ELSEVIER

European Journal of Operational Research 110 (1998) 166–186

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Iterative methods based on splittings for stochastic automata networks

Ertuğrul Uysal, Tuğrul Dayar *

Department of Computer Engineering and Information Science, Bilkent University, 06533 Bilkent, Ankara, Turkey

Received 1 February 1997; accepted 1 May 1997

Abstract

This paper presents iterative methods based on *splittings* (Jacobi, Gauss–Seidel, Successive Over Relaxation) and their block versions for *Stochastic Automata Networks* (SANs). These methods prove to be better than the power method that has been used to solve SANs until recently. With the help of three examples we show that the time it takes to solve a system modeled as a SAN is still substantial and it does not seem to be possible to solve systems with tens of millions of states on standard desktop workstations with the current state of technology. However, the SAN methodology enables one to solve much larger models than those could be solved by explicitly storing the global generator in the core of a target architecture especially if the generator is reasonably dense. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Markov processes; Stochastic automata networks; Tensor algebra; Splittings; Block methods

1. Introduction

Stochastic Automata Networks (SANs) [1–8] are performance modeling tools especially suited for parallel and distributed systems. SANs provide a methodology for modeling systems that have many components interacting with each other. The main advantage of the SAN approach is the considerable reduction in space needed for storing the performance model generated. A SAN is described by several stochastic automata that model the states of the individual components. The interactions among components are captured by *synchronizing events*. Synchronization among automata happens when a state change in one automaton causes a state change in other automata. On the other hand, a transition in one automaton whose rate depends on the states of other automata is called a *functional transition* [5, pp. 467–468].

Systems that do not have functional dependencies and synchronizing events among the components can be modeled by a single stochastic automaton for each component expressed as either an infinitesimal generator matrix or a probability transition matrix [5, pp. 464–466]. The global Markov chain can then be obtained by the tensor sum of the generator matrices of the individual automata in the continuous-time

* Corresponding author. Fax: +90-312-266-4126; e-mail: tugrul@cs.bilkent.edu.tr.

PII S0377-2217(97)00215-4

case and by the tensor product of the transition probability matrices of the individual automata in the discrete-time case. In this paper we concentrate on continuous-time models and aim at finding the stationary probability distribution of the global system. The derivations for discrete-time models are similar.

Synchronizing events introduce additional tensor products to the global generator. Since a tensor sum may be written as a sum of tensor products, the global matrix that corresponds to the system in the presence of synchronizing events can be expressed as a sum of Ordinary Tensor Products (OTP). The global generator when described in the form of a sum of tensor products is called the *descriptor* of the SAN [5, pp. 469–472]. In order to cope with functional transition rates, properties of tensor algebra have been extended [5, pp. 472–474]. *Generalized tensor algebra* deals with *Generalized Tensor Products* (GTP) and *Generalized Tensor Sums* (GTS) [3].

When a SAN is expressed by its descriptor, it is in compact form meaning that the model is described with minimal space requirements. However, this space efficiency is obtained at the expense of increased computation time even though an efficient vector-generalized tensor product multiplication algorithm may be employed when solving for the stationary distribution. The premultiplication of a SAN descriptor with a vector using this algorithm is referred to as *vector-descriptor multiplication* (see [8, pp. 12,21] or [7, p. 8]). The multiplication algorithm which happens to be a basic step in iterative methods such as the power method and Generalized Minimum Residual (GMRES) [6, pp. 516–522] requires certain conditions to be met [8, pp. 24–25]. The convergence of power method is generally slow for large Markov chains, and Krylov subspace methods such as GMRES are not very attractive unless they are accompanied by efficient preconditioners [5, pp. 484–486]. The methods discussed in this paper aim at improving the computation time of the stationary distribution of a SAN descriptor.

In Section 2, we show how one can split a SAN descriptor. Section 3 presents iterative methods based on splittings, which are in fact preconditioned power iterations, to compute the stationary vector of a Markov chain modeled as a SAN. Specifically we show how one can implement the classical iterative methods Jacobi, Gauss–Seidel (GS), and Successive Over Relaxation (SOR) for solving $\pi Q = 0$, $\|\pi\|_1 = 1$, where Q is the descriptor of the SAN and π its unknown stationary vector. Section 4 discusses the block versions of the methods. Numerical experiments with these methods appear in Section 5. Section 6 has concluding remarks.

2. The splitting of a SAN descriptor

In order to use classical iterative methods such as Jacobi, GS, and SOR for solving a SAN, the corresponding descriptor needs to be split. Here we give a suitable splitting for a SAN descriptor in the form $D - L - U$ [5, p. 126]. By a suitable splitting we mean one in which L, D, and U each consists of a sum of tensor products so that iterative methods of interest may be implemented in terms of the efficient vector–tensor product multiplication algorithm (see [6, pp. 516–517]).

The derivations of the splittings are based on the associativity of tensor products and distributivity of tensor product over matrix addition [9]. These two properties are valid for both OTP and GTP [8]. In other words, the splittings exist in both the nonfunctional (i.e., OTP) case and the functional (i.e., GTP) case. Obviously, limitations on the applicability of the efficient vector-descriptor multiplication algorithm still remain [8, pp. 13–24].

The descriptor of a SAN with N automata and E synchronizing events is given by

$$Q = \sum_{j=\ell}^{2E+N} \bigotimes_{i=\ell}^{N} Q_j^{(i)}. \tag{1}$$

However we can rewrite (1) as $Q = Q_\ell + Q_e + \bar{Q}_e$, where

$$Q_\ell = \bigoplus_{i=1}^{N} Q_\ell^{(i)}, \qquad Q_e = \sum_{e=1}^{E} \bigotimes_{i=1}^{N} Q_e^{(i)}, \qquad \bar{Q}_e = \sum_{e=1}^{E} \bigotimes_{i=1}^{N} \bar{Q}_e^{(i)}.$$

Assuming that the ith automaton has n_i states, the global generator will have $n = \prod_{i=1}^{N} n_i$ states. The generator $Q_\ell^{(i)}$ is comprised of local transitions in the ith automaton. *Local transitions* are those that do not affect the state of other automata, and they therefore exclude synchronizing transitions. The matrix $Q_e^{(i)}$ is the generator of the ith automaton corresponding to the synchronizing event labeled e. The diagonal matrix $\bar{Q}_e^{(i)}$ is the corrector of $Q_e^{(i)}$ as described in [3].

The splitting for the descriptor of a SAN that follows from the lemmas in Appendix A is summarized in a theorem.

Theorem 2.1. *The descriptor of a SAN given by* $Q(= Q_\ell + Q_e + \bar{Q}_e)$ *can be split as* $Q = D - L - U$, *where* D *is diagonal,* L *is strictly lower triangular, and* U *is strictly upper triangular. In particular*

$$\begin{aligned} Q &= Q_\ell + Q_e + \bar{Q}_e \\ &= (D_\ell - L_\ell - U_\ell) + (D_e - L_e - U_e) + \bar{Q}_e \\ &= \underbrace{(D_\ell + D_e + \bar{Q}_e)}_{D} - \underbrace{(L_\ell + L_e)}_{L} - \underbrace{(U_\ell + U_e)}_{U}. \end{aligned}$$

Moreover, $D, L,$ *and* U *each may be written in the form of a sum of tensor products.*

The following example better illustrates the concept of splitting a SAN descriptor.

Example 2.2. The example SAN appears in [5, pp. 470–472]. It is composed of two automata and two synchronizing events (i.e., $N = E = 2$) with $n_1 = 2$, $n_2 = 3$. For the first automaton, we have

$$Q_\ell^{(1)} = \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & 0 \end{pmatrix},$$

$$Q_{e_1}^{(1)} = \begin{pmatrix} 0 & 0 \\ \lambda_2 & 0 \end{pmatrix}, \qquad \bar{Q}_{e_1}^{(1)} = \begin{pmatrix} 0 & 0 \\ 0 & -\lambda_2 \end{pmatrix},$$

$$Q_{e_2}^{(1)} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \bar{Q}_{e_2}^{(1)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

For the second automaton, we have

$$Q_\ell^{(2)} = \begin{pmatrix} -\mu_1 & \mu_1 & 0 \\ 0 & -\mu_2 & \mu_2 \\ 0 & 0 & 0 \end{pmatrix},$$

$$Q_{e_1}^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad \bar{Q}_{e_1}^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$Q_{e_2}^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_3 & 0 & 0 \end{pmatrix}, \qquad \bar{Q}_{e_2}^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\mu_3 \end{pmatrix}.$$

The global generator of the example SAN is given by

$$\begin{aligned} Q &= Q_f + Q_e + \bar{Q}_e \\ &= \bigoplus_{i=1}^{N} Q_f^{(i)} + \sum_{e=1}^{E} \bigotimes_{i=1}^{N} Q_e^{(i)} + \sum_{e=1}^{E} \bigotimes_{i=1}^{N} \bar{Q}_e^{(i)} \\ &= Q_f^{(1)} \oplus Q_f^{(2)} + Q_{e_1}^{(1)} \otimes Q_{e_1}^{(2)} + Q_{e_2}^{(1)} \otimes Q_{e_2}^{(2)} + \bar{Q}_{e_1}^{(1)} \otimes \bar{Q}_{e_1}^{(2)} + \bar{Q}_{e_2}^{(1)} \otimes \bar{Q}_{e_2}^{(2)}. \end{aligned}$$

Hence

$$Q = \begin{pmatrix} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\mu_2) & \mu_2 & 0 & \lambda_1 & 0 \\ \mu_3 & 0 & -(\lambda_1+\mu_3) & 0 & 0 & \lambda_1 \\ \lambda_2 & 0 & 0 & -(\lambda_2+\mu_1) & \mu_1 & 0 \\ \lambda_2 & 0 & 0 & 0 & -(\lambda_2+\mu_2) & \mu_2 \\ \lambda_2+\mu_3 & 0 & 0 & 0 & 0 & -(\lambda_2+\mu_3) \end{pmatrix}. \tag{2}$$

From Theorem 2.1, we have

$$\begin{aligned} Q &= D - L - U \\ &= (D_f + D_e + \bar{Q}_e) - (L_f + L_e) - (U_f + U_e), \end{aligned}$$

where D_f, L_f, U_f are obtained from Lemma A.7 and D_e, L_e, U_e are obtained from Lemma A.8. In the following, we use $I_{n_i:n_j}$ to represent an identity matrix of size $\prod_{k=i}^{j} n_k$ when $i \leqslant j$, else a one. I_k and O_k are identity and zero matrices of order k, respectively. Then from the lemmas, we have

$$\begin{aligned} D &= D_f + D_e + \bar{Q}_e \\ &= \sum_{i=1}^{N} (I_{n_1:n_{i-1}} \otimes D_f^{(i)} \otimes I_{n_{i+1}:n_N}) + \sum_{e=1}^{E} \bigotimes_{i=1}^{N} D_e^{(i)} + \sum_{e=1}^{E} \bigotimes_{i=1}^{N} \bar{Q}_e^{(i)} \\ &= D_f^{(1)} \otimes I_3 + I_2 \otimes D_f^{(2)} + D_{e_1}^{(1)} \otimes D_{e_1}^{(2)} + D_{e_2}^{(1)} \otimes D_{e_2}^{(2)} + \bar{Q}_{e_1}^{(1)} \otimes \bar{Q}_{e_1}^{(2)} + \bar{Q}_{e_2}^{(1)} \otimes \bar{Q}_{e_2}^{(2)} \\ &= \begin{pmatrix} -\lambda_1 & 0 \\ 0 & 0 \end{pmatrix} \otimes I_3 + I_2 \otimes \begin{pmatrix} -\mu_1 & 0 & 0 \\ 0 & -\mu_2 & 0 \\ 0 & 0 & 0 \end{pmatrix} + 0_2 \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ &\quad + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes 0_3 + \begin{pmatrix} 0 & 0 \\ 0 & -\lambda_2 \end{pmatrix} \otimes I_3 + I_2 \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\mu_3 \end{pmatrix}. \end{aligned}$$

For L, we have

$$\begin{aligned} L = L_f + L_e &= \sum_{i=1}^{N} (I_{n_1:n_{i-1}} \otimes L_f^{(i)} \otimes I_{n_{i+1}:n_N}) + \sum_{e=1}^{E} \sum_{k=1}^{N} \bigotimes_{i=1}^{k-1} D_e^{(i)} \otimes L_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)} \right) \\ &= L_f^{(1)} \otimes I_3 + I_2 \otimes L_f^{(2)} + L_{e_1}^{(1)} \otimes Q_{e_1}^{(2)} + D_{e_1}^{(1)} \otimes L_{e_1}^{(2)} + L_{e_2}^{(1)} \otimes Q_{e_2}^{(2)} + D_{e_2}^{(1)} \otimes L_{e_2}^{(2)} \end{aligned}$$

$$= 0_2 \otimes I_3 + I_2 \otimes 0_3 + \begin{pmatrix} 0 & 0 \\ -\lambda_2 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} + 0_2 \otimes \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$
$$+ \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_3 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -\mu_3 & 0 & 0 \end{pmatrix}.$$

Finally for U, we have

$$U = U_\ell + U_e$$
$$= \sum_{i=1}^{N} (I_{n_1:n_{i-1}} \otimes U_\ell^{(i)} \otimes I_{n_{i+1}:n_N}) + \sum_{e=1}^{E} \sum_{k=1}^{N} \left(\bigotimes_{i=1}^{k-1} D_e^{(i)} \right) \otimes U_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)} \right)$$
$$= U_\ell^{(1)} \otimes I_3 + I_2 \otimes U_\ell^{(2)} + U_{e_1}^{(1)} \otimes Q_{e_1}^{(2)} + D_{e_1}^{(1)} \otimes U_{e_1}^{(2)} + U_{e_2}^{(1)} \otimes Q_{e_2}^{(2)} + D_{e_2}^{(1)} \otimes U_{e_2}^{(2)}$$
$$= \begin{pmatrix} 0 & -\lambda_1 \\ 0 & 0 \end{pmatrix} \otimes I_3 + I_2 \otimes \begin{pmatrix} 0 & -\mu_1 & 0 \\ 0 & 0 & -\mu_2 \\ 0 & 0 & 0 \end{pmatrix} + 0_2 \otimes \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} + 0_2 \otimes 0_3$$
$$+ 0_2 \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_3 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes 0_3.$$

The global generator matrix given in (2) may be verified by computing $D - L - U$. In Section 3, we present three iterative methods that follow from the splitting in Theorem 2.1.

3. Iterative methods based on splittings

Chapter 3 of [5] discusses iterative solution methods based on splittings for Markov chains. A summary of the convergence properties of these methods may be found in [10, pp. 26–28]. In all cases the problem may be formulated as one of computing a nontrivial solution to a homogeneous system of linear algebraic equations with a singular coefficient matrix under a normalization constraint. That is, the $(1 \times n)$ unknown vector π in

$$\pi Q = 0, \qquad \|\pi\|_1 = 1 \tag{3}$$

is sought. Concerning notation, all vectors are probability, hence row, vectors. The methods based on splittings amount to using the power method with an iteration matrix that corresponds to the particular splitting until a predetermined stopping criterion is met. We should also remark that the efficient vector–(generalized) tensor product multiplication algorithm used by the methods of interest has a time complexity of order $\mathcal{O}\left(\prod_{i=1}^{N} n_i \sum_{i=1}^{N} n_i\right)$. This complexity result assumes that all matrices in a tensor product are dense. In reality, some of these matrices are identity and zero, some are diagonal, and the remaining sparse. See, for instance, the matrices forming the descriptor in Example 2.2.

3.1. Jacobi

In matrix notation, applying the Jacobi method to a homogeneous linear system as in (3) is equivalent to applying the power method to the iteration matrix $(L+U)D^{-1}$; that is,

$$\pi^{(k+1)} = \pi^{(k)}(L+U)D^{-1}, \quad k = 0, 1, \ldots,$$

where Q is split as $D-(L+U)$. As it can be seen from the given formulation, each iteration may be implemented in two steps. First, postmultiply the most recent approximation $\pi^{(k)}$ with $(L+U)$, which is a sum of tensor products, and obtain $y^{(k)}$. Then postmultiply $y^{(k)}$ with D^{-1}. This last step can be implemented by multiplying the reciprocal of each diagonal element in D with the corresponding element of $y^{(k)}$ to give $\pi^{(k+1)}$.

3.2. Gauss–Seidel

In matrix notation, applying GS to a homogeneous system as in (3) is equivalent to applying the power method to the iteration matrix $U(D-L)^{-1}$. However, in order to employ the efficient vector–tensor product multiplication algorithm, we propose a slightly different implementation of the method. A backward GS iteration corresponds to the splitting $Q=(D-L)-U$ and may be written as

$$\pi^{(k+1)}(D-L) = \pi^{(k)}U, \quad k = 0, 1, \ldots$$

The right hand side of the iteration requires the use of vector–tensor product multiplication. Once the right hand side is computed as $b^{(k)}$, the next step involves solving the lower triangular system of equations $\pi^{(k+1)}(D-L) = b^{(k)}$. Similarly one can define forward GS using the splitting $Q=(D-U)-L$. In order to employ the efficient vector–tensor product multiplication algorithm, we should examine the nonzero structure of the matrix $(D-L)$.

D is a diagonal matrix of order $n = \prod_{i=1}^{N} n_i$ from which all the diagonal elements of $(D-L)$ come. That is, none of the nonzero elements of L, a strictly lower triangular matrix, appear along the diagonal of $(D-L)$.

By considering Lemmas A.7, A.8 and relabeling L_ℓ as $L_{e=0}$, we can rewrite L as

$$\begin{aligned}
L &= \sum_{i=1}^{N} I_{n_1:n_{i-1}} \otimes L_\ell^{(i)} \otimes I_{n_{i+1}:n_N} + \sum_{e=1}^{E}\sum_{k=1}^{N} \left(\bigotimes_{i=1}^{k-1} D_e^{(i)}\right) \otimes L_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)}\right) \\
&= \sum_{e=0}^{E} \left(\sum_{k=1}^{N} \left(\bigotimes_{i=1}^{k-1} D_e^{(i)}\right) \otimes L_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)}\right)\right) \\
&= \sum_{e=0}^{E} L_e^{(1)} \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right) + \sum_{e=0}^{E} D_e^{(1)} \left(\sum_{k=2}^{N} \left(\bigotimes_{i=2}^{k-1} D_e^{(i)}\right) \otimes L_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)}\right)\right) \\
&= \sum_{e=0}^{E} Q_e^{L},
\end{aligned}$$

where all Q_e^L are strictly lower triangular matrices formed by summing similar tensor products. For Q_0^L (i.e., L_ℓ), all matrices except $L_0^{(i)}$ in the tensor products are identity matrices.

Similarly, using Lemmas A.6–A.8 and relabeling $\bar{Q}_e^{(i)}$ as $D_{e+E}^{(i)}$ for $e = 1, 2, \ldots, E$, we get

$$D = \sum_{i=1}^{N} I_{n_1:n_{i-1}} \otimes D_\ell^{(i)} \otimes I_{n_{i+1}:n_N} + \sum_{e=1}^{E} \bigotimes_{i=1}^{N} D_e^{(i)} + \sum_{e=1}^{E} \bigotimes_{i=1}^{N} \bar{Q}_e^{(i)}$$
$$= \sum_{i=1}^{N} I_{n_1:n_{i-1}} \otimes D_\ell^{(i)} \otimes I_{n_{i+1}:n_N} + \sum_{e=1}^{2E} \bigotimes_{i=1}^{N} D_e^{(i)}.$$

Next we expose the block structure of $(D-L)$ and build the lower triangular solution on this structure.

Each Q_e^L matrix is the sum of N tensor products. All tensor products in this summation introduce nonzero entries to Q_e^L that are in mutually exclusive locations. In other words, each nonzero element in Q_e^L comes from a different tensor product. To see this, partition Q_e^L into n_1 blocks each of order $\prod_{i=2}^{N} n_i$. Its lower triangular blocks come from the term $L_e^{(1)} \otimes Q_e^{(2)} \otimes \cdots \otimes Q_e^{(N)}$ and its diagonal blocks come from the remaining terms (i.e., terms that have $D_e^{(1)}$ as the first factor). Observe that block (i,j) $i > j$ of Q_e^L can be expressed as $l_{e(i,j)}^{(1)} \left(\bigotimes_{k=2}^{N} Q_e^{(k)} \right)$, where $l_{e(i,j)}^{(1)}$ is the (i,j)th element of $L_e^{(1)}$. Similarly, block (j,j) of Q_e^L can be expressed as $d_{e(j,j)}^{(1)} \left(\sum_{k=2}^{N} \left(\bigotimes_{i=2}^{k-1} D_e^{(i)} \right) \otimes L_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)} \right) \right)$, where $d_{e(j,j)}^{(1)}$ is the jth diagonal element of $D_e^{(1)}$.

Given the above (first level) partitioning of L, our algorithm for solving π in the system $\pi(D-L) = b$ stems from the following observation. The linear equations for the subvector of π corresponding to the jth diagonal block of $(D-L)$, denoted $\bar{\pi}_j$, can be expressed as

$$\bar{\pi}_j D_{j,j} = \bar{b}_j + \sum_{i=j+1}^{n_1} \bar{\pi}_i \left(\sum_{e=0}^{E} l_{e(i,j)}^{(1)} \left(\bigotimes_{k=2}^{N} Q_e^{(k)} \right) \right), \tag{4}$$

or as

$$\bar{\pi}_j D_{j,j} = \bar{c}_j, \quad j = n_1, \ldots, 2, 1.$$

Here $D_{j,j}$ is the jth diagonal block of $(D-L)$, $\bar{b}_j$ and $\bar{c}_j$ are respectively the jth subvectors of b and c, the new right hand side.

At this point, we are left with the problem of solving $\bar{\pi}_j D_{j,j} = \bar{c}_j$. Fortunately, the block structure of the diagonal blocks $D_{j,j}$ is similar to that of the original matrix $(D-L)$. Each diagonal block at level 1 is a lower triangular matrix that can be expressed as a sum of tensor products. Thus,

$$D_{j,j} = \left(d_{\ell(j,j)}^{(1)} \otimes I_{n_2:n_N} + \sum_{k=2}^{N} I_{n_2:n_{k-1}} \otimes D_\ell^{(k)} \otimes I_{n_{k+1}:n_N} \right) + \sum_{e=1}^{2E} d_{e(j,j)}^{(1)} \left(\bigotimes_{k=2}^{N} D_e^{(k)} \right)$$
$$- \sum_{e=0}^{E} d_{e(j,j)}^{(1)} \left(\sum_{k=2}^{N} \left(\bigotimes_{i=2}^{k-1} D_e^{(i)} \right) \otimes L_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)} \right) \right),$$

where $d_{\ell(j,j)}^{(1)}$ is the jth diagonal element of $D_\ell^{(1)}$. Note that the diagonal elements of $D_{j,j}$ come from the first and the second terms. The strictly lower triangular elements come from the third term. Next we can partition each diagonal block $D_{j,j}$ into n_2 blocks each of order $\prod_{i=3}^{N} n_i$. This continues recursively until we have a system of order n_N (i.e., order of the last automaton) to solve. The first and the second terms of $D_{j,j}$ come into play only at the deepest level and the recursion is inherent in the third term. Hence, the algorithm we present for point GS is a recursive one. The lower triangular solution algorithm calls itself until the recursion ends at level N when a single iteration over the point equations is performed: the systems to be solved at level N are lower triangular.

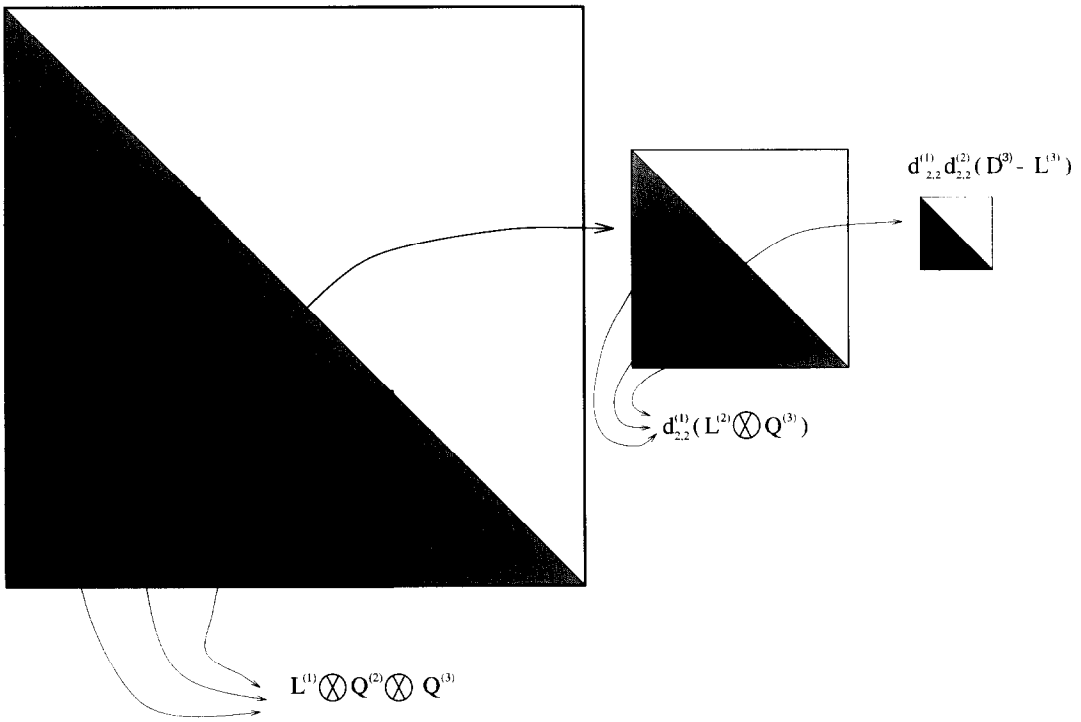

Fig. 1. Lower triangular part of $Q_1 \otimes Q_2 \otimes Q_3$ partitioned into blocks.

The illustrative example in Fig. 1 shows the partitioning of a three term tensor product. The lower triangular block structure of the tensor product $Q_1 \otimes Q_2 \otimes Q_3$ is emphasized. The dark grey shaded blocks of the product on the left come from the term $L_1 \otimes Q_2 \otimes Q_3$. The grey blocks on the left correspond to the three diagonal blocks each of order $n_2 n_3$. The partitioning of the second diagonal block $D_{2,2}$ is shown in the middle. The smallest matrix on the right is the second diagonal block of $D_{2,2}$.

3.2.1. Algorithm for solving $\pi(D - L) = b$

The algorithm discussed in this section solves the system $\pi(D - L) = b$ using the efficient vector–(generalized) tensor product multiplication algorithm when there are no *cyclic dependencies* in the SAN [8, pp. 20–22]. Here D and L are respectively diagonal and strictly lower triangular matrices. In the absence of cyclic dependencies, all tensor products in a SAN (see Eq. (1)) may be ordered (and relabeled) such that each matrix in each tensor product has entries with functional dependencies, if at all, only to the automata that come before itself in the given ordering. A SAN that lacks cyclic dependencies may be written in the form $Q^{(1)}, Q^{(2)}[\mathcal{Q}^{(1)}], Q^{(3)}[\mathcal{Q}^{(1)}, \mathcal{Q}^{(2)}], \ldots, Q^{(N)}[\mathcal{Q}^{(1)}, \ldots, \mathcal{Q}^{(N-1)}]$. The arguments in the square brackets of each matrix indicate dependencies that may exist among automata.

The recursive lower triangular solution algorithm for SANs:

SolveD-L$(id, states, first, \pi, b)$

1. $nright = n_{id+1} n_{id+2} \ldots n_N$
2. if $(id = N)$
   - $T = 0$
   - for $e = 1$ to $2E + N$

- $d = \prod_{i=1}^{id-1} d^{(i)}_{e(states[i],states[i])}[states]$
- $T = T + d\left(D_e^{(id)}[states] - L_e^{(id)}[states]\right)$

• Solve $\pi_{first-n_N+1:n_N} T = b_{first-n_N+1:n_N}$
• return

else

• $states[id] = n_{id}$
• SolveD-L$(id+1, states, first, \pi, b)$

3. $\pi first = first$
4. $bfirst = first - (n_{id} nright) + 1$
5. for $irow = n_{id}$ downto 2
   • $\pi first = \pi first - nright + 1$
   • $states[id] = irow$
   • for $e = 0$ to E
     - $b' = \pi_{\pi first:nright}\left(\bigotimes_{i=id+1}^{N} Q_e^{(i)}[states]\right)$
     - for $k = 1$ to $irow - 1$
       - ◇ Reset $states[(id+1), \ldots, N]$ to the first indexed states of automata $(id+1)$ to N
       - ◇ for $i = 1$ to $nright$
         - ⋆ $d = \prod_{j=1}^{id-1} d^{(j)}_{e(states[j],states[j])}$
         - ⋆ $b_{bfirst+(k-1)nright+i-1} = b_{bfirst+(k-1)nright+i-1} + \left(d \cdot l^{(id)}_{e(irow,k)}[states] b'_i\right)$
         - ⋆ Update $states[(id+1), \ldots, N]$ for automata $(id+1)$ to N
   • SolveD-L$(id+1, states, \pi first - nright + 1, \pi, b)$

For instance, transitions in automata 3 may depend only on the states of automata 1 and 2, but not on the states of others. Before we use the algorithm, we make sure the automata are ordered appropriately.

The initial call to the recursive algorithm is SolveD-L$(1, states, n, \pi, b)$. The first parameter $id(=1)$ corresponds to the level of block partitioning. It might also be thought of as the current level automaton number. The initial call at level 1 partitions the global descriptor into n_1 blocks each of order $\prod_{i=2}^{N} n_i$. The second parameter *states*, an array of size N, stores the state of each automaton to be used in function evaluations. For instance, if we are solving the ith diagonal block (see Eq. (4)) in the first call (i.e., no recursive calls have been made yet), the state of automaton 1 is i. The parameter *states* is also used to determine the scalar multipliers that form the diagonal blocks. For example, in order to solve the smallest block in Fig. 1, we need to multiply the lower triangular matrix $\left(D^{(3)} - L^{(3)}\right)$ with $d^{(1)}_{2,2}\, d^{(2)}_{2,2}$. See also step 2 in the SolveD-L algorithm; if e corresponds to the corrector of a synchronizing event, then $L_e^{(id)}[states] = 0$. Furthermore, we represent matrices arising from local automata by $e = 2E+1, \ldots, 2E+N$ in step 2. We determine both the coordinates of the scalar multipliers and the current states of lower indexed automata using *states*. The initial contents of *states* is irrelevant since it is updated when deemed necessary. The third parameter $first(=n)$ is set to the size of the unknown vector in the current call. The fourth parameter is the solution vector initially set to $\pi_i = 1/n\ \forall i$ and overwritten with the new approximation at each iteration. The last parameter b is the right hand side of the lower triangular system. The algorithm assumes that the generator matrices of automata are available globally. Since the algorithm implements a backward solution and computes the last unknown (subvector) first, we use $\pi_{first:n_N}$ to denote the subvector of π with first element π_{first} and length n_N.

Vector–tensor product multiplications arising from the local and synchronizing event generator matrices (see the for-loop on e in step 5 of the algorithm) may be reduced to scalar–vector multiplications (see the third statement from the bottom in step 5). For each block in a row, a vector–tensor product multiplication possibly with functional transitions depending on the current state of the automata at that level (see *irow* in step 5) is required. An efficient approach is to loop on blocks in a row (see the for-loop on k in step 5)

because in each row all blocks below the diagonal have a common vector–tensor product multiplication and all functional entries in these blocks use the same *irow* value while being evaluated (see Eq. (4)). It is also observed that many matrices encountered in the test problems are zero, have zero diagonals, have zero strictly lower or strictly upper triangular parts. We have taken advantage of this as well. The actual timings depend heavily on such implementation details.

3.2.2. Gauss–Seidel algorithm

The following algorithm implements GS for solving a SAN in the functional case assuming that a splitting $(D - L - U)$ for the SAN descriptor and an initial approximation π are available. Remember that the triangular solution algorithm overwrites the input approximation with the new approximation on return from the call. Upon termination *it* gives the number of iterations performed.

The GS algorithm using SolveD-L

$it = 0$

- Repeat until convergence
  - $it = it + 1$
  - Compute $b = \pi U$
  - SolveD-L$(1, states, n, \pi, b)$

3.3. Successive over relaxation

The SOR method can be expressed as $\pi_{\mathrm{SOR}}^{(k+1)} := w\pi_{\mathrm{GS}}^{(k+1)} + (1 - w)\pi_{\mathrm{SOR}}^{(k)}$, where $\pi_{\mathrm{GS}}^{(k+1)}$ is the $(k + 1)$st approximation of GS, $\pi_{\mathrm{SOR}}^{(k)}$ is the kth approximation of SOR, and w is the relaxation parameter (i.e., a weighing factor) satisfying $0 < w < 2$.

4. Block methods

We argued that one can perform a lower triangular backward solution on the blocks of order n_N at the final depth of recursion: see the third bullet in step 2 of the SolveD-L algorithm. Instead of doing this, one may choose to solve these blocks directly, i.e., by Gaussian elimination (GE). This approach, which we call block GS, is likely to decrease the number of iterations since blocks at each iteration are solved exactly. When doing this, the right hand side b that goes into SolveD-L is computed in a slightly different manner. Now one must exclude the strictly upper triangular parts of the matrices corresponding to the last automaton from the multiplication. That is,

$$b^{(k)} = \pi^{(k)}\left(\sum_{i=1}^{N-1} I_{n_1:n_{i-1}} \otimes U_f^{(i)} \otimes I_{n_{i+1}:n_N} + \sum_{e=1}^{E}\sum_{k=1}^{N-1}\left(\bigotimes_{i=1}^{k-1} D_e^{(i)}\right) \otimes U_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)}\right)\right).$$

What has been excluded from the new right hand side must be included at level N in step 2 of the recursive backward solution algorithm. The matrix that corresponds to automaton N at step 2 must include the whole matrices that correspond to synchronizing events, their diagonal correctors and to local automata, not just the lower triangular parts. The matrices of order n_N formed in this way at the deepest level of recursion for each one of the $\prod_{i=1}^{N-1} n_i$ diagonal blocks will be solved using GE. Even though the space requirement is larger, if the decrease in the iteration count is substantial, the cost of solving the blocks directly is offset by a smaller overall solution time. Another possibility is to terminate recursion earlier and solve larger blocks. Also one can choose to generate and store larger blocks at the outset, then use these at each iteration (see the concept of *grouping* in [7, pp. 13–14]).

In the experiments, we noticed an interesting feature of block methods.

Remark 4.1. For a block coefficient matrix with lower (upper) triangular diagonal blocks in Eq. (3), backward (forward) block GS/SOR is equivalent to point GS/SOR.

The remark follows from inspecting the linear equations in systems with the described nonzero structure. Section 5 provides results of experiments with three problems.

5. Numerical results

In order to make illuminating comparisons, we implemented power, Jacobi, GS, and SOR methods. We carried out experiments using both backward and forward versions of GS (and hence SOR) together with block versions of Jacobi, GS, and SOR methods. In block implementations, we terminate recursion at the deepest level and solve the blocks of order n_N using GE as discussed in Section 4. During the experiments we used a stopping criterion of 10^{-10} between consecutive approximations. We ran all the experiments on SUN Sparcstation 4's which had 32 MB of RAM. All the algorithms are implemented in C++ and the new methods are incorporated into the software package PEPS [2]. Regardless of its size, each problem produced a smaller number of iterations in either the backward or the forward approach; we present results of the better approach.

We experimented with three problems. The first two, resource sharing and three queues, are explained in [8]. The third one, the model of a mass storage system, appears in [11]. For the mass storage example, we experimented with different orderings of the automata. Obviously, ordering of automata is likely to have an effect on the iteration count. The efficient vector–tensor product multiplication algorithm itself imposes an ordering on the automata. In order to use other orderings, a permutation vector may need to be introduced to the multiplication algorithm. We experimented with orderings that do not require permutation. We also tried orderings different from the original ordering by taking advantage of the position of identity matrices in tensor products. Such orderings follow from Lemma 5.5 and its companion remark in [8, p. 16].

Modeling with SANs still is in its infancy and only recently have researchers started considering large and complex problems. Issues related to cyclic dependencies are currently under investigation. Lemma 5.8 and Theorem 5.2 in [8] show how one can handle cyclic dependencies in generalized tensor products. If the functional dependency graph is fully connected there is not much that can be done to improve the complexity of vector-generalized tensor product multiplication. On the other hand, if the cutset of the cycles in the dependency graph has a small number of automata, then a more efficient vector-generalized tensor product multiplication algorithm can be used. However, this multiplication will still take much longer than that of a vector-generalized tensor product lacking cycles. The smaller the cutset, the better the improvement. Moreover, at the end of Section 6 in [8], it is indicated that Theorem 5.2 needs to be resorted only when routing probabilities associated with synchronizing events (i.e., descriptors of slave automata due to synchronizing events) are functional and result in cycles within the functional dependency graph. This situation is suspected to be rare by the authors of [8]. We have not seen such a case. However, it is still not impossible to have generalized tensor products with dependency cycles. We should emphasize that no attempt has been made to avoid cyclic dependencies in the modeling phase of the mass storage problem. In [7], the last paragraph of Section 4.3 discusses the results of some experiments with artificially created cyclic dependencies. There it is mentioned that cycles have a detrimental effect on solution time, as expected. As for ordering the automata in the case of noncyclic dependencies, we think it should not be very difficult. It is an implementation issue. However, we have purposefully concentrated on orderings that do not require the introduction of a permutation vector. Searching for optimal orderings and relaxation

parameters when testing newly devised algorithms is a problem in its own right and we have not attempted experimenting with all $N!$ orderings of N automata.

The main advantage of using SANs is its memory efficiency as opposed to time efficiency. We implemented all of the above methods for sparse matrices in the Harwell–Boeing format so that a comparison can be made. The sparse matrices are generated using the descriptors, which are also stored in sparse format, and their generation times reported. We should remark that identity matrices arising in synchronizing events or local transitions are kept in a special data structure and do not contribute to the space complexity of the descriptor approach. The generation time of the descriptor in each problem is negligible and hence not reported. Since one is limited with a certain amount of core memory on a target architecture, we report results with sparse methods only in problems for which we could generate and store the global transition rate matrix. That we could solve larger problems using the sparse matrix approach if we had used a larger core is immaterial. In this work, we aim at investigating the "relative" worth of the SAN approach compared to the sparse matrix approach for the solution methods at hand on a target architecture. Research along other viable alternatives for handling large numbers of nonzeros in sparse matrices is also of interest to researchers (see [12], for instance). In the following w_* refers to the optimal relaxation parameter, *it* and *time* denote respectively the number of iterations and the CPU time (in seconds) to converge to the prespecified tolerance. The bold figures in Tables 1–3 indicate the best run times for the particular problem. Now we describe the problems.

The resource sharing problem has four parameters. The number of processes N, the number of processes P that can simultaneously access the critical resource, the rate $\lambda^{(i)}$ at which each process wakes up and tries to acquire the resource, and the rate $\mu^{(i)}$ at which each resource using the process releases the resource for $i = 1, 2, \ldots, N$. Each process is modeled using a single automaton with two states. There are N such automata implying a state space size of $n = 2^N$. This model does not have any synchronizing events; it has functional transition rates but no cyclic dependencies.

In our experiments we used $\lambda^{(i)} = 0.04$ and $\mu^{(i)} = 0.4$ for $i = 1, 2, \ldots, N$. Since all matrices are identical for the given $\lambda^{(i)}$ and $\mu^{(i)}$, reordering the automata is futile. The resource sharing problem does not converge for Jacobi and block Jacobi methods. As for backward block GS and SOR methods, they are expected to give (slightly) smaller iteration counts than their point versions when P is closer to N than to 0. This follows from Remark 4.1 and is particularly substantiated for the GS iteration. When P is small compared to N, many of the upper diagonal elements of the 2×2 matrices evaluate to zero and there is no advantage of using block methods. On the other hand, when P is larger, many functional rates evaluate to nonzero values and the block methods start to make a difference, however very little due to the extremely small block size.

Table 1
Results of experiments with the resource sharing problem

Prob.		Power		GS		SOR			Block GS		Block SOR		
N	P	*it*	*time*	*it*	*time*	w_*	*it*	*time*	*it*	*time*	w_*	*it*	*time*
12	1	142	83	2	**2**	1.0	2	**2**	2	**2**	1.0	2	**2**
12	6	222	131	26	23	1.3	18	16	26	22	1.3	18	**15**
12	11	222	123	28	25	1.3	18	16	26	22	1.3	18	**15**
16	1	188	2299	2	39	1.0	2	40	2	**38**	1.0	2	**38**
16	8	294	3793	32	613	1.3	22	420	32	592	1.3	22	**402**
16	15	294	3562	34	650	1.4	22	420	32	589	1.3	22	**402**
20	1	236	63,265	2	825	1.0	2	826	2	**740**	1.0	2	**740**
20	10	362	94,157	38	15,039	1.5	26	9777	38	13,764	1.4	24	**8734**
20	19	364	89,126	40	15,554	1.5	24	8891	40	14,311	1.5	24	**8673**

The results of the experiments are summarized in Table 1. Observe that block SOR takes approximately 1/10th of the time the power method takes for the case $N = 20$, $P = 19$.

The three queues problem is an open queueing network of three finite capacity queues respectively with capacities $C_1 - 1$, $C_2 - 1$, and $C_3 - 1$ in which customers from queues 1 and 2 (try to) join queue 3. The customers that come through queues 1 and 2 are referred to as type 1 and type 2 customers. The arrival and service rates of queue i are respectively given by λ_i and μ_i for $i = 1, 2$. Queue 3 has a service rate of μ_{3_1} for type 1 and a service rate of μ_{3_2} for type 2 customers. The network is modeled using 4 automata $\mathscr{A}^{(1)}$, $\mathscr{A}^{(2)}$, $\mathscr{A}^{(3_1)}$, $\mathscr{A}^{(3_2)}$ with respectively C_1, C_2, C_3, and C_3 states. The state space size is given by $n = C_1 C_2 C_3^2$. This model has both synchronizing events and functional rates; it does not have any cyclic dependencies. Details of this queueing network may be found in [8].

The parameters used in the experiments are $\lambda_1 = 0.4$, $\lambda_2 = 0.3$, $\mu_1 = 0.6$, $\mu_2 = 0.5$, $\mu_{3_1} = 0.7$, and $\mu_{3_2} = 0.2$. The automata are ordered as $\mathscr{A}^{(1)}$, $\mathscr{A}^{(2)}$, $\mathscr{A}^{(3_1)}$, $\mathscr{A}^{(3_2)}$. Backward SOR gives the best results. However, block versions of GS and SOR do not improve the iteration counts of their point counterparts since the matrices that correspond to the last automaton are all lower triangular. Block Jacobi is better than point Jacobi in this case, yet we do not think the results are interesting. So we present results only for point methods in Table 2. Note that point SOR takes only a quarter of the time the power method takes for the largest problem that has 1,000,000 states.

Fortunately, we were able to try all iterative methods in the mass storage problem (see [12]). The model is used to investigate the effects of interactive retrieval (get) and storage (put) requests, migration workload, and purging workload on a robotic tape library (RTL). The first (i.e., online storage) layer usually consists of magnetic disks which provide fast access time but at a relatively high cost per byte. The second (i.e., near-line storage) layer utilizes RTLs, and the third (i.e., offline storage) layer consists of free-standing tape drives with human operators performing the mounting and unmounting of media from the drives. Since the interest is mainly in the performance of RTLs, it is assumed that the system to be modeled only consists of an online and a nearline layer. The parameters in this problem are quite a few. The unit of time for the given parameters is minutes.

λ_g arrival rate of get requests to the system
λ_p arrival rate of put requests to the disk cache
h_g hit ratio of get requests at the disk cache
h_p hit ratio of put requests at the disk cache
μ service rate of tape drives (includes robot tape mount and file seek times)
T total number of available tape drives in the tape server
t_i number of tape drives dedicated to interactive get requests
t_m number of tape drives dedicated to the migration queue ($T = t_i + t_m$)

Table 2
Results of experiments with the three queues problem

Prob. 2			Power		Jacobi		GS		SOR		
C_1	C_2	C_3	*it*	*time*	*it*	*time*	*it*	*time*	w_*	*it*	*time*
5	5	10	696	82	450	66	164	27	1.6	102	**17**
10	10	10	912	411	590	336	226	154	1.6	142	**98**
10	10	20	1084	1954	726	1658	270	722	1.6	168	**455**
15	15	20	1548	6215	1064	5390	404	2485	1.6	256	**1577**
15	15	30	1664	15,052	1154	13,103	436	6288	1.6	274	**3838**
15	15	50	1874	47,240	1318	41,535	492	21,726	1.6	310	**11,962**
20	20	50	2306	101,680	1642	91,187	618	44,002	1.6	390	**27,123**

Table 3
Results of experiments with the mass storage problem

Prob. 3		Power		GS		SOR			Block GS		Block SOR		
C	N_1	*it*	*time*	*it*	*time*	w_*	*it*	*time*	*it*	*time*	w_*	*it*	*time*
26	6	178	**78**	254	217	1.7	168	**144**	158	140	1.7	98	88
51	11	612	3062	334	3485	1.6	228	2354	156	1673	1.6	106	**1125**
76	16	1146	29,432	428	21,207	1.5	306	14,910	286	14,759	1.7	170	**8876**
101	21	1860	145,162	668	104,229	1.5	454	70,774	470	79,665	1.7	282	**46,708**

n_1 number of requests in the interactive tape queue (including any request(s) currently being served) $(0 \leqslant n_1 \leqslant N_1 - 1)$
T_1 threshold of requests at the interactive tape queue above which one tape drive from the migration tape queue is borrowed
n_2 number of requests in the migration tape queue (including any request(s) currently being served) $(0 \leqslant n_2 \leqslant N_2 - 1)$
n_3 number of put requests written to the disk cache which have not been migrated to the tape library yet $(0 \leqslant n_3 \leqslant N_3 - 1)$
$C - 1$ maximum capacity of the disk cache
H high water-mark for the disk cache used to activate the purging workload
L low water-mark used to terminate the purging workload
$\hat{C}$ current occupancy level of the disk cache $(\lceil L(C-1) \rceil \leqslant \hat{C} \leqslant \lceil H(C-1) \rceil)$
M inter-migration time
R number of stages in the Erlangian approximation of the periodic migration workload $(R \geqslant 5)$
γ rate of the Erlangian approximation of the periodic migration workload $(\gamma = 1/M)$

The system is modeled using five automata $\mathscr{A}^{(\hat{C})}$, $\mathscr{A}^{(n_1)}$, $\mathscr{A}^{(n_2)}$, $\mathscr{A}^{(n_3)}$, and $\mathscr{A}^{(\mathrm{erl})}$ of order respectively $\lceil (H-L)(C-1) \rceil + 1$, N_1, N_2, N_3, and R giving

$$n = (\lceil (H-L)(C-1) \rceil + 1) N_1 N_2 N_3 R.$$

The mass storage model has both synchronizing events and functional rates; it does not have any cyclic dependencies.

We used $\lambda_g = \lambda_p = 1.5$, $\mu = 0.61$, $h_g = h_p = 0.3$, $t_i = t_m = 2$, $L = 0.75$, $H = 0.95$, $M = 40$, $R = 5$ (see [11, p. 5] for details). The automata are ordered as $\mathscr{A}^{(n_4)}, \mathscr{A}^{(n_1)}, \mathscr{A}^{(\mathrm{erl})}, \mathscr{A}^{(\hat{C})}, \mathscr{A}^{(n_2)}$. In Table 3, we provide results for both block and point methods. Forward SOR gives the best results; its block version decreases both the iteration counts and the solution times. We exclude the results of the Jacobi methods since they do not perform well in this problem.

Interestingly, an alternative ordering, namely $\mathscr{A}^{(n_4)}, \mathscr{A}^{(\hat{C})}, \mathscr{A}^{(\mathrm{erl})}, \mathscr{A}^{(n_1)}, \mathscr{A}^{(n_2)}$ gives better results for both block GS and block SOR as shown in Table 4. Note that it is possible to solve the largest system in less than two hours.

Table 5 shows the results of experiments with the sparse matrix approach. As stated before, we experimented with global generators that fit in core memory. In the table, *nz* denotes the number of nonzeros either in the descriptor approach (Desc.) or the sparse matrix approach (Sparse), and *gtime* denotes the global matrix generation time in sparse format. We wanted to compare descriptor and sparse matrix approaches with their best solvers. However, it was not possible to solve the largest three instances of the resource sharing problem, the largest two instances of the three queues problem, and the largest instance

Table 4
Results of other experiments with the mass storage problem

Prob. 3		Block GS		Block SOR		
C	N_i	*it*	*time*	w_*	*it*	*time*
26	6	44	**41**	1.0	44	**41**
51	11	34	**370**	1.0	34	**370**
76	16	32	**1715**	1.0	32	**1715**
101	21	40	6797	1.1	36	**6115**

Table 5
Results of sparse matrix solvers for all problems.

Prob. 1			Desc.	Sparse	Backward SOR			
N	P	n	nz	nz	w_*	*it*	*gtime*	*time*
12	1	4096	48	28,684	1.0	2	1	0
12	6	4096	48	40,960	1.3	18	1	1
12	10	4096	48	53,236	1.3	18	1	1
16	1	65,536	64	589,840	1.0	2	26	2
16	8	65,536	64	851,968	1.3	22	26	22
16	15	65,536	64	1,114,096	1.4	22	27	29
20	1	1,048,576	80	11,534,356	–	–	870	–
20	10	1,048,576	80	16,777,216	–	–	889	–
20	19	1,048,576	80	22,020,076	–	–	882	–
Prob. 2								
C_1	C_2	C_3						
5	5	10	2500	105	11,875	1.6	102	0	2
10	10	10	10,000	145	50,960	1.6	142	1	9
10	10	20	40,000	225	205,000	1.6	168	6	44
15	15	20	90,000	265	471,605	1.6	256	13	153
15	15	30	202,500	345	1,063,125	1.6	274	30	373
15	15	50	562,500	505	2,957,025	–	–	84	–
20	20	50	1,000,000	545	5,315,100	–	–	147	–
Prob. 3					Forward Block SOR			
C	N_i							
26	6	6480	95	39,960	1.0	44	1	5
51	11	73,205	200	479,160	1.0	34	14	50
76	16	327,680	330	2,191,360	1.0	32	86	255
101	21	972,405	485	6,575,310	–	–	331	–

of the mass storage problem using the sparse matrix approach. For the first two problems, each block method ended up being slower than its point version in the sparse approach; hence we give the results of point SOR. For the third problem, we use the alternative ordering $\mathscr{A}^{(n_4)}, \mathscr{A}^{(\tilde{C})}, \mathscr{A}^{(\mathrm{erl})}, \mathscr{A}^{(n_1)}, \mathscr{A}^{(n_2)}$ when generating the matrix in sparse format, hence the faster block SOR solver.

A final remark is that, for a given problem, the optimal parameter of (block) SOR and therefore the number of iterations taken to convergence in the descriptor approach may be (significantly) different than those of the global generator in sparse format. This is something we observed in the mass storage problem for the ordering $\mathscr{A}^{(n_4)}, \mathscr{A}^{(n_1)}, \mathscr{A}^{(\mathrm{erl})}, \mathscr{A}^{(\tilde{C})}, \mathscr{A}^{(n_2)}$. For instance, $w_* = 1.1$, $it = 144$, $time = 219$ for block SOR in sparse format for the given ordering when $C = 51, N_i = 11$. The cause seems to be rounding errors incurred in generating and storing the global matrix.

6. Conclusion

In this work, we presented iterative methods based on splitting a SAN descriptor. Block versions of the same methods follow directly from considering blocks of order n_N, the order of the last automaton, in the given ordering. Larger blocks may be considered by grouping several automata at the end of the given ordering and terminating recursive calls of the lower triangular backward solution algorithm when the first automata in the group is encountered.

An important and frequently overlooked drawback of Markov chain solvers (including SAN solvers) that attempt at computing each and every stationary probability is the memory consumed by double precision temporary storage allocated to the current approximation, possibly the preceding one, and other work arrays. A vector of one million elements requires 8 MB of memory. Although not as large as the memory taken up by double precision nonzeros in the sparse matrix approach, these vectors may end up taking substantial space in iterative methods.

On a desktop workstation with 32 MB of RAM, one can compute the stationary distribution of a SAN descriptor with one million states in core on the order of hours using block SOR. On the other hand, the largest system that can be solved by the sparse matrix approach may be limited to less than one tenth of that could be solved using SANs if the generator is reasonably dense (as in the resource sharing problem: it takes roughly 176 MB to store the generator matrix in sparse format for the most difficult case). We believe the SAN modeling methodology has its merits and drawbacks. It is likely to gain popularity as a viable modeling and analysis tool as faster solvers become available.

Acknowledgments

This work was initiated while the second author was visiting INRIA–LMC–IMAG in June 1996. The authors thank Brigitte Plateau and Paulo Fernandes for supplying a recent version of the PEPS package. They also thank Paulo Fernandes, Brigitte Plateau, and Billy Stewart for the stimulating discussions on SANs. Suggestions of the referees have led to an improved manuscript whom the authors thank for their constructive reports.

Appendix A. Proof of Theorem 2.1

The proof of Theorem 2.1 follows from the lemmas below.

Lemma A.1. *The tensor product of two diagonal matrices D_1 and D_2 is a diagonal matrix D $(= D_1 \otimes D_2)$.*

Proof. By the definition of the $\otimes$ operator, D is a block diagonal matrix where each block is equal to D_2, and since D_2 is a diagonal matrix, D is also diagonal. □

We state Lemmas A.2 and A.3 without proof since they follow from exactly the same line of reasoning as Lemma A.1.

Lemma A.2. *The tensor product of a diagonal matrix D_1 and a strictly lower triangular matrix L_1 is a strictly lower triangular matrix $L(= D_1 \otimes L_1)$.*

Lemma A.3. *The tensor product of a diagonal matrix D_1 and a strictly upper triangular matrix U_1 is a strictly upper triangular matrix $U(= D_1 \otimes U_1)$.*

Lemma A.4. *The tensor product of a strictly lower triangular matrix L_1 and a matrix A_1 of arbitrary nonzero structure is a strictly lower triangular matrix $L(= L_1 \otimes A_1)$.*

Proof. By the definition of the $\otimes$ operator, L is a block strictly lower triangular matrix with zero blocks of the order of A_1 in the diagonal and upper triangular parts. Thus L has zero elements in the diagonal and upper triangular parts; it is strictly lower triangular. □

The proof of the next lemma is similar to that of Lemma A.4.

Lemma A.5. *The tensor product of a strictly upper triangular matrix U_1 and a matrix A_1 of arbitrary nonzero structure is a strictly upper triangular matrix $U(= U_1 \otimes A_1)$.*

Lemma A.6. *$\bar{Q}_e$ is a diagonal matrix.*

Proof. Since $\bar{Q}_e = \sum_{e=1}^{E} \bigotimes_{i=1}^{N} \bar{Q}_e^{(i)}$ and each $\bar{Q}_e^{(i)}$ is diagonal. Then from Lemma A.1, $\bar{Q}_e$ is diagonal. □

Lemma A.7. *Q_ℓ can be split as $D_\ell - L_\ell - U_\ell$, where D_ℓ is diagonal, L_ℓ is strictly lower triangular, U_ℓ is strictly upper triangular and each of the three terms is in the form of a sum of tensor products.*

Proof. Let $Q_\ell^{(i)}$ be split as $D_\ell^{(i)} - L_\ell^{(i)} - U_\ell^{(i)}$, where $D_\ell^{(i)}$ is diagonal, $L_\ell^{(i)}$ is strictly lower triangular, and $U_\ell^{(i)}$ is strictly upper triangular. We use $I_{n_i:n_j}$, to represent an identity matrix of size $\prod_{k=i}^{j} n_k$ when $i \leqslant j$, else a one. Then

$$
\begin{aligned}
Q_\ell &= \bigoplus_{i=1}^{N} Q_\ell^{(i)} \\
&= \sum_{i=1}^{N} I_{n_1} \otimes I_{n_2} \otimes \cdots \otimes Q_\ell^{(i)} \otimes \cdots \otimes I_{n_{N-1}} \otimes I_{n_N} \\
&= \sum_{i=1}^{N} I_{n_1:n_{i-1}} \otimes Q_\ell^{(i)} \otimes I_{n_{i+1}:n_N} \\
&= \sum_{i=1}^{N} I_{n_1:n_{i-1}} \otimes \left(D_\ell^{(i)} - L_\ell^{(i)} - U_\ell^{(i)} \right) \otimes I_{n_{i+1}:n_N} \\
&= \sum_{i=1}^{N} \left(I_{n_1:n_{i-1}} \otimes D_\ell^{(i)} \otimes I_{n_{i+1}:n_N} \right) - \sum_{i=1}^{N} \left(I_{n_1:n_{i-1}} \otimes L_\ell^{(i)} \otimes I_{n_{i+1}:n_N} \right) - \sum_{i=1}^{N} \left(I_{n_1:n_{i-1}} \otimes U_\ell^{(i)} \otimes I_{n_{i+1}:n_N} \right) \\
&= D_\ell - L_\ell - U_\ell . \qquad \square
\end{aligned}
$$

The last equality is a consequence of Lemmas A.1–A.5.

Lemma A.8. *Q_e can be split as $D_e - L_e - U_e$ where D_e is diagonal, L_e is strictly lower triangular, U_e is strictly upper triangular and each of the three terms are in the form of a sum of tensor products.*

Proof. Let $Q_e^{(i)}$ be split as $D_e^{(i)} - L_e^{(i)} - U_e^{(i)}$, where $D_e^{(i)}$ is diagonal, $L_e^{(i)}$ is strictly lower triangular, and $U_e^{(i)}$ is strictly upper triangular. Then

$$
\begin{aligned}
Q_e &= \sum_{e=1}^{E} \bigotimes_{i=1}^{N} Q_e^{(i)} \\
&= \sum_{e=1}^{E} \left(D_e^{(1)} - L_e^{(1)} - U_e^{(1)}\right) \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right) \\
&= \sum_{e=1}^{E} \left[\left(D_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right) - \left(L_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right) - \left(U_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right)\right] \\
&\quad \sum_{e=1}^{E} \left(D_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right) - \sum_{e=1}^{E} \left(L_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right) - \sum_{e=1}^{E} \left(U_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right) \\
&= \sum_{e=1}^{E} \left(D_e^{(1)} \otimes (D_e^{(2)} - L_e^{(2)} - U_e^{(2)}) \otimes \left(\bigotimes_{i=3}^{N} Q_e^{(i)}\right)\right) - \sum_{e=1}^{E} \left(L_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right) \\
&\quad - \sum_{e=1}^{E} \left(U_e^{(1)} \otimes \left(\bigotimes_{i=3}^{N} Q_e^{(i)}\right)\right) \\
&= \sum_{e=1}^{E} \left(D_e^{(1)} \otimes D_e^{(2)} \otimes \left(\bigotimes_{i=3}^{N} Q_e^{(i)}\right)\right) - \sum_{e=1}^{E} \left(D_e^{(1)} \otimes L_e^{(2)} \otimes \left(\bigotimes_{i=3}^{N} Q_e^{(i)}\right)\right) \\
&\quad - \sum_{e=1}^{E} \left(D_e^{(1)} \otimes U_e^{(2)} \otimes \left(\bigotimes_{i=3}^{N} Q_e^{(i)}\right)\right) - \sum_{e=1}^{E} \left(L_e^{(1)} \otimes \left(\bigotimes_{i=2}^{N} Q_e^{(i)}\right)\right) - \sum_{e=1}^{E} \left(U_e^{(1)} \otimes \left(\bigotimes_{i=3}^{N} Q_e^{(i)}\right)\right) \\
&= \cdots \\
&= \sum_{e=1}^{E} \left(\bigotimes_{i=1}^{N} D_e^{(i)}\right) - \sum_{e=1}^{E} \sum_{k=1}^{N} \left(\left(\bigotimes_{i=1}^{k-1} D_e^{(i)}\right) \otimes L_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)}\right)\right) \\
&\quad - \sum_{e=1}^{E} \sum_{k=1}^{N} \left(\left(\bigotimes_{i=1}^{k-1} D_e^{(i)}\right) \otimes U_e^{(k)} \otimes \left(\bigotimes_{i=k+1}^{N} Q_e^{(i)}\right)\right) \\
&= D_e - L_e - U_e.
\end{aligned}
$$

The last equality is a consequence of Lemmas A.1–A.5. □

Theorem 2.1 follows from Lemmas A.6–A.8.

Appendix B. An upper bound on SolveD-L

In this section we provide an upper bound on the number of multiplications performed in the SolveD-L algorithm for point GS (see Section 3.2.1). Remember that multiplying the approximate subvector $\bar{\pi}_j$ with block (j,i) $j > i$ of the descriptor at the first level partitioning can be expressed as $l^{(1)}_{e(j,i)}\bar{\pi}_j(\bigotimes_{k=2}^{N} Q_e^{(k)})$. If the row index j in this expression changes, the product $\bar{\pi}_j\left(\bigotimes_{k=2}^{N} Q_e^{(k)}\right)$ should be reevaluated for each value of j in case there are functional dependencies among automata. At worst, the value of the functional rate remains constant for all blocks in the same row. We use the efficient vector–(generalized) tensor product multiplication algorithm that has a time complexity of $\mathcal{O}\left(\prod_{i=1}^{N} n_i \sum_{i=1}^{N} n_i\right)$ for a tensor product with N matrices each of order n_i. This complexity result assumes that all matrices that participate in the multiplication are dense.

In the following, T_i represents the number of multiplications performed in SolveD-L when the matrix to be solved is partitioned into n_i blocks each of order $\prod_{j=i+1}^{N} n_j$.

$$T_i = E\left((n_i - 1)\prod_{j=i+1}^{N} n_j \sum_{j=i+1}^{N} n_j + i\frac{n_i^2 - n_i}{2}\prod_{j=i+1}^{N} n_j\right) + \frac{n_i^2 - n_i}{2}\prod_{j=i+1}^{N} n_j + n_i T_{i+1} \quad \text{for } i < N,$$

$$T_N = E\left(N\frac{n_N(n_N-1)}{2}\right) + \frac{n_N(n_N-1)}{2} + ENn_N + n_N.$$

The initial call to SolveD-L views the global matrix as partitioned into n_1 blocks each of order $\prod_{i=2}^{N} n_i$. We aim at bounding T_1 given by

$$T_1 = E\left((n_1 - 1)\prod_{i=2}^{N} n_i \sum_{i=2}^{N} n_i + \frac{n_1^2 - n_1}{2}\prod_{i=2}^{N} n_i\right) + \frac{n_1^2 - n_1}{2}\prod_{i=2}^{N} n_i + n_1 T_2.$$

The last term $n_1 T_2$ of T_1 means that in the next call we solve n_1 diagonal blocks of order $\prod_{i=2}^{N} n_i$ recursively. The term that is inside the E parentheses arises from the multiplication of the current approximate subvector with tensor products corresponding to E synchronizing events. The first term $(n_1 - 1)\prod_{i=2}^{N} n_i \sum_{i=2}^{N} n_i$ inside the parentheses is for the multiplication of the current approximate subvector with all blocks below the diagonal due to a synchronizing event. Remember that for each row of blocks all such multiplications are the same (hence we have $n_1 - 1$ of them), however each of the blocks below the diagonal gets multiplied with a different scalar giving the second term $(n_1^2 - n_1)/2\prod_{i=2}^{N} n_i$ inside the parentheses. In the first level of partitioning, $(n_1^2 - n_1)/2$ is simply the number of blocks below the diagonal and $\prod_{i=2}^{N} n_i$ is the length of the subvector. The second term of T_1 is for the number of scalar multiplications performed in computing the current approximate subvector–tensor product multiplication due to local automata. Note that the actual vector–tensor product multiplications are accounted for as the first term inside the E parentheses.

In T_N, we have the number of scalar multiplications due to synchronizing events and due to local automata as the first and the second terms, respectively. The third term is for the number of multiplications performed in computing the diagonal corrector elements (i.e., each of the n_N diagonal elements in a block gets multiplied with the diagonal elements of the previous $N - 1$ levels and this happens for all E synchronizing events), and the last term is for the number of divisions made at level N to obtain the solution.

In order to find a closed form, we write

$$
\begin{aligned}
T_1 &= E\left((n_1-1)\prod_{i=2}^{N} n_i \sum_{i=2}^{N} n_i + \frac{n_1^2-n_1}{2}\prod_{i=2}^{N} n_i\right) + \frac{n_1^2-n_1}{2}\prod_{i=2}^{N} n_i \\
&\quad + n_1\left[E\left((n_2-1)\prod_{i=3}^{N} n_i \sum_{i=3}^{N} n_i + 2\frac{n_2^2-n_2}{2}\prod_{i=3}^{N} n_i\right) + \frac{n_2^2-n_2}{2}\prod_{i=3}^{N} n_i + n_2 T_3\right] \\
&< E\left(\prod_{i=1}^{N} n_i \sum_{i=2}^{N} n_i + \frac{n_1}{2}\prod_{i=1}^{N} n_i\right) + \frac{n_1}{2}\prod_{i=1}^{N} n_i + E\left(\prod_{i=1}^{N} n_i \sum_{i=3}^{N} n_i + n_2\prod_{i=1}^{N} n_i\right) + \frac{n_2}{2}\prod_{i=1}^{N} n_i + n_1 n_2 T_3 \\
&< \prod_{i=1}^{N} n_i\left[E\left(\sum_{i=2}^{N} n_i + \frac{n_1}{2}\right) + E\left(\sum_{i=3}^{N} n_i + n_2\right) + \frac{n_1}{2} + \frac{n_2}{2}\right] + n_1 n_2 T_3 \\
&< E\left(\sum_{i=2}^{N} n_i + \sum_{i=3}^{N} n_i\right)\prod_{i=1}^{N} n_i + \left(\frac{E}{2}(n_1+2n_2) + \frac{n_1+n_2}{2}\right)\prod_{i=1}^{N} n_i + n_1 n_2 T_3 \\
&< \cdots \\
&< E\left(\sum_{i=2}^{N} n_i + \cdots + \sum_{i=N}^{N} n_i\right)\prod_{i=1}^{N} n_i + \left(\frac{E}{2}\sum_{i=1}^{N-1} i n_i\right)\prod_{i=1}^{N} n_i + \frac{1}{2}\sum_{i=1}^{N-1} n_i \prod_{i=1}^{N} n_i + \prod_{i=1}^{N-1} n_i T_N .
\end{aligned}
$$

Noting that

$$
\prod_{i=1}^{N-1} n_i T_N = \frac{1}{2} EN n_N \prod_{i=1}^{N} n_i + \frac{1}{2} EN \prod_{i=1}^{N} n_i + \frac{1}{2} n_N \prod_{i=1}^{N} n_i + \frac{1}{2}\prod_{i=1}^{N} n_i,
$$

we get the (loose) bound

$$
T_1 < \frac{3}{2} EN \sum_{i=1}^{N} n_i \prod_{i=1}^{N} n_i + \frac{1}{2}\sum_{i=1}^{N} n_i \prod_{i=1}^{N} n_i .
$$

Similarly one can find an upper bound on the number of multiplications performed in computing the right hand side b as $(EN+1)\sum_{i=1}^{N} n_i \prod_{i=1}^{N} n_i$. Here, EN is due to synchronizing events and 1 is due to local automata. Each tensor product arising from local automata has one upper triangular matrix; all others are identity. It is not surprising to find the total number of multiplications performed in one iteration of the GS method on a SAN descriptor for the algorithm given in this paper to be $\mathcal{O}\left(EN \sum_{i=1}^{N} n_i \prod_{i=1}^{N} n_i\right)$.

References

[1] B. Plateau, On the stochastic structure of parallelism and synchronization models for distributed algorithms, Proceedings of the SIGMETRICS Conference on Measurement and Modelling of Computer Systems, Austin, TX, August 1985, 147–154.

[2] B. Plateau, J.M. Fourneau, K.-H. Lee, PEPS: A package for solving complex Markov models of parallel systems, in: R. Puigjaner, D. Potier (Eds.), Modeling Techniques and Tools for Computer Performance Evaluation, Spain, September 1988, 291–305.

[3] B. Plateau, J.M. Fourneau, A methodology for solving Markov models of parallel systems, Journal of Parallel and Distributed Computing 12 (1991) 370–387.

[4] B. Plateau, K. Atif, Stochastic automata network for modeling parallel systems, IEEE Transactions on Software Engineering 17/10 (1991) 1093–1108.

[5] W.J. Stewart, Introduction to the Numerical Solutions of Markov Chains, Princeton University Press, Princeton, NJ, 1994.

[6] W.J. Stewart, K. Atif, B. Plateau, The numerical solution of stochastic automata networks, European Journal of Operational Research 86 (1995) 503–525.

[7] P. Fernandes, B. Plateau, W.J. Stewart, Numerical issues for stochastic automata networks, PAPM 96, Fourth Process Algebras and Performance Modelling Workshop, Torino, Italy, July 1996.
[8] P. Fernandes, B. Plateau, W.J. Stewart, Efficient descriptor–vector multiplications in stochastic automata networks, INRIA Report #2935 (Anonymous ftp ftp.inria.fr/INRIA/Publication/RR).
[9] M. Davio, Kronecker products and shuffle algebra, IEEE Transactions on Computers C–30/2 (1981) 116–125.
[10] M. Benzi, T. Dayar, The arithmetic mean method for finding the stationary vector of Markov chains, Parallel Algorithms and Applications 6 (1995) 25–37.
[11] T. Dayar, O.I. Pentakalos, A.B. Stephens, Analytical modeling of robotic tape libraries using stochastic automata, Technical Report TR-97-198, CESDIS, NASA/GSFC, 1997.
[12] L.M. Malhis, W.H. Sanders, An efficient two–stage iterative method for the steady-state analysis of Markov regenerative stochastic Petri net models, Performance Evaluation 27, 28 (1996) 583–601.