}(l_i) = \{i\}$. In other words, the set $\mathcal{S}(t)$ contains all components (those that may and may not disable transition t) participating in transition t, whereas the set $\mathcal{E}(t)$ is a subset of $\mathcal{S}(t)$ and contains only those components that may disable t.

Matrix $\mathbf{P}$ then can be represented as

$$\mathbf{P} = \left(\sum_{i=1}^{J} \frac{\lambda_{l_i}}{\Lambda} \underbrace{\left(\bigotimes_{j=1}^{i-1} \mathbf{I}_{n_j} \right) \bigotimes \mathbf{P}_l^{(i)} \bigotimes \left(\bigotimes_{j=i+1}^{J} \mathbf{I}_{n_j} \right)}_{\mathbf{P}_{l_i}} \right) +$$
$$\left(\sum_{t\in\mathcal{T}_S} \frac{\lambda_t}{\Lambda} \underbrace{\left(\bigotimes_{i=1}^{J} \mathbf{E}_t^{(i)} \right)}_{\mathbf{P}_t} \right) +$$
$$\left(\sum_{t\in\mathcal{T}_S} \frac{\lambda_t}{\Lambda} \underbrace{\sum_{i\in\mathcal{E}(t)} \left(\bigotimes_{j=1}^{i-1} \mathbf{D}_t^{(j)} \right) \bigotimes \overline{\mathbf{E}}_t^{(i)} \bigotimes \left(\bigotimes_{j=i+1}^{J} \mathbf{I}_{n_j} \right)}_{\mathbf{P}_{\bar{t}_i}} \right). \tag{8}$$

The first sum describes local transitions in the different components (matrices $\mathbf{P}_{l_i}$), and the remaining sums describe synchronized transitions (matrices $\mathbf{P}_t$ and $\mathbf{P}_{\bar{t}_i}$). The second sum describes *actual* synchronization among components in $\mathcal{S}(t)$ at synchronized transition t (matrix $\mathbf{P}_t$). For each synchronized transition t, we have also the possibility that the transition may be disabled by at least one component. This *pseudo-synchronized* transition is described by the third sum. Each term in this sum (matrix $\mathbf{P}_{\bar{t}_i}$) describes the situation in which t is enabled by components with indices 1 through $(i-1)$, is disabled by component i, and may be enabled or disabled by components $(i+1)$ through J.

4 Structured Path-based Approach

In this section, we describe our path-based approach for computing transient rewards of large CTMCs. Often, approximate solutions are computed in these models, and thus it is also necessary to compute the upper and lower bounds on the solutions to determine whether the errors in the approximations are acceptable. This guarantee on the amount of error is especially relevant in many dependability analyses in which it is necessary to assure, for example, the minimal level of availability or reliability. We proceed by describing first the general concept of paths in our approach. Then we formalize the concept as an enumeration on strings and show how numerical computations can be done on paths. Because there may be many paths to enumerate, we also describe an equivalence relation among paths to reduce the number of paths that have to be explored and computed explicitly.

4.1 Conceptual Overview of the Approach

Conceptually, we consider paths at the higher level of local and synchronized transitions among components. By considering paths at this level, we gain several advantages. First, instead of considering each path individually, each step in our approach considers several paths at the same time. Thus, we can analyze a larger number of paths by computing a single vector-matrix product. Second, at the higher level of analysis, we can re-use previous computation. Third, by choosing the partition of the model and grouping of components appropriately, it is possible to choose some intermediate step between a complete numerical analysis and a path-based bound computation. The reward accumulated from analyzing a set of paths constitutes the lower bound for the reward of a CTMC. Knowing the probability of paths that have been considered, we can also compute the upper bound by assuming the maximum reward for all paths that have not been considered.

4.2 Numerical Analysis on Paths

The computation of path probabilities and rewards is efficient and simple in our approach because it requires only vector-matrix products of order n_i, rather than of order n. In our setting, paths are observable only by their transition labels, which correspond to different matrices. Let us define an alphabet $\mathcal{A} = \{l_1, \ldots, l_J\} \cup \mathcal{T}_S \cup (\cup_{t\in\mathcal{T}_S}\mathcal{E}(t))$, where again $l_i \in \{l_1, \ldots, l_J\}$ is a label signifying the occurence of a local transition in component i. Let $\mathcal{P}$ be the set of all strings made from symbols in $\mathcal{A}$, and let $\mathcal{P}^l \subseteq \mathcal{P}$ be the set of strings of length l. For any $\boldsymbol{\pi} \in \mathcal{P}$, $\boldsymbol{\pi}(i) \in \mathcal{A}$ is the ith element and $|\boldsymbol{\pi}|$ is the length of path $\boldsymbol{\pi}$. Thus, $\mathcal{P}$ is the set of all paths of a model and $\mathcal{P}^l$ contains all paths of length l. Now consider a specific path $\boldsymbol{\pi} \in \mathcal{P}^l$, and let $\mathbf{p}_0$ be the initial distribution and $\mathbf{r}$ be the reward vector. The reward after the transition of the path can be computed by

$$R(\boldsymbol{\pi}) = \mathbf{p}_0 \left(\prod_{k=1}^{|\boldsymbol{\pi}|} \mathbf{P}_{\boldsymbol{\pi}(k)} \right) \mathbf{r} \, , \tag{9}$$

where

$$\mathbf{P}_{l_i} = \mathbf{I}_{n_1^{i-1}} \bigotimes \mathbf{P}_l^{(i)} \bigotimes \mathbf{I}_{n_{i+1}^J},$$

$$\mathbf{P}_t = \bigotimes_{i=1}^{J} \mathbf{E}_t^{(i)}, \text{ and}$$

$$\mathbf{P}_{\bar{t}_i} = \bigotimes_{j=1}^{i-1} \mathbf{D}_t^{(j)} \bigotimes \overline{\mathbf{E}}_t^{(i)} \bigotimes \mathbf{I}_{n_{i+1}^J}.$$

The probability of the path $\boldsymbol{\pi}$ is given by

$$Prob(\boldsymbol{\pi}) = \prod_{k=1}^{|\boldsymbol{\pi}|} \frac{\lambda_{\boldsymbol{\pi}(k)}}{\Lambda} \tag{10}$$

where $\lambda_{\bar{t}_i} = \lambda_t$. The expected reward at time s and the expected accumulated reward in the interval $[0,s)$ can then be computed, respectively, in our path-based version by

$$E[R_s] = \sum_{l=0}^{\infty} \beta(\Lambda s, l) \sum_{\boldsymbol{\pi} \in \mathcal{P}^l} Prob(\boldsymbol{\pi}) R(\boldsymbol{\pi}) \tag{11}$$

$$E[AR_s] = \sum_{l=0}^{\infty} \frac{1}{\Lambda} \left(1 - \sum_{k=0}^{l} \beta(\Lambda s, k)\right) \sum_{\boldsymbol{\pi} \in \mathcal{P}^l} Prob(\boldsymbol{\pi}) R(\boldsymbol{\pi}) \tag{12}$$

The computation of path values can be done in an iterative manner. Define $\boldsymbol{\pi} \circ a$ as the path that results from $\boldsymbol{\pi}$ by appending $a \in \mathcal{A}$, and $a \circ \boldsymbol{\pi}$ as the path that results from $\boldsymbol{\pi}$ by prepending a. Let $\mathbf{p}[\boldsymbol{\pi}]$ be the state vector and $\mathbf{r}[\boldsymbol{\pi}]$ be the reward vector after $\boldsymbol{\pi}$ has been observed. Define $\mathbf{p}[\epsilon] = \mathbf{p}_0$ and $\mathbf{r}[\epsilon] = \mathbf{r}$, where ϵ is the empty string. Then vectors $\mathbf{p}[\boldsymbol{\pi}]$ and $\mathbf{r}[\boldsymbol{\pi}]$ can be computed iteratively:

$$\mathbf{p}[\boldsymbol{\pi} \circ a] = \mathbf{p}[\boldsymbol{\pi}]\mathbf{P}_a \text{ and } \mathbf{r}[a \circ \boldsymbol{\pi}] = \mathbf{P}_a \mathbf{r}[\boldsymbol{\pi}] \tag{13}$$

Theorem 1 *Vectors $\mathbf{r}[\boldsymbol{\pi}]$ and $\mathbf{p}[\boldsymbol{\pi}]$ can be represented in compact form as*

$$\mathbf{r}[\boldsymbol{\pi}] = \bigotimes_{i=1}^{J} \mathbf{r}^{(i)}[\boldsymbol{\pi}] \textit{ and } \mathbf{p}[\boldsymbol{\pi}] = \bigotimes_{i=1}^{J} \mathbf{p}^{(i)}[\boldsymbol{\pi}]$$

where

$$\mathbf{r}^{(i)}[\boldsymbol{\pi}] = \prod_{k=1}^{|\boldsymbol{\pi}|} \boldsymbol{\Phi}_{\boldsymbol{\pi}(k)}^{(i)} \mathbf{r}^{(i)},\ \mathbf{p}^{(i)}[\boldsymbol{\pi}] = \mathbf{p}_0^{(i)} \prod_{k=1}^{|\boldsymbol{\pi}|} \boldsymbol{\Phi}_{\boldsymbol{\pi}(k)}^{(i)}, \textit{ and}$$

$$\boldsymbol{\Phi}_{\boldsymbol{\pi}(k)}^{(i)} = \begin{cases} \mathbf{P}_l^{(i)} & \textit{if } \boldsymbol{\pi}(k) = l_i \\ \mathbf{E}_t^{(i)} & \textit{if } \boldsymbol{\pi}(k) = t \textit{ for } t \in \mathcal{T}_S \\ \mathbf{D}_t^{(i)} & \textit{if } \boldsymbol{\pi}(k) = \bar{t}_j \textit{ for } i < j \textit{ and } t \in \mathcal{T}_S \\ \overline{\mathbf{E}}_t^{(i)} & \textit{if } \boldsymbol{\pi}(k) = \bar{t}_i \textit{ for } t \in \mathcal{T}_S \\ \mathbf{I}_{n_i} & \textit{otherwise} \end{cases} .$$

Proof: We show the proof for $\mathbf{p}[\boldsymbol{\pi}]$ (the proof for $\mathbf{r}[\boldsymbol{\pi}]$ is completely analogous). Following (9), $\mathbf{p}[\boldsymbol{\pi}] = \mathbf{p}_0 \prod_{k=1}^{|\boldsymbol{\pi}|} \mathbf{P}_{\boldsymbol{\pi}(k)}$. Using basic properties of Kronecker products [10, 12] we obtain the following relation

$$\mathbf{p}[\boldsymbol{\pi}] = \mathbf{p}_0 \prod_{k=1}^{|\boldsymbol{\pi}|} \mathbf{P}_{\boldsymbol{\pi}(k)} = \bigotimes_{i=1}^{J} \mathbf{p}_0^{(i)} \prod_{k=1}^{|\boldsymbol{\pi}|} \bigotimes_{i=1}^{J} \boldsymbol{\Phi}_{\boldsymbol{\pi}(k)}^{(i)}$$
$$= \prod_{k=1}^{|\boldsymbol{\pi}|} \bigotimes_{i=1}^{J} \mathbf{p}_0^{(i)} \bigotimes_{i=1}^{J} \boldsymbol{\Phi}_{\boldsymbol{\pi}(k)}^{(i)} = \bigotimes_{i=1}^{J} \left(\prod_{k=1}^{|\boldsymbol{\pi}|} \mathbf{p}_0^{(i)} \boldsymbol{\Phi}_{\boldsymbol{\pi}(k)}^{(i)} \right)$$

which proves the theorem for $\mathbf{p}[\boldsymbol{\pi}]$. ■

The above results show three important aspects of the method. First, values for paths can be computed efficiently, and the representation of the vectors remains compact. Second, state reachability is considered implicitly via uniformization in our formulation. Unreachable states and their paths have probabilities of zero. Hence, our approach does not have to incur the extra step of having to analyze reachability explicitly as is normally required in Kronecker approaches. Third, a path of length l is the originating path of $|\mathcal{A}|$ paths of length $l+1$. The set $\mathcal{P}^l$ contains $(|\mathcal{A}|)^l$ paths, which is an enormous number, even for small values of J and $|\mathcal{T}_S|$. Thus, it is crucial to be able to enumerate paths efficiently and to determine equivalent paths to avoid redundant computation. This issue will be addressed in the next subsection; for now, we consider how to compute bounds from sets of paths.

Without loss of generality, we assume that all rewards are non-negative[1]. In this case, each path contributes a non-negative amount to the reward solution, and any finite summation of path rewards yields a lower bound for the required result. Thus, if $\mathcal{AP}^l \subseteq \mathcal{P}^l$ is a subset of the set of paths of length l and $l_{max} = \max_l(\mathcal{AP}^l \neq \emptyset)$, then the bounds for $E[R_s]$ are

$$\sum_{l=0}^{l_{max}} \beta(\Lambda s, l) \sum_{\boldsymbol{\pi} \in \mathcal{AP}^l} Prob(\boldsymbol{\pi}) R(\boldsymbol{\pi}) \le E[R_s] \le$$
$$\sum_{l=0}^{l_{max}} \beta(\Lambda s, l) \sum_{\boldsymbol{\pi} \in \mathcal{AP}^l} Prob(\boldsymbol{\pi}) R(\boldsymbol{\pi})$$
$$+ \left(1 - \sum_{l=0}^{l_{max}} \beta(\Lambda s, l) \sum_{\boldsymbol{\pi} \in \mathcal{AP}^l} Prob(\boldsymbol{\pi}) \mathbf{p}[\boldsymbol{\pi}] \mathbf{e}^T \right) \max_x(\mathbf{r}(x)) . \tag{14}$$

We use a derivation from [7] to formulate a tight upper bound on $E[AR_s]$. Thus, the bounds for $E[AR_s]$ are

$$\sum_{l=0}^{l_{max}} \frac{1}{\Lambda} \left(1 - \sum_{k=0}^{l} \beta(\Lambda s, k)\right) \sum_{\boldsymbol{\pi} \in \mathcal{P}^l} Prob(\boldsymbol{\pi}) R(\boldsymbol{\pi}) \le$$
$$E[AR_s] \le \sum_{l=0}^{l_{max}} \frac{1}{\Lambda} \left(1 - \sum_{k=0}^{l} \beta(\Lambda s, k)\right) \sum_{\boldsymbol{\pi} \in \mathcal{P}^l} Prob(\boldsymbol{\pi}) R(\boldsymbol{\pi})$$
$$+ \left[\left(1 - \sum_{k=0}^{l} \beta(\Lambda s, k)\right) - \frac{l+1}{\Lambda t} \left(1 - \sum_{k=0}^{l+1} \beta(\Lambda s, k)\right) \right] \max_x(\mathbf{r}(x)) \cdot t . \tag{15}$$

[1]Negative rewards can be scaled to meet this condition.

4.3 Equivalence Relation Among Paths

Although our approach computes a whole set of paths for each path $\boldsymbol{\pi} \in \mathcal{P}$, the set of paths that need to be explored continues to grow exponentially. Thus, an efficient realization of path generation and analysis is very crucial. We consider now the equivalence among paths to help in reducing the number of paths that actually need to be analyzed. For $a, b \in \mathcal{A}$ and $\mathcal{S}(a) \cap \mathcal{S}(b) = \emptyset$, then $\mathbf{p}[\boldsymbol{\pi} \circ a \circ b] = \mathbf{p}[\boldsymbol{\pi} \circ b \circ a]$ and $\mathbf{r}[a \circ b \circ \boldsymbol{\pi}] = \mathbf{r}[b \circ a \circ \boldsymbol{\pi}]$. The reason is that matrix $\mathbf{P}_a$ modifies only the states of components in $\mathcal{S}(a)$ (and likewise, $\mathbf{P}_b$ modifies only those in $\mathcal{S}(b)$). This equivalence can be used to define an equivalence relation among paths. Let $\boldsymbol{\pi} \in \mathcal{P}^l$ and let $Perm(\boldsymbol{\pi})$ be a set of permutations on $\{1, \ldots, l\}$ such that

$$perm(x) = y \Rightarrow \forall z \in \{\min(x, y), \ldots \max(x, y)\} \\ : \mathcal{S}(\boldsymbol{\pi}(x)) \cap \mathcal{S}(\boldsymbol{\pi}(y)) \cap \mathcal{S}(\boldsymbol{\pi}(z)) = \emptyset \; .$$

All paths that result from $\boldsymbol{\pi}$ by reordering of the elements according to some permutation of $Perm(\boldsymbol{\pi})$ are equivalent to $\boldsymbol{\pi}$ and have identical probabilities and rewards. The transitive closure of $Perm()$ defines an equivalence relation for the set of paths. Let $[\boldsymbol{\pi}]_\sim$ be the equivalence class that contains $\boldsymbol{\pi}$ and let $n_\sim(\boldsymbol{\pi})$ be the cardinality of the class. Then, after $\boldsymbol{\pi}$ has been explored, all paths from $[\boldsymbol{\pi}]_\sim$ can be considered to have been explored, and the rewards $R(\boldsymbol{\pi})$ and $AR(\boldsymbol{\pi})$ and probability $Prob(\boldsymbol{\pi})$ are multiplied by $n_\sim(\boldsymbol{\pi})$ for the computation of the lower and upper bounds.

5 Implementation Issues

Our approach is implemented as a C++ module that takes as input local and synchronized transition matrices and initial state probability and reward vectors. These matrices and vectors can be generated for a given model by existing tools such as the APNN Toolbox [1, 3] or a modified version of Möbius [5].

The current implementation of our approach explores and computes paths in a depth-first manner to minimize memory use in storing intermediate results. To further reduce memory demand, intermediate results are reused during path extension. These intermediate results include component state probability vectors and component reward vectors, which are much smaller than the complete model vectors. It should be emphasized that at no time during the computation are the complete model state probability and reward vectors computed or stored explicitly.

Our path generation algorithm generates only *canonical* paths. A canonical path is a representative path for a class of equivalent paths. Canonical paths have unique ordering of their nodes such that other canonical paths can be generated by considering only their last nodes. Equivalent paths are then calculated from the canonical paths, thus helping to improve performance and memory use by avoiding path lookups and storage. The current implementation exploits only the equivalence of paths from local transitions. Even so, the computational saving from exploitation of equivalent paths is quite substantial, as is shown in the next section.

6 Experimental Results

We evaluate our approach by studying its performance in analyzing a model of a distributed information service system adapted from the model in [8]. The example model describes the propagation of faults across the components constituting the system. After describing the system, we discuss the performance results we obtained from analyzing various configurations of the model. In this example, we show that the spread of the bounds shrinks rapidly with an increasing path length.

6.1 Model Description

The information service system consists of a front-end module that interacts with four processing units. Each processing unit consists of two processors, one switch, one memory unit, and one database. Each of these components has its own repair facility. They all go through the cycle of *Working*, *Corrupted*, *Failed*, and *Repaired*. The *stochastic activity network* (SAN) model of the system is shown in Fig. 1.

Fault propagation in the system is modeled as follows:

- When the front-end is corrupted, it may propagate the error to any of the four processing units in which there are 2 working processors. Propagation occurs via the synchronized activities between the front-end and the processors in the processing units. The front-end or any of the processors may disable the synchronized activities. After propagating the error to a processing unit, the front-end may remain in the corrupted state and continue to propagate errors to other processing units until it fails or there are no more processing units to propagate the error.

- When both processors in a processing unit are corrupted, they both may propagate the error to their working switch and memory via a synchronized activity. Any of the involved components may disable the synchronized activity. After the error propagation, the processors remain in the corrupted state until they fail.

- When both the switch and memory of a processing unit are corrupted, they may propagate their errors to the working database via a synchronized activity. Any of these components may disable the activity. After propagating the error, the switch and memory remain in the corrupted state until they fail.

We vary activity rates in the submodels and among the synchronized activities, so the resulting model does not have symmetries that would allow it to be lumped. In total, the model has 21 submodels and 12 synchronized activities. Because each submodel has three states, the state space of the whole model has $3^{21} \approx 10.5$ billion states.

In the following subsections, we discuss the performance results of our approach in computing reliability and availability for various configurations of the model. In particular, in all three configurations, we computed the reliability measure at transient time point 10.0 and the availability measure at transient time point 0.1 when all components in the model were in the working state. The two measures were computed at different time points because the high repair rates in the model would make computation of availability lengthy. Other variants of uniformization, such as adaptive uniformization [14], could be adapted to address this issue in a path-based approach. In addition to showing that our approach works for all configurations by converging to the simulated solution computed by Möbius, we also show how the sizes of the submodels in the configurations affect the performance of our approach. Due to space constraints, we show numerical data for all three configurations, but graphs for only the large-submodel-size configuration (more details are given in the following subsections). The trends in

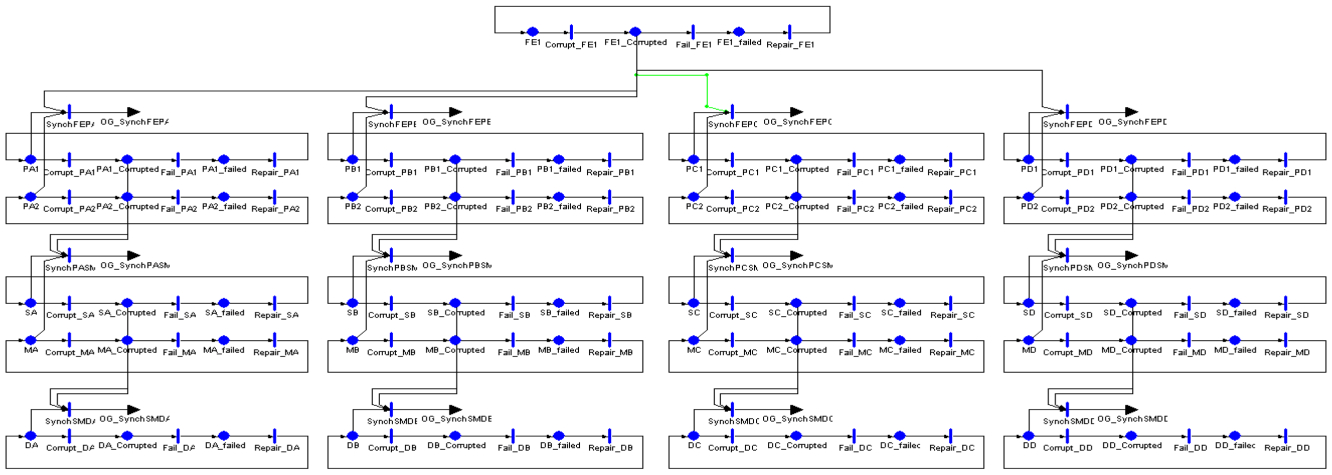


Figure 1. SAN model of a distributed information service system.

Table 1. Mean corruption, failure, and repair rates for the submodels

Submodels	Corruption Rate	Failure Rate	Repair Rate
Front-end	1/3000	10/1000	10/10
Processor A1	4/5000	4/1000	9/10
Processor B1	3/5000	4/1000	9/10
Processor C1	2/5000	4/1000	9/10
Processor D1	1/5000	4/1000	9/10
Processor A2	4/7000	4/1000	9/10
Processor B2	3/7000	4/1000	9/10
Processor C2	2/7000	4/1000	9/10
Processor D2	1/7000	4/1000	9/10
Switch A	4/11000	3/1000	8/10
Switch B	3/11000	3/1000	8/10
Switch C	2/11000	3/1000	8/10
Switch D	1/11000	3/1000	8/10
Memory A	4/13000	2/1000	7/10
Memory B	3/13000	2/1000	7/10
Memory C	2/13000	2/1000	7/10
Memory D	1/13000	2/1000	7/10
Database A	4/17000	1/1000	6/10
Database B	3/17000	1/1000	6/10
Database C	2/17000	1/1000	6/10
Database D	1/17000	1/1000	6/10

the graphs of the other two configurations are similar to that in the large-submodel-size configuration.

We evaluated all of our experiments on a workstation that had the AMD Athlon XP 2700+ processor running at 2.17 GHz with 512 MB of RAM. The operating system was Red Hat Linux 9.0 with mounted file systems. We compiled our implementation using the compiler g++ 3.3 with optimization flag -O3 only.

For all configurations, we observe that our implementation used a maximum of 1.32 MB of resident memory for data, code, and stack. In all, it used less than 5 MB of memory to include libraries. Tables 1 and 2 list the rates that we used for all of the activites in the model.

6.2 Small-submodel-size Configuration

We begin with the configuration that has the smallest submodel size. In this configuration, each submodel represents a component, such as the front-end, processor, or database, as shown in Fig. 1. Thus, each submodel has 3 states. Altogether there are 21 submodels and 12 synchronized activities in the configuration.

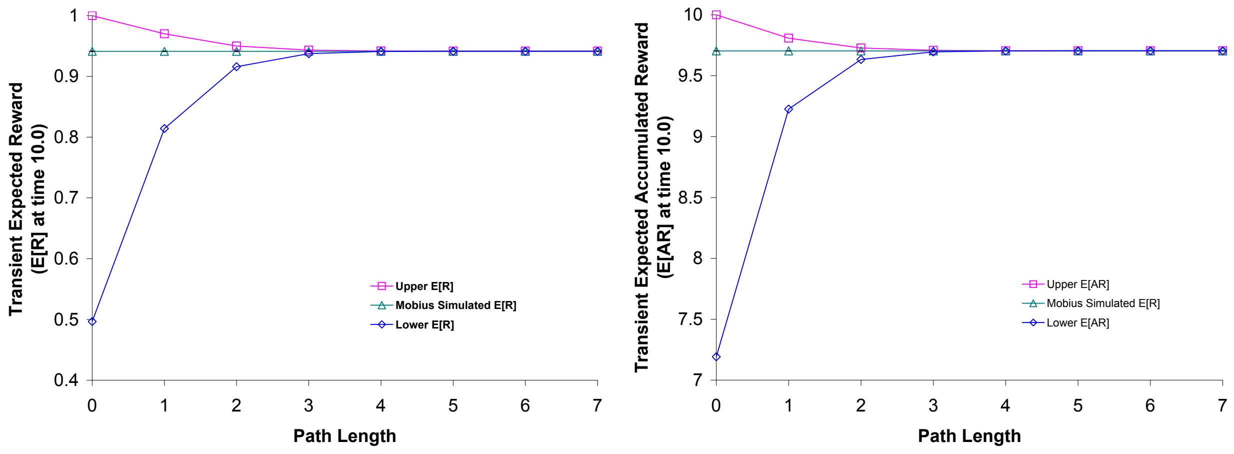
Table 2. Mean synchronized transition rates among submodels

Transition	Rate
SynchFEPA	4.5/3000
SynchFEPB	3.5/3000
SynchFEPC	2.5/3000
SynchFEPD	1.5/3000
SynchPASM	4.5/5000
SynchPBSM	3.5/5000
SynchPCSM	2.5/5000
SynchPDSM	1.5/5000
SynchSMDA	4.5/7000
SynchSMDB	3.5/7000
SynchSMDC	2.5/7000
SynchSMDD	1.5/7000

Tables 3 and 4 show the numerical results for the reliability and availability measures, respectively, computed for this configuration. Note the columns *Canonical Paths* and *Potential Paths*. The former shows the number of path sets (each set is represented by one canonical path) that our algorithm explored and computed; the latter shows the potential number of paths that would have to be explored and computed up to the corresponding path length if we had not used the path equivalence relation to identify equivalent paths. In general, the path equivalence relation can be better exploited when longer paths are considered. For this configuration and up to the path length of 5, the number of canonical paths computed to obtain the bounds was only 16.92% of the number of potential paths. Column *Time (sec)* shows the time taken to compute the paths.

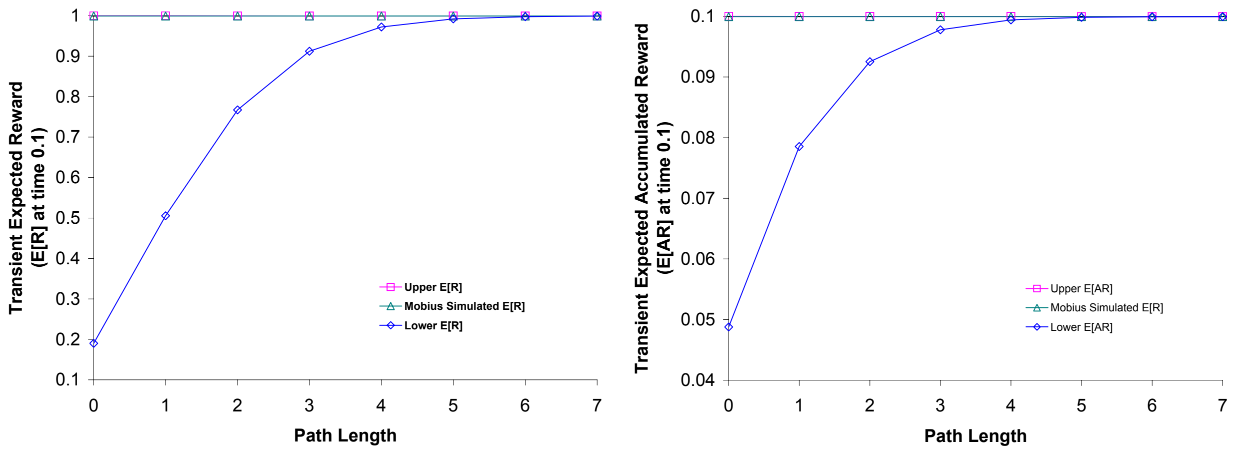
The Möbius simulation results are $0.9413224 \pm 1.456671 \times 10^{-4}$ for the expected instant-of-time reliability and $0.9993968 \pm 1.521793 \times 10^{-5}$ for the expected instant-of-time availability. The simulation results are $9.7036663 \pm 8.538210 \times 10^{-4}$ for the expected interval-of-time reliability and $0.0999397 \pm 8.711766 \times 10^{-7}$ for the expected interval-of-time availability. These results were obtained at a 95% confidence level.

Figs. 2 and 3 show the graphs of the reliability and availability measures, respectively, for the large-submodel-size configuration. Similar trends are found for the small-submodel-size configuration. The only difference is the time needed for the bounds to converge toward the exact solution. A comparison of the convergence rates for the three



(a) Expected Reward (b) Expected Accumulated Reward

Figure 2. Convergence of the bounds on the reliability measure toward the simulated solution obtained by Möbius for the large-submodel-size configuration.



(a) Expected Reward (b) Expected Accumulated Reward

Figure 3. Convergence of the bounds on the availability measure toward the simulated solution obtained by Möbius for the large-submodel-size configuration.

configurations is shown in Figs. 4 and 5 for the reliability and availability measures, respectively. Note that the larger configurations attain tighter bounds faster than the smaller configurations do.

6.3 Medium-submodel-size Configuration

For the medium-submodel-size configuration, we combined the two processors, the switch, and the memory in each processing unit into one submodel. There were four of these submodels in the complete model, and each of them had 81 states. In total, there were 9 submodels and 8 synchronized activities.

Tables 5 and 6 show the numerical results for the reliability and availability measures, respectively, computed for this configuration. Using the path equivalence relation, the number of canonical paths explored was only 19.79% of the potential number of paths.

6.4 Large-submodel-size Configuration

Lastly, for the large-submodel-size configuration, we combine the two processors, a switch, memory, and a database into one submodel. Of the three example configurations, this one has the largest submodels; each of them has 243 states. The complete model has 5 submodels and 4 synchronized activities.

Tables 7 and 8 show the numerical results for the reliability and availability measures, respectively, computed for this configuration. Using the path equivalence relation, the number of canonical paths explored to obtain the tight bounds shown in Figs. 2 and 3 was only 15.34% of the number of potential paths. Note that because there are fewer submodels, our algorithm can extend the paths to longer lengths more quickly for this configuration than for others. As a result, tighter bounds can be obtained more quickly.

Figs. 4 and 5 show the comparisons of the times needed to tighten the bounds around the solutions of the reliability and availability measures, respectively, for the three configurations. Except for the initial transient fluctuation[2] in the timing, which is due to the variability of time measurements below 0.1 seconds, the general trend is that the large-submodel-size configuration converges fastest among the three configurations.

[2]The erratic timing pattern is probably caused by the reading of data files (for the transition matrices and reward vectors) that resided on remote file systems.

Table 3. Numerical results of reliability for the small-submodel-size configuration

Path Length	Lower E[R]	Upper E[R]	Lower E[AR]	Upper E[AR]	Canonical Paths	Potential Paths	Time (sec)
0	0.476569	1	7.06248	10	0	0	0.03
1	0.801045	0.97127	9.17246	9.81318	73	73	0
2	0.911506	0.950875	9.62039	9.73034	3620	5402	0.07
3	0.936575	0.94363	9.69364	9.70912	165671	394419	1.96
4	0.940843	0.941912	9.70335	9.70519	7.65514e+06	2.87927e+07	89.65
5	0.941424	0.941607	9.70443	9.70462	3.55539e+08	2.10186e+09	4177.58

Table 4. Numerical results of availability for the small-submodel-size configuration

Path Length	Lower E[R]	Upper E[R]	Lower E[AR]	Upper E[AR]	Canonical Paths	Potential Paths	Time (sec)
0	0.189985	1	0.0487723	0.1	0	0	0.03
1	0.505399	0.999885	0.0785352	0.0999892	73	73	0.05
2	0.767226	0.999695	0.0925224	0.099979	3620	5402	0.07
3	0.912122	0.999537	0.0977801	0.0999733	165671	394419	1.96
4	0.972261	0.99945	0.0994149	0.0999709	7.65514e+06	2.87927e+07	89.74
5	0.99223	0.999414	0.0998466	0.0999701	3.55539e+08	2.10186e+09	4145.85

Table 5. Numerical results of reliability for the medium-submodel-size configuration

Path Length	Lower E[R]	Upper E[R]	Lower E[AR]	Upper E[AR]	Canonical Paths	Potential Paths	Time (sec)
0	0.488145	1	7.13742	10	0	0	0.03
1	0.808787	0.970572	9.20375	9.81036	33	33	0.02
2	0.914095	0.950388	9.62794	9.72892	830	1122	0.02
3	0.937153	0.943459	9.69495	9.70873	19611	37059	0.2
4	0.940939	0.941872	9.70353	9.70512	465834	1.22298e+06	4.45
5	0.941437	0.941599	9.70445	9.70461	1.10816e+07	4.03584e+07	105.25
6	0.941491	0.941562	9.70454	9.70455	2.63546e+08	1.33183e+09	2502.38

Table 6. Numerical results of availability for the medium-submodel-size configuration

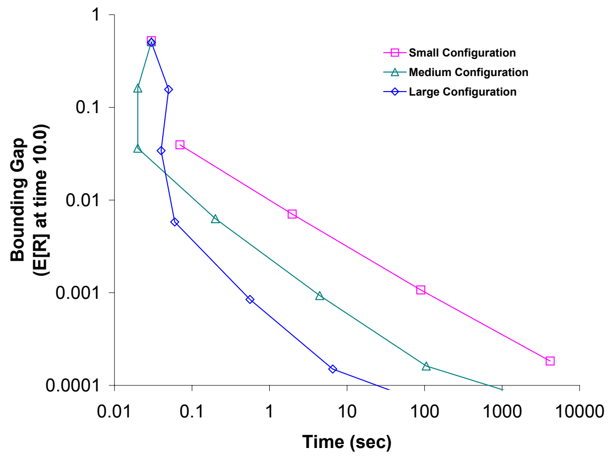
Path Length	Lower E[R]	Upper E[R]	Lower E[AR]	Upper E[AR]	Canonical Paths	Potential Paths	Time (sec)
0	0.19003	1	0.0487766	0.1	0	0	0.03
1	0.505475	0.999885	0.0785393	0.0999892	33	33	0.02
2	0.767289	0.999695	0.0925247	0.099979	830	1122	0.01
3	0.912156	0.999537	0.0977811	0.0999733	19611	37059	0.2
4	0.972275	0.99945	0.0994152	0.0999709	465834	1.22298e+06	4.46
5	0.992234	0.999414	0.0998467	0.0999701	1.10816e+07	4.03584e+07	105.21
6	0.997756	0.999402	0.0999456	0.0999699	2.63546e+08	1.33183e+09	2500.12

Table 7. Numerical results of reliability for the large-submodel-size configuration

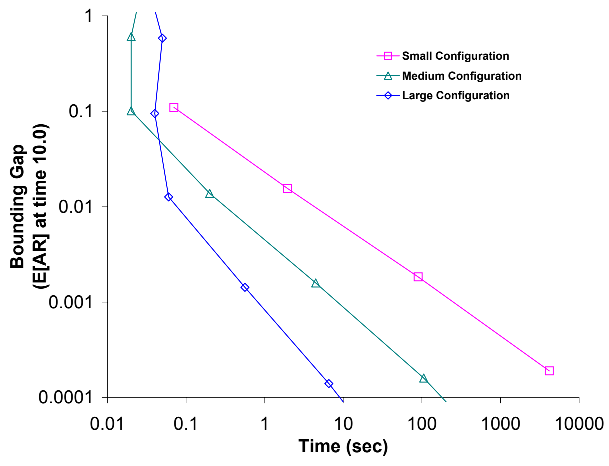
Path Length	Lower E[R]	Upper E[R]	Lower E[AR]	Upper E[AR]	Canonical Paths	Potential Paths	Time (sec)
0	0.496585	1	7.19164	10	0	0	0.03
1	0.814259	0.970064	9.22574	9.80831	17	17	0.05
2	0.915869	0.950043	9.63308	9.72792	232	306	0.04
3	0.937536	0.943342	9.69582	9.70846	2871	5219	0.06
4	0.941001	0.941846	9.70364	9.70507	35437	88740	0.56
5	0.941445	0.941595	9.70446	9.7046	437967	1.5086e+06	6.54
6	0.941492	0.941561	9.70454	9.70455	5.41298e+06	2.56462e+07	81.2
7	0.941496	0.941557	9.70454	9.70454	6.68963e+07	4.35985e+08	997.02

Table 8. Numerical results of availability for the large-submodel-size configuration

Path Length	Lower E[R]	Upper E[R]	Lower E[AR]	Upper E[AR]	Canonical Paths	Potential Paths	Time (sec)
0	0.190063	1	0.0487796	0.1	0	0	0.05
1	0.505529	0.999885	0.0785422	0.0999892	17	17	0.05
2	0.767334	0.999695	0.0925263	0.099979	232	306	0.03
3	0.912181	0.999537	0.0977817	0.0999733	2871	5219	0.09
4	0.972286	0.99945	0.0994154	0.0999709	35437	88740	0.57
5	0.992238	0.999414	0.0998468	0.0999701	437967	1.5086e+06	6.59
6	0.997757	0.999402	0.0999456	0.0999699	5.41298e+06	2.56462e+07	81.63
7	0.999066	0.999398	0.0999656	0.0999699	6.68963e+07	4.35985e+08	1009.77

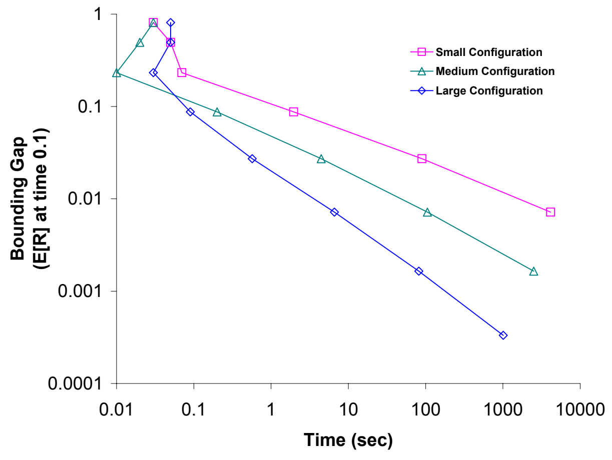


(a) Bounding Gap of Expected Reward

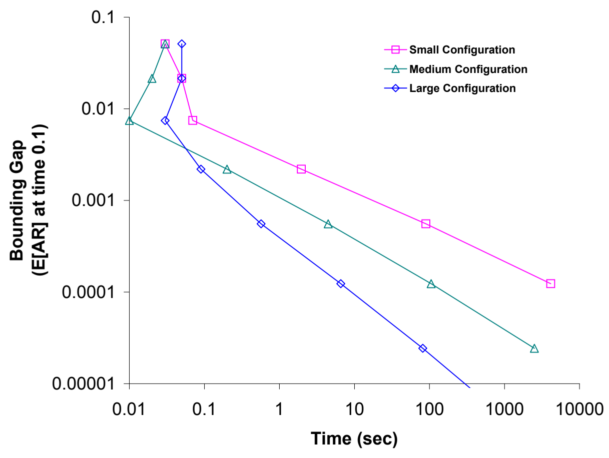


(b) Bounding Gap of Expected Accumulated Reward

Figure 4. Comparison of times to achieve tight bounds on the reliability measure.



(a) Bounding Gap of Expected Reward



(b) Bounding Gap of Expected Accumulated Reward

Figure 5. Comparison of times to achieve tight bounds on the availability measure.

7 Algorithmic Cost Analysis

We briefly compare the computational costs of the path-based approach against a standard numerical solution using uniformization. For the comparison, we make the following assumptions, which hold in many practical examples. The number of iterations of the uniformization approach is on the order of $O(\alpha s) \approx O(\Lambda s)$ assuming that $\Lambda \approx \alpha$. The number of states for each component is $O(n_1)$, and the number of reachable states is $O({n_1}^J)$. Matrices are sparse such that the effort for a vector-matrix product computation is $O(n_1)$ and $O({n_1}^J)$ for the path-based and complete approach, respectively. The cardinality of $\mathcal{A}$ equals L.

We begin with a comparison of the memory requirements and consider a depth-first enumeration of paths for the path-based approach. Furthermore, we assume that all intermediate results are stored, if they are required later. This means that up to $O(\Lambda s)$ intermediate results have to be stored and the storage requirements are in $O(n_1 \Lambda s)$. For a Kronecker-based representation of standard uniformization, storage requirements are thus $O(n_1^J)$. Therefore, the path-based approach requires less memory if $\Lambda s \leq n_1^{J-1}$. Since we are mainly interested in the analysis of large models that are partitioned into components of similar size, the path-based approach often requires less memory, even if all intermediate results are stored. Take, as an example, a system with 4 components and 10^2 states per component. Uniformization requires $O(10^8)$ memory, whereas the path-based approach requires $O(\Lambda s 10^2)$. For $\Lambda s < 10^6$, the path-based approach needs less memory. Furthermore, in uniformization each vector element has to be accessed in each iteration step, whereas intermediate results of the path-based approach need to be accessed less frequently, especially if they belong to short paths.

The computational effort of standard uniformization is $O(\Lambda s {n_1}^J)$ under the assumptions introduced above. The effort of the path-based approach depends on the number of paths to be explored. The number of paths depends on L and the sizes of the equivalence classes. We consider here two extreme cases. If no symbols can be exchanged, then the number of paths of length k equals L^k. If all symbols can be exchanged, then only $\binom{L+k-1}{L-1}$ different paths of length k have to be explored. If we consider the worst case, in which symbols cannot be exchanged, then the effort of the path-based approach is $O(n_1 L^{\Lambda s})$, whereas the effort of uniformization is $O(\Lambda s {n_1}^J)$. For $L > n_1$, the effort for the exhaustive path-based approach is similar to that for the uniformization approach if $\Lambda s \approx J$, which holds only for very short time horizons. If we consider the best case with independent transitions, then the path-based approach becomes more efficient, but it still outperforms uniformization only for small values of Λs.

8 Conclusion and Future Work

We presented a new approach that extends the current numerical capability to analyze very large Markov models for transient measures. We demonstrated the approach by analyzing a 10.5-billion-state model and showed that the computed bounds converged to results that were obtained through simulation.

In our approach, we extended existing path- and uniformization-based methods and formulated an equivalence relation among paths to identify sets of paths that are equivalent with respect to a reward measure. This relation allows us to explore multiple paths at the same time, thus significantly increasing the number of paths that need to be explored. Furthermore, the use of a structured representation for the state space and the direct computation of the desired reward measure (without ever storing the solution or iteration vectors) allow us to analyze very large models using a modest amount of storage. Additional benefits deriving from our approach include (1) implicit state reachability computation, (2) efficient path value computations via vector-matrix multiplications only, and (3) numerically stable computation.

Further work is being done to improve the approach. First, when bounds are computed, the goal of an algorithm is to find the most important paths first. The importance of a path depends on four factors:

1. The probability that a path of a given length will occur, which equals $\beta(\Lambda s, k)$ for a path of length k and can be computed a priori,
2. The path probability $Prob(\boldsymbol{\pi})$, which depends on the transitions in the path and also can be computed a priori,
3. The norm of the vector $\mathbf{p}[\boldsymbol{\pi}]$, which is 1 for paths consisting of local transitions only and depends otherwise on the enabling of synchronized transitions, and
4. The path reward value $\mathbf{r}[\boldsymbol{\pi}]$, which depends on the distribution $\mathbf{p}[\boldsymbol{\pi}]$ and the reward values.

A selective algorithm should first explore the paths that have high values for all four factors. Since $\mathbf{p}[\boldsymbol{\pi}]$ is computed from left to right and $\mathbf{r}[\boldsymbol{\pi}']$ is computed from right to left, one may think of using a two-phase algorithm that computes $\mathbf{p}[\boldsymbol{\pi}]$ from left to right and $\mathbf{r}[\boldsymbol{\pi}']$ from right to left. Since $\mathbf{r}[\boldsymbol{\pi} \circ \boldsymbol{\pi}'] = \mathbf{p}[\boldsymbol{\pi}]\mathbf{r}[\boldsymbol{\pi}']$, paths with high reward values and high probabilities can be selected from the subpaths.

Furthermore, since the path computations are independent, the path-based bounding approach can be easily parallelized for a network of PCs. Additionally, the approach can be extended to models with impulse rewards, since a path includes labels of all transitions. However, for that case, transitions with different impulse rewards have to be labeled with different labels.

Acknowledgment

*This material is based upon work supported by Pioneer Hi-Bred and by the National Science Foundation under Grant No. 0086096. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

†The research of Peter Buchholz has been partially supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Center SFB 358.

We would also like to thank Ms. Jenny Applequist for her editorial assistance.

References

[1] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. In *Quantitative Evaluation of Computing and Communication Systems*, LNCS 1469, pages 356–359. Springer, 1998.

[2] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal of Computing*, 13(3):203–222, 2000.

[3] P. Buchholz and P. Kemper. Numerical analysis techniques in the APNN toolbox. In *Proceedings of the Workshop on Formal Methods in Performance Evaluation and Applications*, pages 1–6, 1999.

[4] G. Ciardo and A. S. Miner. A data structure for the efficient Kronecker solution of GSPNs. In *Proceedings of PNPM'99: 8th International Workshop on Petri Nets and Performance Modeling*, pages 22–31, September 1999.

[5] D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders. Möbius: An extensible tool for performance and dependability modeling. In *Computer Performance Evaluation: Modelling Techniques and Tools*, LNCS 1786, pages 332–336. Springer, 2000.

[6] S. Derisavi, P. Kemper, and W. H. Sanders. Symbolic state-space exploration and numerical analysis of state-sharing composed models. In *Proceedings of NSMC '03: The Fourth International Conference on the Numerical Solution of Markov Chains*, pages 167–189, 2003.

[7] W. Grassmann. Means and variances of time averages in Markovian environments. *European Journal of Operational Research*, 31(1):132–139, 1987.

[8] R. R. Muntz and J. Lui. Computing bounds on steady-state availability of repairable computer systems. *Journal of the ACM*, 41(4):676–707, 1994.

[9] D. M. Nicol and D. L. Palumbo. Reliability analysis of complex models using SURE bounds. *IEEE Transactions on Reliability*, 44(1):46–53, March 1995.

[10] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. *Performance Evaluation Review*, 13(2):147–154, 1985.

[11] M. A. Qureshi and W. H. Sanders. A new methodology for calculating distributions of reward accumulated during a finite interval. In *Proceedings of the 26th International Symposium on Fault-Tolerant Computing*, pages 116–125, 1996.

[12] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[13] A. P. A. van Moorsel and B. R. Haverkort. Probabilistic evaluation for the analytical solution of large Markov models: Algorithms and tool support. *Microelectronics and Reliability*, 36(6):733–755, 1996.

[14] A. P. A. van Moorsel and W. H. Sanders. Adaptive uniformization. *ORSA Communications in Statistics: Stochastic Models*, 10(3):619–648, 1994.