



HAL
open science

Numerical Issues for Stochastic Automata Networks

Paulo Fernandes, Brigitte Plateau, William J. Stewart

► **To cite this version:**

Paulo Fernandes, Brigitte Plateau, William J. Stewart. Numerical Issues for Stochastic Automata Networks. [Research Report] RR-2938, INRIA. 1996. inria-00073761

HAL Id: inria-00073761

<https://inria.hal.science/inria-00073761>

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

*Numerical Issues for Stochastic Automata
Networks.*

Paulo Fernandes, Brigitte Plateau and William J. Stewart

N° 2938

16 juillet 1996

_____ THÈME 1 _____



*R*apport
de recherche



Numerical Issues for Stochastic Automata Networks.

Paulo Fernandes*, Brigitte Plateau† and William J. Stewart‡

Thème 1 — Réseaux et systèmes
Projet APACHE

Rapport de recherche n° 2938 — 16 juillet 1996 — 24 pages

Abstract: In this paper we consider some numerical issues in computing solutions to networks of stochastic automata (SAN). In particular our concern is with keeping the amount of computation per iteration to a minimum, since iterative methods appear to be the most effective in determining numerical solutions. In a previous paper we presented complexity results concerning the vector-descriptor multiplication phase of the analysis. In this paper our concern is with implementation details. We experiment with the size and sparsity of individual automata; with the ordering of the automata; with the percentage and location of functional elements; with the occurrence of different types of synchronizing events and with the occurrence of cyclic dependencies within terms of the descriptor. We also consider the possible benefits of grouping many small automata in a SAN with many small automata to create an equivalent SAN having a smaller number of larger automata.

Key-words: Markov chains, Stochastic automata networks, Vector-descriptor multiplications, Grouping of automata, Tensor algebra.

(Résumé : *tsvp*)

* IMAG-LMC, 100 rue des Mathématiques, 38041 Grenoble cedex, France. Research supported by the (CNRS – INRIA – INPG – UJF) joint project *Apache*, and CAPES-COFECUB Agreement (Project 140/93).

† IMAG-LMC, 100 rue des Mathématiques, 38041 Grenoble cedex, France. Research supported by the (CNRS – INRIA – INPG – UJF) joint project *Apache*.

‡ Department of Computer Science, N. Carolina State University, Raleigh, N.C. 27695-8206, USA. Research supported in part by NSF (DDM-8906248 and CCR-9413309).

Optimisations numériques pour les Réseaux d'Automates Stochastiques.

Résumé : Dans cet article, nous étudions quelques optimisations numériques pour les réseaux d'automates stochastiques. En particulier, nous cherchons à diminuer le coût d'une itération, sachant que les méthodes itératives apparaissent comme les plus performantes pour déterminer numériquement la solution. Dans un article précédent, nous avons présenté des résultats de complexité sur l'opération de multiplication vecteur-descripteur. Dans cet article, on s'intéresse à l'implantation de l'algorithme. On cherche à identifier l'influence, sur les performances de l'algorithme, de la taille et la densité des divers automates, du pourcentage et de la position des éléments fonctionnels, de l'occurrence de divers types d'événements synchronisants et de l'occurrence de dépendances cycliques dans les termes du descripteur. On montre aussi les bénéfices qui peuvent être obtenus en groupant les petits automates d'un réseau afin d'obtenir un réseau équivalent avec un plus petit nombre d'automates plus gros.

Mots-clé : Chaîne de Markov, Réseaux d'automates stochastiques, multiplication vecteur-descripteur, groupement, Algèbre tensorielle.

1 Introduction

The use of *Stochastic Automata Networks* (SANs) is becoming increasingly important in performance modelling issues related to parallel and distributed computer systems. Models that are based on Markov chains allow for considerable complexity, but in practice they often suffer from difficulties that are well-documented. The size of the state space generated may become so large that it effectively prohibits the computation of a solution. This is true whether the Markov chain results from a stochastic Petri net formalism, or from a straightforward Markov chain analyzer.

In many instances, the SAN formalism is an appropriate choice. Parallel and distributed systems are often viewed as collections of components that operate more or less independently, requiring only infrequent interaction such as synchronizing their actions, or operating at different rates depending on the state of parts of the overall system. This is exactly the viewpoint adopted by SANs. The components are modelled as individual stochastic automata that interact with each other. Furthermore, the state space explosion problem associated with Markov chain models is mitigated by the fact that the state transition matrix is not stored, nor even generated. Instead, it is represented by a number of much smaller matrices, one for each of the stochastic automata that constitute the system, and from these all relevant information may be determined without explicitly forming the global matrix. The implication is that a considerable saving in memory is effected by storing the matrix in this fashion. We do not wish to give the impression that we regard SANs as a panacea for all modelling problems, just that there is a niche that it fills among the tools that modellers may use. It is fairly obvious that their memory requirements are minimal; it remains to show that this does not come at the cost of a prohibitive amount of computation time.

Stochastic Automata Networks and the related concept of *Stochastic Process Algebras* have become a hot topic of research in recent years. This research has focused on areas such as the development of languages for specifying SANs and their ilk, [17, 18], and on the development of suitable solution methods that can operate on the transition matrix given as a compact SAN descriptor. The development of languages for specifying stochastic process algebras is mainly concerned with structural properties of the nets (compositionality, equivalence, etc.) and with the mapping of these specifications onto Markov chains for the computation of performance measures [18, 2, 6]. Although a SAN may be viewed as a stochastic process algebra, its original purpose was to provide an efficient and convenient methodology for the study of quantitative rather than structural properties of complex systems, [21]. Nevertheless, computational results such as those presented in this paper can also be applied in the context of stochastic process algebras.

There are two overriding concerns in the application of any Markovian modelling methodology, viz., memory requirements and computation time. Since these are frequently functions of the number of states, a first approach is to develop techniques that minimize the number of states in the model. In SANs, it is possible to make use of symmetries as well as lumping and various superpositioning of the automata to reduce the computational burden, [1, 8, 25]. Furthermore, in [14], structural properties of the Markov chain graph (specifically the occurrence of cycles) are used to compute steady state solutions. We point

out that similar, and even more extensive results have previously been developed in the context of Petri nets and stochastic activity networks [7, 8, 9, 15, 24, 26].

Once the number of states has effectively been fixed, the problem of memory and computation time still must be addressed, for the number of states left may still be large. With SANs, the use of a compact descriptor goes a long way to satisfying the first of these, although with the need to keep a minimum of two vectors of length equal to the global number of states, and considerably more than two for more sophisticated procedures such as the GMRES method, we cannot afford to become complacent about memory requirements. As far as computation time is concerned, since the numerical methods used are iterative, it is important to keep both the number of iterations and the amount of computation per iteration to a minimum. The number of iterations needed to compute the solution to a required accuracy depends on the method chosen. In a previous paper, [29], it was shown how projection methods such as Arnoldi and GMRES could be used to substantially reduce the number of iterations needed when compared with the basic power method. Additionally, some results were also given concerning the development of preconditioning strategies that may be used to speed the iterative process even further, but much work still remains to be done in this particular domain. In this paper we concentrate on procedures that allow us to keep the amount of computation per iteration to a minimum. In a previous paper, [13], we proved a theorem concerning the complexity of a matrix-vector multiplication when the matrix is stored as a compact SAN descriptor, since this step is fundamental to all iterative methods and is usually the most expensive operation in each iteration. Additionally, we provided an algorithm that implements the multiplication procedure. The objective of this paper is to analyze the cost of the implementation of this algorithm and to propose improvements that bring its performance close to those of the more usual sparse methods.

2 The SAN Descriptor and Examples

2.1 The SAN Descriptor

There are basically two ways in which stochastic automata interact:

1. The rate at which a transition may occur in one automaton may be a *function* of the state of other automata. Such transitions are called *functional* transitions.
2. A transition in one automaton may *force* a transition to occur in one or more other automata. We allow for both the possibility of a *master/slave* relationship, in which an action in one automaton (the master) actually occasions a transition in one or more other automata (the slaves), and for the case of a *rendez-vous* in which the presence (or absence) of two or more automata in designated states causes (or prevents) transitions to occur. We refer to such transitions collectively under the name of *synchronizing* transitions. Synchronizing transitions may also be functional.

The elements in the matrix representation of any single stochastic automaton are either constants, i.e., nonnegative real numbers, or functions from the global state space to the

nonnegative reals. Transition rates that depend only on the state of the automaton itself, and not on the state of any other automaton, are to all intents and purposes, constant transition rates. A synchronizing transition may be either functional or constant. In any given automaton, transitions that are not synchronizing transitions are said to be *local* transitions.

As a general rule, it is shown in [20], that stochastic automata networks may always be treated by separating out the local transitions, handling these in the usual fashion by means of a tensor sum and then incorporating the sum of two additional tensor products per synchronizing event. Furthermore, since tensor sums are defined in terms of the (usual) matrix sum (of N terms) of tensor products, the infinitesimal generator of a system containing N stochastic automata with E synchronizing events may be written as

$$Q = \sum_{j=1}^{2E+N} \otimes_{g,i=1}^N Q_j^{(i)}.$$

This formula is referred to as the *descriptor* of the stochastic automata network. The subscript g denotes a generalization of the tensor product concept to matrices with functional entries. We now illustrate these concepts via two examples. These examples will also be used later in our numerical experiments.

2.2 A Model of Resource Sharing

In this model, N distinguishable processes share a certain resource. Each of these processes alternates between a *sleeping* state and a resource *using* state. However, the number of processes that may concurrently use the resource is limited to P where $1 \leq P \leq N$ so that when a process wishing to move from the sleeping state to the resource using state finds P processes already using the resource, that process fails to access the resource and returns to the sleeping state. Notice that when $P = 1$ this model reduces to the usual mutual exclusion problem. When $P = N$ all of the processes are independent. We shall let $\lambda^{(i)}$ be the rate at which process i awakes from the sleeping state wishing to access the resource, and $\mu^{(i)}$, the rate at which this same process releases the resource when it has possession of it.

In our SAN representation, each process is modelled by a two state automaton $\mathcal{A}^{(i)}$, the two states being *sleeping* and *using*. We shall let $s\mathcal{A}^{(i)}$ denote the current state of automaton $\mathcal{A}^{(i)}$. Also, we introduce the function

$$f = \delta \left(\sum_{i=1}^N \delta(s\mathcal{A}^{(i)} = \textit{using}) < P \right),$$

where $\delta(b)$ is an integer function that has the value 1 if the boolean b is true, and the value 0 otherwise. Thus the function f has the value 1 when access is permitted to the resource and has the value 0 otherwise. Figure 1 provides a graphical illustration of this model.

The local transition matrix for automaton $\mathcal{A}^{(i)}$ is

$$Q_i^{(i)} = \begin{pmatrix} -\lambda^{(i)} f & \lambda^{(i)} f \\ \mu^{(i)} & -\mu^{(i)} \end{pmatrix},$$

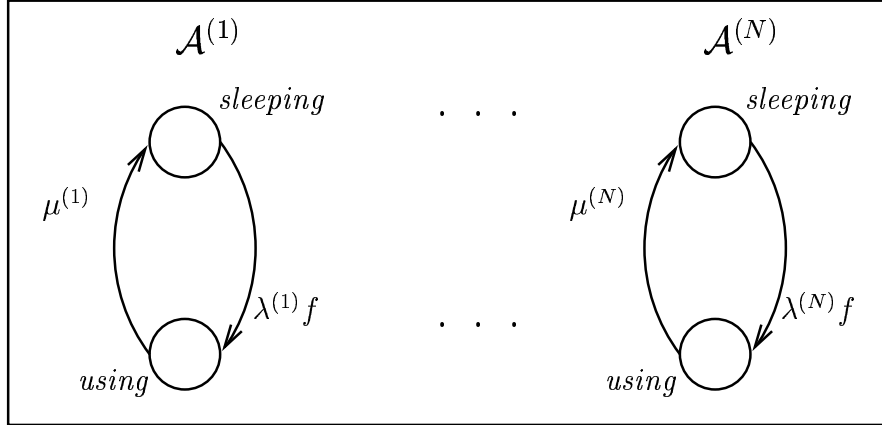


Figure 1: Resource Sharing Model

and the overall descriptor for the model is

$$D = \bigoplus_g \bigoplus_{i=1}^N Q_l^{(i)} = \sum_{i=1}^N I_2 \otimes_g \cdots \otimes_g I_2 \otimes_g Q_l^{(i)} \otimes_g I_2 \otimes_g \cdots \otimes_g I_2,$$

where \otimes_g denotes the generalized tensor operator.

The SAN product state space for this model is of size 2^N . Notice that when $P = 1$, the reachable state space is of size $N + 1$, which is considerably smaller than the product state space, while when $P = N$ the reachable state space is the entire product state space. Other values of P give rise to intermediate cases.

2.3 A Queuing Network with Blocking and Priority Service

The second model we shall use is an open queueing network of three finite capacity queues and two customer classes. Class 1 customers arrive from the exterior to queue 1 according to a Poisson process with rate λ_1 . Arriving customers are lost if they arrive and find the buffer full. Similarly, class 2 customers arrive from outside the network to queue 2, also according to a Poisson process, but this time at rate λ_2 and they also are lost if the buffer at queue 2 is full. The servers at queues 1 and 2 provide exponential service at rates μ_1 and μ_2 respectively. Customers that have been served at either of these queues try to join queue 3. If queue 3 is full, class 1 customers are blocked (blocking after service) and the server at queue 1 must halt. This server cannot begin to serve another customer until a slot becomes available in the buffer of queue 3 and the blocked customer is transferred. On the other hand, when a (class 2) customer has been served at queue 2 and finds the buffer at queue 3 full, that customer is lost. Queue 3 provides exponential service at rate μ_{3_1} to class 1 customers and at rate μ_{3_2} to class 2 customers. It is the only queue to serve both classes.

In this queue, class 1 customers have preemptive priority over class 2 customers. Customers departing after service at queue 3 leave the network. We shall let $C_k - 1$, $k = 1, 2, 3$ denote the finite buffer capacity at queue k .

Queues 1 and 2 can each be represented by a single automaton ($\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ respectively) with a one-to-one correspondance between the number of customers in the queue and the state of the associated automaton. Queue 3 requires two automata for its representation; the first, $\mathcal{A}^{(3_1)}$, provides the number of class 1 customers and the second, $\mathcal{A}^{(3_2)}$, the number of class 2 customers present in queue 3. Figure 2 illustrates this model.

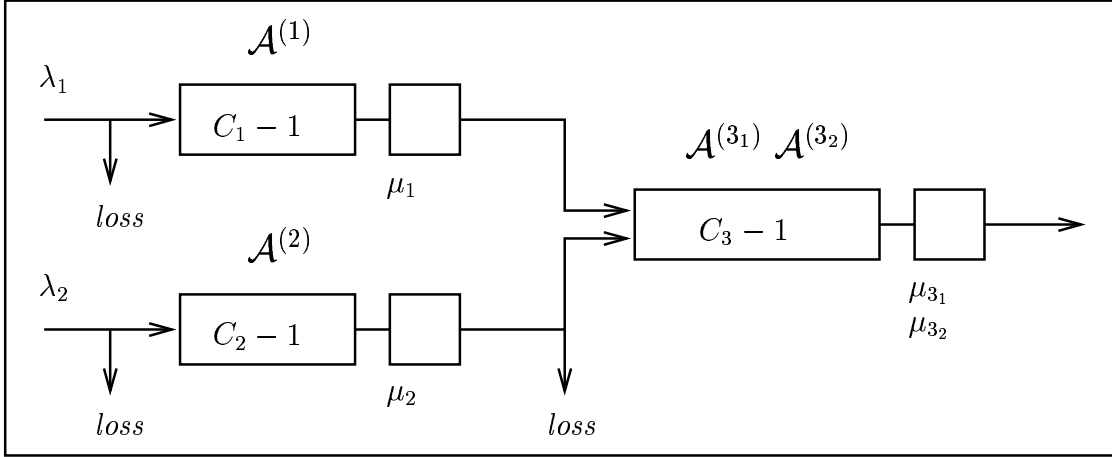


Figure 2: Network of Queues Model

This SAN has two synchronizing events: the first corresponds to the transfer of a class 1 customer from queue 1 to queue 3 and the second, the transfer of a class 2 customer from queue 2 to queue 3. These are synchronizing events since a change of state in automaton $\mathcal{A}^{(1)}$ or $\mathcal{A}^{(2)}$ occasioned by the departure of a customer, must be synchronized with a corresponding change in automaton $\mathcal{A}^{(3_1)}$ or $\mathcal{A}^{(3_2)}$, representing the arrival of that customer to queue 3. We shall denote these synchronizing events as s_1 and s_2 respectively. In addition to these synchronizing events, this SAN required two functions. They are:

$$f = \delta(s\mathcal{A}^{(3_1)} + s\mathcal{A}^{(3_2)} < C_3 - 1)$$

$$g = \delta(s\mathcal{A}^{(3_1)} = 0)$$

The function f has the value 0 when queue 3 is full and the value 1 otherwise, while the function g has the value 0 when a class 1 customer is present in queue 3, thereby preventing a class 2 customer in this queue from receiving service. It has the value 1 otherwise.

Since there are two synchronizing events, each automaton will give rise to *five* separate matrices in our representation. For each automaton k , we will have a matrix of local transitions, denoted by $Q_i^{(k)}$; a matrix corresponding to each of the two synchronizing events,

$Q_{s_1}^{(k)}$ and $Q_{s_2}^{(k)}$, and a diagonal corrector matrix for each synchronizing event, $\bar{Q}_{s_1}^{(k)}$ and $\bar{Q}_{s_2}^{(k)}$. In these last two matrices, nonzero elements can appear only along the diagonal; they are defined in such a way as to make $(\otimes_k Q_{s_j}^{(k)}) + (\otimes_k \bar{Q}_{s_j}^{(k)})$, $j = 1, 2$, generator matrices (row sums equal to zero). The five matrices for each of the four automata in this SAN are as follows (where we use I_m to denote the identity matrix of order m).

For $\mathcal{A}^{(1)}$:

$$Q_l^{(1)} = \begin{pmatrix} -\lambda_1 & \lambda_1 & 0 & \cdots & 0 \\ 0 & -\lambda_1 & \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\lambda_1 & \lambda_1 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \quad Q_{s_1}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_1 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_1 & 0 & 0 \\ 0 & \cdots & 0 & \mu_1 & 0 \end{pmatrix},$$

$$\bar{Q}_{s_1}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -\mu_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mu_1 & 0 \\ 0 & \cdots & 0 & 0 & -\mu_1 \end{pmatrix}, \quad Q_{s_2}^{(1)} = I_{C_1} = \bar{Q}_{s_2}^{(1)}.$$

For $\mathcal{A}^{(2)}$:

$$Q_l^{(2)} = \begin{pmatrix} -\lambda_2 & \lambda_2 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & \lambda_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\lambda_2 & \lambda_2 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \quad Q_{s_2}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_2 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_2 & 0 & 0 \\ 0 & \cdots & 0 & \mu_2 & 0 \end{pmatrix},$$

$$\bar{Q}_{s_2}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -\mu_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mu_2 & 0 \\ 0 & \cdots & 0 & 0 & -\mu_2 \end{pmatrix}, \quad Q_{s_1}^{(2)} = I_{C_2} = \bar{Q}_{s_1}^{(2)}.$$

For $\mathcal{A}^{(3_1)}$:

$$Q_l^{(3_1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_{3_1} & -\mu_{3_1} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_{3_1} & -\mu_{3_1} & 0 \\ 0 & \cdots & 0 & \mu_{3_1} & -\mu_{3_1} \end{pmatrix}, \quad Q_{s_1}^{(3_1)} = \begin{pmatrix} 0 & f & 0 & \cdots & 0 \\ 0 & 0 & f & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & f \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix},$$

$$\bar{Q}_{s_1}^{(3_1)} = \begin{pmatrix} f & 0 & 0 & \cdots & 0 \\ 0 & f & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & f & 0 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \quad Q_{s_2}^{(3_1)} = I_{C_3} = \bar{Q}_{s_2}^{(3_1)}.$$

For $\mathcal{A}^{(3_2)}$:

$$Q_l^{(3_2)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_{3_2}g & -\mu_{3_2}g & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_{3_2}g & -\mu_{3_2}g & 0 \\ 0 & \cdots & 0 & \mu_{3_2}g & -\mu_{3_2}g \end{pmatrix}, \quad Q_{s_2}^{(3_2)} = \begin{pmatrix} 1-f & f & 0 & \cdots & 0 \\ 0 & 1-f & f & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1-f & f \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix},$$

$$\bar{Q}_{s_2}^{(3_2)} = I_{C_3} = Q_{s_1}^{(3_2)} = \bar{Q}_{s_1}^{(3_2)}.$$

The overall descriptor for this model is given by

$$D = \bigoplus_g Q_l^{(i)} + \bigotimes_g Q_{s_1}^{(i)} + \bigotimes_g \bar{Q}_{s_1}^{(i)} + \bigotimes_g Q_{s_2}^{(i)} + \bigotimes_g \bar{Q}_{s_2}^{(i)},$$

where the generalized tensor sum and the four generalized tensor products are taken over the index set $\{1, 2, 3_1 \text{ and } 3_2\}$. The reachable state space of the SAN is of size $C_1 \times C_2 \times C_3(C_3 + 1)/2$ whereas the complete SAN product state space has size $C_1 \times C_2 \times C_3^2$. Finally, we would like to draw our readers attention to the sparsity of the matrices presented above.

3 Algorithm Analysis

3.1 The Complexity Result and Algorithm

We present without proof, the theorem concerning vector-descriptor multiplication and its accompanying algorithm, [13]. We use the notation $B[\mathcal{A}]$ to indicate that the matrix B may contain transitions that are a function of the state of the automaton \mathcal{A} , and more generally, $A^{(m)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \dots, \mathcal{A}^{(m-1)}]$ to denote that the matrix $A^{(m)}$ may contain elements that are a function of the state variable of one or more of the automata $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \dots, \mathcal{A}^{(m-1)}$.

Theorem 3.1 *The multiplication*

$$x \times \left(A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}] \otimes_g \cdots \otimes_g A^{(N)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(N-1)}] \right)$$

where x is a real vector of length $\prod_{i=1}^N n_i$ may be computed in $O(\rho_N)$ multiplications, where

$$\rho_N = n_N \times \left(\rho_{N-1} + \prod_{i=1}^N n_i \right) = \prod_{i=1}^N n_i \times \sum_{i=1}^N n_i,$$

by the algorithm described below.

Algorithm: Vector Multiplication with a Generalized Tensor Product

$$\begin{aligned}
& x \left(A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}] \otimes_g \dots \otimes_g A^{(N)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(N-1)}] \right) = \\
& \quad x \left(\begin{aligned}
& I_{1:N-1} \otimes_g A^{(N)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(N-1)}] \\
& \times I_{1:N-2} \otimes_g A^{(N-1)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(N-2)}] \otimes_g I_{N:N} \\
& \times \dots \\
& \times I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g I_{3:N} \\
& \times A^{(1)} \otimes_g I_{2:N} \end{aligned} \right) \tag{1}
\end{aligned}$$

2.	Initialize: $nleft = n_1 n_2 \dots n_{N-1}$; $nright = 1$.
2.	For $i = N, \dots, 2, 1$ do
2.	• $base = 0$; $jump = n_i \times nright$;
2.	• For $k = 1, 2, \dots, nleft$ do
3.	◦ For $j = 1, 2, \dots, i - 1$ do
3.	* $k_j = \left(\left[(k - 1) / \prod_{l=j+1}^{i-1} n_l \right] \bmod \left(\prod_{l=j}^{i-1} n_l \right) \right) + 1$
2.	◦ For $j = 1, 2, \dots, nright$ do
2.	* $index = base + j$;
2.	* For $l = 1, 2, \dots, n_i$ do
2.	• $z_l = x_{index}$; $index = index + nright$;
1.	* Multiply: $z' = z \times A^{(i)}[k_1, \dots, k_{i-1}]$
2.	* $index = base + j$;
2.	* For $l = 1, 2, \dots, n_i$ do
2.	• $x'_{index} = z'_l$; $index = index + nright$;
2.	◦ $base = base + jump$;
2.	• $nleft = nleft / n_{i-i}$;
2.	• $nright = nright \times n_i$;
2.	$x = x'$;

Figure 3: SAN Algorithm Parts

It is useful to consider the code as consisting of the three parts illustrated in Figure 3.

- Part 1 corresponds to the vector-matrix multiplication, including evaluation of functions when needed. Here $k_i \in \{1, 2, \dots, n_i\}$ denotes the state of the i^{th} automaton.
- Part 2 corresponds to fetching the sub-vectors to be multiplied and the loop management for computing the various permutations.
- Part 3 occurs only in models with functional transition rates and corresponds to the transformation of a vector index into a SAN state, which is subsequently used as an argument for the functions. It is performed only if the matrix $A^{(i)}$ in the inner loop has functional entries.

Notice that the number of multiplications in the innermost loop of the algorithm may be reduced by taking advantage of the fact that the block matrices are generally sparse. The above complexity result was computed under the assumption that the matrices are full. The cost of the function evaluations is included in the definition of the big Oh formula.

In equation (1), only one automata can depend on the $(N - 1)$ other automata and so on. One automaton must be independent of all the others. This provides a means by which the individual factors on the right-hand side of equation (1) *must* be ranked; i.e., according to the automata on which they *may* depend. A given automaton may actually depend on a subset of the automata in its parameter list.

What is not immediately apparent from the algorithm as described is the fact that in *ordinary tensor products* with no functional terms, the normal factors commute and the order in which the innermost multiplication is carried out may be arbitrary. In *generalized tensor products* with functional terms, however, this freedom of choice does not exist: the order is prescribed. As indicated by the right hand side of equation (1), each automata $\mathcal{A}^{(i)}$ is represented by a normal factor of the form

$$I_{1:i-1} \otimes_g A^{(i)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(i-1)}] \otimes_g I_{i+1:m}$$

which *must* be processed before any factor of its arguments $[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(i-1)}]$ is processed.

Now consider an arbitrary permutation of the factors on the left-hand side of (1); In [13], we show how to perform this simple transformation, which gives us the freedom of ordering the factors of a term to optimize computation cost. We have on the left-hand side,

$$A^{(\sigma_1)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(\sigma_1-1)}] \otimes_g \dots \otimes_g A^{(\sigma_N)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(\sigma_N-1)}].$$

Indeed this is just such a permutation of equation (1). No matter which permutation is chosen, the right-hand side remains essentially identical in the sense that the terms are always ordered from largest to smallest set of *possible* dependencies. The only change results from the manner in which each *normal factor* is written. Each must have the form

$$I_{\sigma_1:\sigma_{i-1}} \otimes_g A^{(\sigma_i)}[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(\sigma_i-1)}] \otimes_g I_{\sigma_{i+1}:\sigma_m},$$

with $I_{\sigma_l:\sigma_k} = \prod_{i=l}^k n_{\sigma_i}$.

We would like to point out that although we may reorganize the terms of the left-hand side of equation (1), in any way we wish, the advantage of leaving them in the form given by equation (1) is precisely that the computation of the state indices (part 3) can be moved outside the innermost j -loop of the algorithm. A different rearrangement would require more extensive index computation *within* the innermost j -loop. Note that the order given by the left-hand side of equation (1) is precisely the opposite of that given by its right-hand side.

Example:

Consider the following term in the descriptor of a SAN with 4 automata, \mathcal{A}_1 , \mathcal{A}_2 , \mathcal{A}_3 and \mathcal{A}_4 . The term is constituted from four matrices as follows: A_3 , (a constant matrix, but \mathcal{A}_3 appears as an argument of A_1 and A_2); $A_1(\mathcal{A}_3)$, (the automaton \mathcal{A}_1 is an argument of A_2); $A_2(\mathcal{A}_1, \mathcal{A}_3)$; and A_4 . The algorithm imposes an order in ranking the factors (A_3 , A_1 , A_2):

$$A_3 \otimes A_1(\mathcal{A}_3) \otimes A_2(\mathcal{A}_1, \mathcal{A}_3) \otimes A_4$$

A_4 may be placed anywhere, but the algorithm will perform better if it is placed in the final position and thus avoid additional computation in part 3; the automaton \mathcal{A}_4 is not an argument of any function. Note that the order of the factor multiplication is

$$\left(I_{1:4} \otimes_g A^{(4)} \right) \times \left(I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}, \mathcal{A}^{(3)}] \otimes_g I_{3:4} \right) \times \left(A^{(1)}[\mathcal{A}^{(3)}] \otimes_g I_{2:4} \right) \times \left(I_{1:2} \otimes_g A^{(3)} \otimes_g I_{4:4} \right)$$

From this first SAN, we can construct a second with an additional automaton, \mathcal{A}_5 and in this term of the descriptor a single matrix $A_5(\mathcal{A}_4)$. The algorithm then requires, on the right-hand side, that A_3 must precede A_1 , that A_4 precedes A_5 , and that A_1 and A_3 precede A_2 , (the arguments before the functions). This provides us with 11 different possible orderings. To improve the running cost of the algorithm, the best choice is the one that minimizes part 3, which means that the size of the matrices should be taken into account, with the smallest coming first and the greatest last. For example, if $n_1 \leq n_2 \leq n_3 \leq n_4 \leq n_5$, the best choice is:

$$A_3 \otimes A_1(\mathcal{A}_3) \otimes A_2(\mathcal{A}_1, \mathcal{A}_3) \otimes A_4 \otimes A_5(\mathcal{A}_4)$$

which takes into account the imposed sub-orderings of 3 before 1, 1 and 3 before 2, and 4 before 5.

It is shown in [13] that if such an order cannot be found, that is to say, if there is a cyclic dependency of function and argument in a term, this term can be exactly decomposed into a number of non-cyclic terms.

So a rule of thumb in ordering a term is;

- Take the set of factors that are functions or arguments within functions. The functions *must* be ranked after their *arguments*. Better performance is obtained when, considering these constraints, they are ranked in non-decreasing size.
- Then take the automata that are neither functional nor arguments and rank them last, in any order.

4 Some Small Examples

All numerical experiments were conducted using the software package PEPS, version 3.0 [23]. This version of PEPS is implemented in C++. Only the non-diagonal elements of the descriptor are handled by this algorithm. The diagonal elements of the descriptor are pre-calculated once and stored in a vector. The vector-matrix multiplication includes a dot product with the diagonal entries. The computing platform was an IBM RS6000/370 workstation running AIX version 3.2.5.

Our first objective is to experiment with the size and sparsity of individual automata; with the ordering of automata; with the percentage and location of functional elements; with the occurrence of different types of synchronizing events and with the occurrence of cyclic dependencies, (see [13]), within terms of the descriptor. We choose to do so by means of the set of small examples given in table 1.

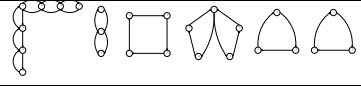
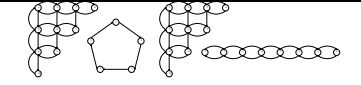
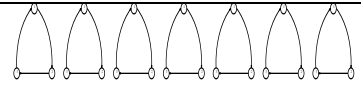
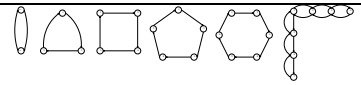
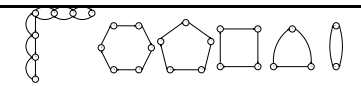
Example	SAN	state space size	nonzero transitions
N1		3780	0.22%
N2		4000	0.20%
N3		2187	0.37%
N4		5040	0.15%
N5		5040	0.15%

Table 1: The Set of Small Examples

Observe that SAN **N1** has sparse and non-sparse local automata. SAN **N2** has larger automata. SAN **N3** has seven small automata. SANs **N4** and **N5** have automata of different sizes and in different order, and so on. In these examples and their variations with functional rates and synchronizing events (see Sections 4.2 and 4.3) the reachable state space is the product state space. However, the sparsity of the global generator is significant and measured as the percentage of nonzero transition rates in the global descriptor. For each model, 1000 iterations of each vector-descriptor multiplications were performed, and the cpu time in seconds is reported here. Each part of the SAN algorithm is evaluated as a percentage of the total cpu time (SAN column).

Example	SAN	Part 1	Part 2	Diag.
N1	63s	20s	38s	5s
	100%	32%	60%	8%
N2	52s	20s	27s	5s
	100%	38%	52%	10%
N3	39s	11s	25s	3s
	100%	28%	64%	8%
N4	84s	24s	53s	7s
	100%	29%	63%	8%
N5	84s	24s	53s	7s
	100%	29%	63%	8%

Table 2: Simple SANs

4.1 Simple SANs

Let us first analyse the part of Table 2 showing the comparative cost of each part of the code, according to the division give in Figure 3. Note that the cost of the dot product with the diagonal is small (less than 10%) and this will be always the case for all experiments. The cost of part 2 is very high, larger than the multiplication cost (part 1). Notice that the greater the number of automata, the larger the overhead cost of the algorithm. This suggests that it might be worthwhile, in SANs with a lot of small automata, to compose subgroups with small automata to build up an equivalent SAN with less, but larger, automata. This tendency has to be moderated by the cost of generating composed automata, as it will be seen later. Finally, there is no apparent difference between **N4** and **N5**, which confirms that in these simple SANs without functions, ordering has no noticeable effect on performance.

4.2 SANs with functional rates

In the next set of experiments, functional rates are added to the SANs. For example, **F2a** is derived from SAN **N2**; it is composed of 4 automata of size 10, 5, 10, 8 as indicated in the first column of Table 3. Functional rates are included in the automaton whose size (8) has a tilde; functions entries are indicated by a circumflex. The percentage in the first column of Table 3 indicates the percentage of nonzero entries that are functional.

Observe the effect of including functional rates on the overhead to compute the functions (part 1) and on the computation of the SAN state (part 3). This is particularly noticeable when comparing **N1** and **F1** in which part 2 is identical. In general, the time spent in part 3 varies considerably (between 4% and 23%) and depends on the location of the automata that have functional rates. This effect is particularly obvious in SANs **F3a**, **F3b** and **F3c**. The reason for such a difference (expressed exclusively in part 3) is that the evaluation of the function arguments is more expensive when the automata is in the last position. What really counts is the product of the sizes of the automata that are ranked before the automata with

Example	Functions	SAN	Part 1	Part 2	Part 3	Diag.
F1	16%	94s	44s	38s	7s	5s
$\hat{7} \times \hat{3} \times \tilde{4} \times \tilde{5} \times 3 \times 3$	100%	100%	47%	41%	7%	5%
F2a	2%	65s	23s	27s	10s	5s
$\hat{10} \times \hat{5} \times 10 \times \tilde{8}$	100%	100%	35%	42%	15%	8%
F2b	2%	70s	23s	27s	15s	5s
$\hat{10} \times \hat{8} \times 10 \times \tilde{5}$	100%	100%	33%	39%	21%	7%
F3a	5%	46s	16s	25s	2s	3s
$\hat{3} \times \tilde{3} \times 3 \times 3 \times 3 \times 3 \times 3$	100%	100%	35%	55%	4%	6%
F3b	5%	50s	16s	25s	6s	3s
$\hat{3} \times 3 \times 3 \times 3 \times 3 \times \tilde{3} \times 3$	100%	100%	32%	50%	12%	6%
F3c	5%	57s	16s	25s	13s	3s
$\hat{3} \times 3 \times 3 \times 3 \times 3 \times 3 \times \tilde{3}$	100%	100%	28%	44%	23%	5%

Table 3: SANs with functional rates

functional rates in the SAN. The larger this product, the more expensive part 3 becomes. It can be seen in **F2a** and **F2b** that better efficiency of the SAN code is obtained when it is the larger automata that has the functional rates, (which implies that the product state of the automata ranked before is smaller). These experiments confirm the previous analysis concerning algorithm cost.

4.3 SANs with synchronizing events and functional transition probabilities

The last set of experiments on these small examples is obtained by adding synchronizing events to the SANs (one to each model, except **S2c** in which two events are added). The automata concerned by the synchronizing event are those whose size is within a box in column 1 of table 4. The percentage of nonzero transitions that are in fact synchronized transitions is also indicated in the first column. When an automata size has a tilde on top of it, this indicates that the corresponding automata has functional transition probabilities in its synchronizing matrix while a circumflex indicates the location of arguments. Some of these functional transition probabilities lead to dependency cycles in the synchronizing terms of the descriptor, and this is indicated in the first column (cyclic). When an automata size is in a bold box, this indicates that the automata is the *cutset* (see [13]) chosen to break the functional dependency cycle.

If we compare, for example, the cpu times for SANs **S2a** and **S2b**, we see that it is not sensitive to the proportion of synchronized transitions. If we compare the cpu time for SANs **S2a** and **S2c** we see that it is sensitive to the number of synchronizing events (each synchronized event adds one term to the descriptor). The position in the SAN of the synchronizing event (without functions) is not relevant to the performance (see **S4a** and **S5a**). The comparison of the time for SAN **S1a**, **S1b** and **S1c**, shows that a cycle decreases

Example	Synch.	SAN	Part 1	Part 2	Part 3	Diag.
S1a	3%	88s	28s	46s	9s	5s
	$\hat{7} \times 3 \times 4 \times \hat{5} \times 3 \times \hat{3}$	100%	32%	52%	10%	6%
S1b	3%(cyclic)	95s	31s	50s	9s	5s
	$\hat{7} \times 3 \times 4 \times \hat{5} \times 3 \times \hat{3}$	100%	33%	53%	9%	5%
S1c	3%(cyclic)	104s	34s	56s	9s	5s
	$\hat{7} \times 3 \times 4 \times \hat{5} \times 3 \times \hat{3}$	100%	33%	54%	9%	5%
S2a	9%	79s	23s	51s	0s	5s
	$10 \times 5 \times 10 \times 8$	100%	29%	65%	0%	6%
S2b	17%	79s	23s	51s	0s	5s
	$10 \times 5 \times 10 \times 8$	100%	29%	65%	0%	6%
S2c	9%	83s	23s	55s	0s	5s
	$10 \times 5 \times 10 \times 8$	100%	28%	66%	0%	6%
S4a	7%	99s	28s	64s	0s	7s
	$2 \times 3 \times 4 \times 5 \times 6 \times 7$	100%	28%	65%	0%	7%
S5a	7%	99s	28s	64s	0s	7s
	$7 \times 6 \times 5 \times 4 \times 3 \times 2$	100%	28%	65%	0%	7%

Table 4: SANs with synchronizing events and functional transition probabilities

the performance (it adds terms in the descriptor), and that the best choice is to choose the smaller cutset.

4.4 Summary

This concludes the experiment on the set of small examples. It gives rules of thumb to order automata in a network to achieve better performance. More precisely, it is not the automata in the SAN that must be ordered, but within each term of the descriptor, the best ordering should be computed independently. It now remains to examine the effect of reducing the number of automata in more detail, for it was observed that this also has a role to play in the efficiency of the algorithm.

5 Grouping of Automata

The objective in this section is to show how we can reduce a SAN to an “equivalent” SAN with less automata. The equivalence notion is with respect to the underlying Markov chain and is defined below. Our approach is based on simple algebraic transformations of the descriptor and not on the automata network. Consider a SAN containing N stochastic automata $\mathcal{A}_1, \dots, \mathcal{A}_N$ of size n_i respectively, E synchronizing events s_1, \dots, s_E , and functional

transition rates. Its descriptor may be written as

$$Q = \sum_{j=1}^{N+2E} \otimes_{g,i=1}^N Q_j^{(i)}$$

Let $1, \dots, N$ be partitioned in B groups named b_1, \dots, b_B , and, without loss of generality, assume that $b_1 = [1, \dots, c_2]$, $b_2 = [c_2 + 1, \dots, c_3]$, etc, for some increasing sequence of c_i , with $c_1 = 0$, $c_{B+1} = N$. The descriptor can be rewritten, using the associativity of the generalized tensor product, as

$$Q = \sum_{j=1}^{2E+N} \otimes_{g,k=1}^B \left(\otimes_{g,j=c_k+1}^{c_{k+1}} Q_j^{(i)} \right).$$

The matrices $R_j^{(k)} = \otimes_{g,j=c_k+1}^{c_{k+1}} Q_j^{(i)}$, for $j \in 1, \dots, 2E+N$, are, by definition, the transition matrices of a grouped automaton, named G_k of size $h_k = \prod_{i=c_k+1}^{c_{k+1}} n_i$. The descriptor may be rewritten

$$Q = \sum_{j=1}^{2E+N} \otimes_{g,k=1}^B R_j^{(k)}.$$

This formulation is the basis of the grouping process. From this first algebraic expression, several important simplifications may be carried out and are explained below. Before proceeding to these simplifications, we need to write the descriptor more precisely, separating out the terms resulting from local transitions from those resulting from synchronizing events (Our notation is similar to that used in Section 2 and can be easily interpreted by analogy):

$$Q = \sum_{j=1}^{N+2E} \otimes_{g,i=1}^N Q_j^{(i)} = \bigoplus_{g,i=1}^N Q_l^{(i)} + \sum_{j=1}^E \left(\bigotimes_{g,i=1}^N Q_{s_j}^{(i)} + \bigotimes_{g,i=1}^N \bar{Q}_{s_j}^{(i)} \right)$$

Grouping by associativity gives

$$= \bigoplus_{g,k=1}^B R_l^{(k)} + \sum_{j=1}^E \left(\bigotimes_{g,k=1}^B R_{s_j}^{(k)} + \bigotimes_{g,k=1}^B \bar{R}_{s_j}^{(k)} \right)$$

with

$$R_l^{(k)} = \bigoplus_{g,i=c_k+1}^{c_{k+1}} Q_l^{(i)}$$

and

$$R_{s_j}^{(k)} = \bigotimes_{g,i=c_k+1}^{c_{k+1}} Q_{s_j}^{(i)}$$

and

$$\bar{R}_{s_j}^{(k)} = \bigotimes_{g,i=c_k+1}^{c_{k+1}} \bar{Q}_{s_j}^{(i)}.$$

First simplification: Removal of synchronizing events.

Assume that one of the synchronizing event, say s_1 , is such that it synchronizes automata within a group, say b_1 . Indeed, this synchronized event becomes internal to group b_1 and may be treated as a transition that is local to G_1 . In this case, the value of $R_l^{(1)}$ may be changed in order to simplify the formula for the descriptor. Using

$$R_l^{(1)} \Leftarrow R_l^{(1)} + R_{s_1}^{(1)} + \bar{R}_{s_1}^{(1)}$$

the descriptor may be rewritten as

$$\bigoplus_{g,k=1}^B R_l^{(k)} + \sum_{j=2}^E \left(\bigotimes_{g,i=1}^B R_{s_j}^{(i)} + \bigotimes_{g,i=1}^B \bar{R}_{s_j}^{(i)} \right).$$

The descriptor is thus reduced (two terms having disappeared). This procedure can be applied to all identical situations.

Second simplification: Removal of functional terms.

Following this same line of thought, assume that the local transition matrix of G_1 is a tensor sum of matrices that are functions of the states of automata in b_1 itself. Then the functions in $Q_l^{(i)}$ of

$$R_l^{(1)} = \bigoplus_{g,i=c_1+1}^{c_2} Q_l^{(i)}$$

are evaluated when performing the generalized tensor operator and $R_l^{(1)}$ is a constant matrix. This is true for all situation of this type, for local matrices and synchronized terms.

However, if $R_{s_j}^{(1)}$ is the tensor product of matrices that are functions of the states of automata, some of which are in b_1 and some of which are not in b_1 , then performing the generalized tensor product $R_{s_j}^{(1)} = \bigotimes_{g,i=c_1+1}^{c_2} Q_{s_j}^{(i)}$ allows us to partially evaluate the functions for the arguments in b_1 . Others arguments cannot be evaluated. These must be evaluated later when performing the computation $\bigotimes_{g,i=1}^B R_{s_j}^{(i)}$ and may in fact, result in an increased number of function evaluations. Some numerical effects of this phenomenon are provided in the next section.

Third simplification: Reduction of the reachable state space.

In the process of grouping, the situation might arise that a grouped automata G_i has a reachable state space smaller than the product state space $[1, \dots, \prod_{i=c_j+1}^{c_j+1} n_i]$. This happens after simplifications one and/or two have been performed. For example, functions may evaluate to zero, or synchronizing events may disable certain transitions. In this case, a reachability analysis is performed in order to compute the reachable state space. This analysis is conducted on the matrix :

$$R_l^{(i)} + \sum_{j=1}^F R_{s_j}^{(i)} + \bar{R}_{s_j}^{(i)}$$

where F is the set of remaining synchronizing events for the SAN G_1, \dots, G_B . In practice, in the SAN methodology, the global reachable state space is known in advance and the reachable state space of a group may be computed with a simple projection.

6 The Numerical Benefits of Grouping

Let us now observe the effectiveness of grouping automata on the two examples discussed in Sections 2.2 and 2.3. In particular, we would like to observe the effect on the time required to perform 10 premultiplications of the descriptor by a vector and on the amount of memory needed to store the descriptor itself. Intuitively we would expect the computation time to diminish and the memory requirements to augment as the number of automata is decreased. The experiments in this section quantify these effects.

We first consider the resource sharing model of Section 2.2 with parameter values $N = 12, 16$ and 20 for both $P = 1$ and $P = N - 1$. The models were grouped in various ways, varying from the original (non-grouped) case in which $B = N$ to a purely sparse matrix approach in which $B = 1$, where B of course, denotes the number of automata that remain after the grouping process. In the examples with a single resource ($P = 1$) we differentiate between two distinct cases; the first when the automata are grouped but the state space in each of the larger automata that result from the grouping is not reduced and the second when the state space of the resulting automata is reduced (by the elimination of non-reachable states from each new block of automata). The latter is indicated by the label (\mathfrak{R}) in the tables. The elimination of non-reachable states only affects the states inside a grouped automaton and not all the states of the global model. This reduction is not possible in models with $N - 1$ resources.

The results presented in Table 5 illustrate the substantial gain in CPU time as the number of automata is reduced, and this with relatively little impact on memory requirements. Furthermore, this is seen to be true even when the state space within the grouped automata are not reduced. A more complete set of results for the reduced case is graphically displayed in Figure 4. However, no results are presented for the two cases $N = 20$; $P = 8, 19$; $B = 1$ since the amount of memory needed exceeded that available on our machine. Estimates are presented by a dotted line. When reading these graphs, be aware that the scale varies with increasing values of N . These graphs display both CPU curves and memory curves. Consider first the memory curves. Two contrasting effects are at work here. First there is the reduction in the reachable state space which entails a subsequent reduction in the size of the probability vectors and hence an overall reduction in the amount of memory needed. On the other hand, the size of the matrices representing the grouped automata is increased thereby increasing the amount of memory needed. This latter effect becomes more important with increasing value of P , and indeed becomes the dominant effect, as may be observed from Figure 4. As for the CPU curves, observe that these always decrease with decreased number of blocks of automata. This is a combined effect of a reduction in the reachable state space, algorithm overhead, and the number of functions that need to be evaluated. Notice also that the reduction in the number of function evaluated only occurs when the number of automata

Models	P=1		P=1 (\mathfrak{R})		P=N-1	
	CPU	Mem	CPU	Mem	CPU	Mem
N=12 B=12	1.12	33	1.12	33	1.12	33
N=12 B=6	0.51	33	0.10	6	0.68	33
N=12 B=4	0.36	33	0.03	2	0.53	34
N=12 B=2	0.25	42	0.00	1	0.45	50
N=12 B=1	0.00	0	0.00	0	0.14	1 087
N=16 B=16	27.48	513	27.48	513	27.48	513
N=16 B=8	13.38	513	1.18	52	16.78	514
N=16 B=4	6.79	516	0.06	5	12.51	518
N=16 B=2	4.63	564	0.00	1	12.19	593
N=16 B=1	0.00	1	0.00	1	3.91	22 527
N=20 B=20	610.59	8 193	610.59	8 193	610.59	8 193
N=20 B=10	289.04	8 194	13.45	462	364.43	8 194
N=20 B=5	141.65	8 197	0.33	25	245.16	8 200
N=20 B=2	59.69	8 440	0.01	2	195.62	8 640
N=20 B=1	0.00	1	0.00	1	—	—

Table 5: Resource Sharing Model

in a group is greater than the number of resources. The combined effects of this and the reduction in algorithm overhead and reachable state space leads to the different slopes that are apparent in the CPU curves. Finally notice that although the gains obtained in models with very few reachable states (those cases in which $P = 1$) are impressive, we must keep in mind the fact that the SAN approach is not a realistic in these cases. It is much better to generate the small number of states using a standard sparse matrix approach.

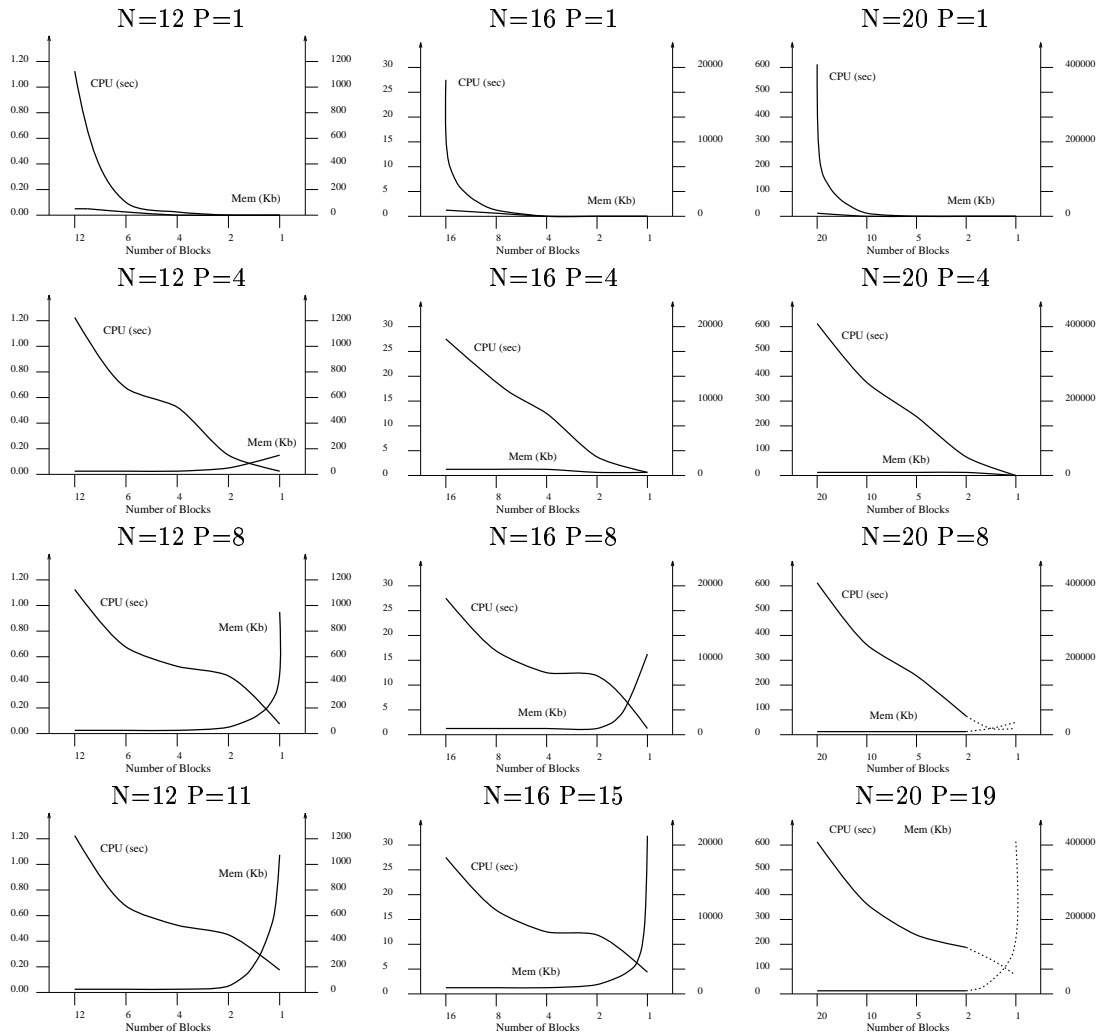


Figure 4. Reduced Models: Computation Time and Memory requirements

It may be argued that the best approach (at least for this particular example) is to combine all the automata into just two groups, for this allows us to avoid the store space explosion problem with just a minimal increase in CPU time over a purely sparse approach.

Let us now turn our attention to the queueing network model presented in the Section 2.3. We analyzed two models with parameters $C_1, C_2 = 5, 10$ and $C_3 = 10, 20, 30$ and 50. With these models, experiments were conducted using two different kinds of grouping:

- A grouping of the automata according to customer class (A_1 and A_{3_1}) and (A_2 and A_{3_2}).
- A grouping of the automata according to queue (A_1 and A_2) and (A_{3_1} and A_{3_2});

The second grouping allows for the possibility of a reduction in the state space of the joint automata, (A_{3_1} and A_{3_2}), since the priority queue is now represented by a single automaton. The results obtained are presented in Table 6

Models			(1)(2)(3 ₁)(3 ₂)		(1, 3 ₁)(2, 3 ₂)		(1, 2)(3 ₁ , 3 ₂)		(1, 2)(3 ₁ , 3 ₂) \mathfrak{R}	
			CPU	Mem	CPU	Mem	CPU	Mem	CPU	Mem
$C_1=5$	$C_2=5$	$C_3=10$	0.15	21	0.22	27	0.07	32	0.04	64
$C_1=5$	$C_2=5$	$C_3=20$	0.64	82	0.89	94	0.30	118	0.16	67
$C_1=10$	$C_2=10$	$C_3=10$	0.62	81	0.92	94	0.31	101	0.17	63
$C_1=10$	$C_2=10$	$C_3=20$	2.43	317	4.00	346	1.44	364	0.64	201
$C_1=10$	$C_2=10$	$C_3=30$	5.50	709	9.72	754	3.82	801	1.79	429
$C_1=10$	$C_2=10$	$C_3=50$	15.20	1 963	29.04	2 039	10.24	2 200	5.28	1 153
$C_1=20$	$C_2=20$	$C_3=50$	58.92	7 824	120.78	7 989	47.53	8 106	24.37	4 186

Table 6: Queuing Network Model

The gains with this example are not as impressive as they were with the resource sharing model, but it should be remembered that there are relatively few automata (just four) in this example. Notice also that the CPU times obtained with the first grouping is *worse* than in the non-grouped case. This is a result of the fact that this model incorporates synchronizing events combined with functions that cannot be removed using simplification 2. The first grouping eliminates these events, but this results in an increase in the number of functions that must be evaluated. In all models that possess synchronizing events, a grouping procedure that includes a subset of these events must incorporate the evaluation of tensor products (and not only tensor sums). The evaluation of tensor products can increase the complexity by increasing the number of non-zero elements to be multiplied by the vector. Especially if the tensor products contain functional elements, the number of functions to be evaluated will increase. This effect is apparent in the table of results. Elimination of synchronizing events may prove useful in certain very large problems since descriptors that include synchronizing events require an additional probability vector of size equal to the global number of states.

The gains obtained from the second grouping result from the elimination of functional elements from the grouped descriptors. All functions depend on the states of automata A_{3_1} and A_{3_2} . Additionally, the elimination of non-reachable states reduces the CPU computation time that is needed and also saves memory space.

The experiments clearly show that the benefits that accrue from grouping are non-negligible, so long as the number of function evaluations do not rise drastically as a result.

In fact, it seems that function evaluation should be the main concern in choosing which automata to group together. Indirectly, functions also play an important role in identifying non-reachable states, the elimination of which permit important reductions in CPU time and memory.

References

- [1] K. Atif. Modelisation du Parallelisme et de la Synchronisation. Thèse de Docteur de l'Institut National Polytechnique de Grenoble, 24 September 1992, Grenoble, France.
- [2] F. Baccelli, A. Jean-Marie and I. Mitrani, Editors, Quantitative Methods in Parallel Systems, Part I : Stochastic Process Algebras; *Basic Research Series*, Springer, 1995.
- [3] G. Balbo, S. Bruell and M. Sereno. Arrival Theorems for Product-form Stochastic Petri Nets; *Proc. of ACM Sigmetrics Conference 1994*, Nashville, pp. 87-97, 1994.
- [4] C. Berge. *Graphes et Hypergraphes*. Dunod, Paris, 1970.
- [5] R. Boucherie. A Characterization of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets. *IEEE Transactions on Soft. Eng.*, Vol 20, pp. 536–544, 1994.
- [6] P. Buchholz. Equivalence Relations for Stochastic Automata Networks. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.
- [7] P. Buchholz. Aggregation and Reduction Techniques for Hierarchical GCSPNs. *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Press, pp. 216–225, October 1993.
- [8] P. Buchholz. Hierarchical Markovian Models – Symmetries and Aggregation; *Modelling Techniques and Tools for Computer Performance Evaluation*, Ed. R. Pooley, J. Hillston, Edinburgh, Scotland, pp. 234–246, 1992.
- [9] G. Chiola, C. Dutheillet, G. Franceschinis and S. Haddad. Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications. *IEEE Transactions on Computers*, Vol 42, No. 11, pp. 1343–1360, 1993.
- [10] G. Ciardo and K. Trivedi. Solution of Large GSPN Models. *Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Marcel Dekker Publisher, New York, pp. 565–595, 1991.
- [11] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Trans. Comput*, Vol. C-30, No. 2, pp. 1099–1109, 1981.
- [12] S. Donatelli. Superposed Stochastic Automata: A Class of Stochastic Petri Nets with Parallel Solution and Distributed State Space. *Performance Evaluation*, Vol. 18, pp. 21–36, 1993.
- [13] P. Fernandes, B. Plateau and W.J. Stewart. Efficient Vector-Descriptor Multiplications in Stochastic Automata Networks. INRIA Report # 2935. Anonymous ftp <ftp.inria.fr/INRIA/Publication/RR>.
- [14] J-M. Fourneau and F. Quessette. Graphs and Stochastic Automata Networks. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

-
- [15] G. Franceschinis and R. Muntz. Computing Bounds for the Performance Indices of Quasi-lumpable Stochastic Well-Formed Nets. *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Press, pp. 148–157, October 1993.
- [16] W. Henderson and D. Lucic. Aggregation and Disaggregation through Insensitivity in Stochastic Petri Nets; *Performance Evaluation*, Vol. 17, pp. 91–114, 1993.
- [17] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, U. Herzog, M. Rettelbach, Editors, Arbeitsberichte, Band 27, No. 4, Erlangen, November 1994.
- [18] J. Hillston. Computational Markovian Modelling using a Process Algebra. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.
- [19] P. Kemper. Closing the Gap between Classical and Tensor Based Iteration Techniques. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.
- [20] B. Plateau. On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms. *Proc. ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, Austin, Texas, August 1985.
- [21] B. Plateau and K. Atif. Stochastic Automata Network for Modelling Parallel Systems. *IEEE Trans. on Software Engineering*, Vol. 17, No. 10, pp. 1093–1108, 1991.
- [22] B. Plateau and J.M. Fourneau. A Methodology for Solving Markov Models of Parallel Systems. *Journal of Parallel and Distributed Computing*. Vol. 12, pp. 370–387, 1991.
- [23] B. Plateau, J.M. Fourneau and K.H. Lee. PEPS: A Package for Solving Complex Markov Models of Parallel Systems. In R. Puigjaner, D. Potier, Eds., *Modelling Techniques and Tools for Computer Performance Evaluation*, Spain, September 1988.
- [24] W.H. Sanders and J.F. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks, *IEEE Jour. on Selected Areas in Communication*, Vol. 9, No. 1, pp. 25–36, 1991.
- [25] M. Siegle. On Efficient Markov Modelling. In *Proc. QMIPS Workshop on Stochastic Petri Nets*, pp. 213–225, Sophia-Antipolis, France, November 1992.
- [26] C. Simone and M.A. Marsan. The Application of the EB-Equivalence Rules to the Structural Reduction of GSPN Models. *Journal of Parallel and Distributed Computing*, Vol. 15, No. 3, pp. 296–302, 1991.
- [27] W.J. Stewart. *An Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, New Jersey, 1994.
- [28] W.J. Stewart. MARCA: Markov Chain Analyzer. *IEEE Computer Repository* No. R76 232, 1976. Also IRISA Publication Interne No. 45, Université de Rennes, France.
- [29] W.J. Stewart, K. Atif and B. Plateau. The Numerical Solution of Stochastic Automata Networks. *European Journal of Operations Research*, Vol. 86, No. 3, pp. 503–525, 1995.



Unit e de recherche INRIA Lorraine, Technop le de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS L ES NANCY
Unit e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit e de recherche INRIA Rh ne-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

 diteur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399