Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 148 (2003) 436–451
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Stochastics and Statistics

Lumpable continuous-time stochastic automata networks ☆

Oleg Gusak [a,*], Tuğrul Dayar [b], Jean-Michel Fourneau [c]

[a] School of Interdisciplinary Computing & Engineering, University of Missouri-Kansas City, 5100 Rockhill Road, Kansas City, MO 64110-2499, USA
[b] Department of Computer Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey
[c] Lab. PRiSM, Université de Versailles, 45 av. des États Unis, 78035 Versailles Cedex, France

Received 2 February 2001; accepted 24 April 2002

Abstract

The generator matrix of a continuous-time stochastic automata network (SAN) is a sum of tensor products of smaller matrices, which may have entries that are functions of the global state space. This paper specifies easy to check conditions for a class of ordinarily lumpable partitionings of the generator of a continuous-time SAN in which aggregation is performed automaton by automaton. When there exists a lumpable partitioning induced by the tensor representation of the generator, it is shown that an efficient aggregation-iterative disaggregation algorithm may be employed to compute the steady-state distribution. The results of experiments with two SAN models show that the proposed algorithm performs better than the highly competitive block Gauss–Seidel in terms of both the number of iterations and the time to converge to the solution.

Keywords: Markov processes; Stochastic automata networks; Ordinary lumpability; Aggregation with iterative disaggregation

1. Introduction

Compared to simulative techniques, the attraction for Markov chains (MCs) lies in that they provide exact results (up to computer precision) for performance or reliability measures of interest through numerical analysis. Unfortunately, Markovian modeling and analysis is liable to the problem of state space explosion since it is not uncommon to encounter systems requiring millions of states in most realistic models today. It is currently a challenge to handle the enormous state spaces of MCs underlying such models. Therefore, structured representations amenable to tensor (i.e., Kronecker) based numerical techniques are gaining popularity. The essence of the tensor based approach is to model the system at hand in the form of interacting components so that its underlying MC can be represented as a sum of tensor products of component matrices, and its state space is given by the cross product of the state spaces of the components. Such a representation obviates the need to store the underlying MC and

☆ This work is supported by grant TÜBİTAK-CNRS and is done while the first author was at the Department of Computer Engineering, Bilkent University.
* Corresponding author. Tel.: +1-816-235-5940; fax: +1-816-235-5159.
E-mail address: gusako@umkc.edu (O. Gusak).

0377-2217/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0377-2217(02)00431-9

mitigates the state space explosion problem. The concept of using tensor algebra [11] to define large MCs underlying structured representations appears in compositional Markovian models such as stochastic automata networks (SANs) [26,27,29,32], different classes of superposed Stochastic Petri Nets (SPNs) [15], structured and hierarchical Markovian models [2,8]. In this work we concentrate on the analysis of continuous-time SANs.

In a SAN (see [32, Chapter 9]), each component of the global system is modeled by a stochastic automaton. When automata do not interact (i.e., when they are independent of each other), description of each automaton consists of local transitions only. In other words, local transitions are those that affect the state of one automaton. Local transitions can be constant (i.e., independent of the state of other automata) or they can be functional. In the latter case, the transition is a function of the global state of the system. Interactions among components are captured by synchronizing transitions. Synchronization among automata happens when a state change in one automaton causes a state change in other automata. Similar to local transitions, synchronizing transitions can be constant or functional.

A continuous-time Markovian system of N components can be modeled by a single stochastic automaton for each component. Local transitions of automaton k (denoted by $\mathscr{A}^{(k)}$) are modeled by local transition rate matrix $Q_l^{(k)}$, which has row sums of 0. When there are E synchronizing events in the system, automaton k has the synchronizing transition matrix $Q_e^{(k)}$ that represents the contribution of $\mathscr{A}^{(k)}$ to synchronization $e \in \{1, 2, \ldots, E\}$, and the corresponding diagonal corrector matrix $\overline{Q}_e^{(k)}$. The automaton that triggers a synchronizing event is called the master, the others that get affected by the event are called the slaves. Matrices associated with synchronizing events are either transition rate matrices (corresponding to master automata) or transition probability matrices (corresponding to slave automata). Without loss of generality, we restrict ourselves to the case in which synchronizing transition probability matrices of a SAN have row sums of 1 or 0. In [20], it is shown how a SAN that does not satisfy this condition can be transformed to an equivalent SAN having synchronizing transition probability matrices with row sums of 1 or 0.

The generator corresponding to the global system is given by

$$Q = Q_l + Q_e + \overline{Q}_e, \tag{1}$$

where

$$Q_l = \bigoplus_{k=1}^{N} Q_l^{(k)}, \quad Q_e = \sum_{e=1}^{E} \bigotimes_{k=1}^{N} Q_e^{(k)},$$

$$\overline{Q}_e = \sum_{e=1}^{E} \bigotimes_{k=1}^{N} \overline{Q}_e^{(k)},$$

$\oplus$ is the tensor sum operator and $\otimes$ is the tensor product operator. We refer to the tensor representation in Eq. (1) associated with the generator as the descriptor of the SAN. Assuming that $\mathscr{A}^{(k)}$ has n_k states, the global system has $n = \prod_{k=1}^{N} n_k$ states. The global state i of the system maps to the state vector $(s\mathscr{A}^{(1)}, s\mathscr{A}^{(2)}, \ldots, s\mathscr{A}^{(N)})$, that is, $i \leftrightarrow (s\mathscr{A}^{(1)}, s\mathscr{A}^{(2)}, \ldots, s\mathscr{A}^{(N)})$, where $s\mathscr{A}^{(k)}$ denotes the state of $\mathscr{A}^{(k)}$. When there are functional elements, tensor products become generalized tensor products [29].

When the steady-state probability vector, π, of the global system exists, it satisfies the following system of linear equations:

$$\pi Q = 0, \quad \sum_{j=1}^{n} \pi_j = 1. \tag{2}$$

In order to analyze structured Markovian models efficiently, various algorithms for vector–tensor product multiplication are devised [7,16,17,26] and used as kernels in iterative solution techniques proposed for related high-level formalisms. In particular, application of projection methods to SANs is discussed in [5,32,33] and experiments with circulant preconditioners for SANs appear in [9]. In [34], a recursive implementation of iterative methods based on splittings that take advantage of the tensor structure of the SAN descriptor is introduced. An iterative aggregation–disaggregation algorithm for SANs, in which aggregation at each iteration is done with respect to the states of an automaton chosen adaptively, appears in [4]. Further improvements in time and space requirements of numerical solution techniques can be obtained by employing reachability analysis and sparse storage schemes

[7] possibly with reordering and grouping of automata under the presence of functional transitions [17] in the efficient vector–tensor product multiplication algorithms. It is also possible to use the directed graph induced by the tensor representation to develop efficient analysis methods for SANs [19–21].

Lumping (i.e., exact aggregation) [23] is another approach that can aid in the analysis of systems having large state space. In the rest of the paper we use the concepts of lumping and exact aggregation interchangeably. Different kinds of lumpability in finite Markov chains and their properties are considered in [1]. Results on exact aggregation of large systems whose MCs are composed of tensor products appear in [2,30]. Notion of exact performance equivalence for SANs is introduced in [6] and application of ordinary and exact lumpability to SANs is discussed in [3].

In this work, we consider the application of ordinary lumpability to continuous-time SANs. Let the state space $\mathscr{S} = \{1, 2, \ldots, n\}$ of a MC given by Q be partitioned into K subsets $\mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_K$ such that $\cup_{k=1}^{K} \mathscr{S}_k = \mathscr{S}$ and $\mathscr{S}_k \cap \mathscr{S}_l = \emptyset$ for $k \neq l$. Following [1], we say that a MC is ordinarily lumpable with respect to the partitioning $\mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_K$ if for all states $x, y \in \mathscr{S}_k$ and subset $\mathscr{S}_l, k, l = 1, 2, \ldots, K$, $\sum_{j \in \mathscr{S}_l} P(x, j) = \sum_{j \in \mathscr{S}_l} P(y, j)$. In other words, Q is ordinarily lumpable if each block in the partitioning of Q has equal row sums. Since in this paper we consider only ordinarily lumpable partitionings, in what follows we name ordinarily lumpable partitionings as lumpable partitionings.

In contrast to the existing work on exact aggregation of SANs and related high-level formalisms, we consider a SAN in its general form with functional transitions. In other words, we assume that the descriptor corresponding to the SAN model at hand is a sum of generalized tensor products. Using the basic results in [22], we derive easy to check conditions for a continuous-time SAN on descriptions of its automata and their ordering that enable us to identify a class of lumpable partitionings in which lumping is performed automaton by automaton. We remark that most of the existing work on exact aggregation of tensor based formalisms amounts to defining equivalence relations among states in a component of the modeled system or among the components of the system [2,3,6]. The goal of our work is to identify lumpable partitionings of Q induced by the block structure of tensor product. Our approach of lumping one or more automata of a SAN independently of each other is a special case of the lumpability and performance equivalence considered in [2,6]. However, our approach also enables the identification of lumpable partitionings in which blocks are composed of multiple automata but individual automaton cannot be lumped. Note that this kind of lumpable partitionings cannot be revealed using the approach discussed in [2,6] since the conditions derived therein are applied to each automaton separately. Furthermore, simplicity of the conditions for lumpability that we impose on the SAN description allows us to show that some of the SAN models that have been considered before are lumpable.

Finally, we remark that existing work on lumpability in tensor based formalisms consider the analysis of the aggregated system, whereas we aim at solving the original system and do not assume a specific Markov reward structure. Continuing our work in [22], we propose an aggregation–iterative disaggregation (AID) algorithm for a class of lumpable continuous-time SANs and compare its performance with that of block Gauss–Seidel (BGS) on two continuous-time SAN models. To the best of our knowledge, this is the first numerical study of AID on lumpable continuous-time SANs.

In the next section, we specify conditions for a class of lumpable continuous-time SANs with functional transitions. In Section 3, we present an example of a lumpable SAN. In Section 4, we introduce the efficient AID algorithm for SANs having lumpable partitionings induced by tensor product. In Section 5, we present the results of numerical experiments with two lumpable SAN models, and in Section 6, we conclude.

2. Lumpable partitionings induced by the block structure of tensor product

Let us first discuss properties associated with the partitioning of a matrix that is a tensor product

of square matrices. The partitionings we consider are induced by the block structure of tensor product and hence have blocks of equal size. See [11] for the definition of tensor product and related concepts. We first specify conditions under which each block of such a partitioning has equal row sums and extend this result to a matrix that is a sum of tensor products of square matrices. Then, using the equal row sums property, we derive conditions under which the MC underlying a SAN model is lumpable.

Let A be the tensor product of N square matrices $A^{(k)}$, $k = 1, 2, \ldots, N$, as in

$$A = \bigotimes_{k=1}^{N} A^{(k)}, \tag{3}$$

where n_k is the order of $A^{(k)}$. Similar to the global state of a SAN, each row of A can be mapped to the vector $(rA^{(1)}, rA^{(2)}, \ldots, rA^{(N)})$, where $rA^{(k)}$ denotes the row index of $A^{(k)}$. In the same way, we can map each column of A to $(cA^{(1)}, cA^{(2)}, \ldots, cA^{(N)})$, where $cA^{(k)}$ denotes the column index of $A^{(k)}$. From the definition of tensor product [11, p. 117] for any $m \in \{2, 3, \ldots, N\}$ the matrix A can be partitioned into K^2 blocks of the same order as

$$A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1K} \\ A_{21} & A_{22} & \ldots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \ldots & A_{KK} \end{pmatrix}, \tag{4}$$

where $K = \prod_{k=1}^{m-1} n_k$,

$$A_{ij} = \xi_{ij} \bigotimes_{k=m}^{N} A^{(k)} = \left(\prod_{k=1}^{m-1} A^{(k)}\left(rA^{(k)}, cA^{(k)}\right) \right) \bigotimes_{k=m}^{N} A^{(k)}, \tag{5}$$

$i \leftrightarrow (rA^{(1)}, rA^{(2)}, \ldots, rA^{(m-1)})$, and $j \leftrightarrow (cA^{(1)}, cA^{(2)}, \ldots, cA^{(m-1)})$. Now, let us assume that the matrices $A^{(k)}$ may have functional elements such that the value of the function depends on the row index of A. We denote by $A^{(k)}[A^{(l)}]$ a functional dependency between the matrices $A^{(k)}$ and $A^{(l)}$ when the value of at least one element in $A^{(k)}$ depends on $rA^{(l)}$. We say, $A^{(k)}$ functionally depends on $A^{(l)}$.

The following theorem specifies a simple and easy to check condition for equal row sums in all blocks A_{ij} of the partitioning in Eq. (4).

Theorem 1. *Each block A_{ij} in Eq. (4) has equal row sums for any $m \in \{2, 3, \ldots, N\}$ if the matrices $A^{(k)}$, $k = 1, 2, \ldots, N$, in Eq. (3) can be reordered and renumbered so that $A^{(k)}[A^{(l)}]$ implies $l \in \{1, 2, \ldots, k-1\}$ and each $A^{(k)}$, $k = m, m+1, \ldots, N$, has equal row sums.*

Proof. We must show in Eq. (4) that $A_{ij}u = l_{ij}u$ for $i, j = 1, 2, \ldots, K$, where l_{ij} is a constant value that depends only on i, j and m, and u represents the column vector of 1's with appropriate length. We are dropping m from l_{ij} since m is fixed for the chosen partitioning. The value $A^{(k)}(rA^{(k)}, cA^{(k)})$ and consequently ξ_{ij} in Eq. (5) may be a function of $rA^{(l)}$ for some $l \in \{1, 2, \ldots, k-1\}$ but is still fixed for the particular mapping $i \leftrightarrow (rA^{(1)}, rA^{(2)}, \ldots, rA^{(m-1)})$. Furthermore, $\otimes_{k=m}^{N} A^{(k)}$ may very well depend on $rA^{(l)}$ for some $l \in \{1, 2, \ldots, m-1\}$.

By the assumption regarding equal row sums in the statement of the theorem, we have $A^{(k)}u = v^{(k)}u$ for $k = 1, 2, \ldots, N$ and for some constant value $v^{(k)}$ that depends only on k. Then

$$\left(\xi_{ij} \bigotimes_{k=m}^{N} A^{(k)}\right)u = \xi_{ij} \bigotimes_{k=m}^{N} \left(A^{(k)}u_{n_k}\right) = \xi_{ij} \bigotimes_{k=m}^{N} \left(v^{(k)}u_{n_k}\right) = \left(\xi_{ij} \prod_{k=m}^{N} v^{(k)}\right)u,$$

where u_{n_k} denotes the column vector of n_k 1's. Hence, when all $A^{(k)}$ have equal row sums, each block A_{ij} in Eq. (4) under the assumed ordering of the matrices retains the equal row sums property, and $l_{ij} = \xi_{ij} \prod_{k=m}^{N} v^{(k)}$. □

We assume that properties of functional elements that may be present in the matrices $A^{(k)}$ are known so that it is possible to state whether row sums of $A^{(k)}$ are equal or not. An example is a set of functional elements in a row of $A^{(k)}$ such that for each row of A in the mapping, only one functional element in the particular row of $A^{(k)}$ evaluates to a constant value, say, α and all its other elements evaluate to 0. Obviously, for all rows of A in the mapping, the sum of the functional elements in the

particular row of $A^{(k)}$ will be α. See also Section 3 and [22] for examples of matrices that have functional elements and possess the equal row sums property.

Now, we introduce a definition and then state a more relaxed version of Theorem 1 for the case of cyclic functional dependencies. See also Theorem 1 in [22].

Definition 1. Let $G(\mathscr{V},\mathscr{E})$ be the directed graph (digraph) associated with the matrices $A^{(k)}$, $k = 1,2,\ldots,N$, in which the vertex $v_k \in \mathscr{V}$ represents $A^{(k)}$ and the edge $(v_k,v_l) \in \mathscr{E}$ if $A^{(k)}[A^{(l)}]$. Then we say that there is a cyclic functional dependency among the matrices $A^{(k)}$ if and only if a topological ordering of G does not exist.

Detailed description of the topological ordering algorithm for digraphs can be found in [10, pp. 485–487].

Theorem 2. *There exists $m \in \{2,3,\ldots,N\}$ and an ordering of matrices $A^{(k)}$, $k = 1,2,\ldots,N$, such that each block A_{ij}, $i,j = 1,2,\ldots,K$, in Eq.* (4) *has equal row sums if the digraph associated with the matrices $A^{(k)}$ has more than one strongly connected component (SCC) and each $A^{(k)}$, $k = m,m+1,\ldots,N$, has equal row sums.*

Proof. Without loss of generality, let the N matrices be partitioned into two SCCs $\mathscr{S}_1 = \{A^{(1)}, A^{(2)},\ldots,A^{(m-1)}\}$ and $\mathscr{S}_2 = \{A^{(m)},A^{(m+1)},\ldots,A^{(N)}\}$. Let $\mathscr{S}_2$ be formed of cyclically dependent matrices (i.e., $N-m>0$). Note that it is possible for the matrices in $\mathscr{S}_1$ to depend on $rA^{(k)}$, where $A^{(k)} \in \mathscr{S}_2$, or vice versa, but both type of functional dependencies cannot be present simultaneously. That is, the matrices in $\mathscr{S}_1$ and the matrices in $\mathscr{S}_2$ cannot be mutually dependent; otherwise we could not have two partitions $\mathscr{S}_1$ and $\mathscr{S}_2$. Now, let $\mathscr{S}_2[\mathscr{S}_1]$; in other words, there is at least one $k \in \{m, m+1,\ldots,N\}$ for which $A^{(k)}$ depends on $rA^{(l)}$ for some $l \in \{1,2,\ldots,m-1\}$. If $\mathscr{S}_1[\mathscr{S}_2]$ were the case, one could exchange $\mathscr{S}_2$ and $\mathscr{S}_1$.

The equal row sums property of each block A_{ij} follows from two arguments. First, the scalars ξ_{ij} in Eq. (5) are independent of the row indices of $A^{(k)}$ for $k = m,m+1,\ldots,N$, and they can still be computed in the same way since each ξ_{ij} is the product of $(m-1)$ values, the lth one coming from a specific element of $A^{(l)}$, where $l = 1,2,\ldots,m-1$. Even when $\mathscr{S}_1$ is formed of cyclically dependent matrices (implying $m>1$), each ξ_{ij} is a well defined constant. Second, the matrices $(\xi_{ij} \otimes_{k=m}^{N} A^{(k)})$ in Eq. (5) still have equal row sums since, by the assumption in the statement of the theorem, each $A^{(k)}$ for $k = m,m+1,\ldots,N$ has equal row sums. Hence, each block A_{ij} in Eq. (4) has equal row sums for the particular value of m.

When there are $S>2$ SCCs $\mathscr{S}_p$, $p = 1,2,\ldots,S$, in the digraph associated with the matrices $A^{(k)}$, the theorem also holds since there are no cyclic functional dependencies among the $\mathscr{S}_p$ and they can be reordered and then renumbered so that for $p = 2,3,\ldots,S$ $\mathscr{S}_p[\mathscr{S}_o]$ implies $o \in \{1,2,\ldots,p-1\}$. In this order, there are clearly $(S-1)$ partitionings for which each square block A_{ij} in Eq. (4) has equal row sums. □

Next, we state a result that extends Theorems 1 and 2 to a square matrix given as the sum of E tensor products.

Corollary 1. *If there exists the same value of m for which each tensor product $\otimes_{k=1}^{N} B_e^{(k)}$ in $B = \sum_{e=1}^{E} \otimes_{k=1}^{N} B_e^{(k)}$, where $B_e^{(k)}$ is of order n_k for $e = 1,2,\ldots,E$, satisfies the conditions of Theorems* 1 *or* 2, *then each block B_{ij}, $i,j = 1,2,\ldots,K$, in the partitioning of B specified by m as in Eq.* (4) *has equal row sums.*

When B is a generator matrix that satisfies the conditions of Corollary 1, B is said to be lumpable [23, p. 124]. Now, consider the application of Corollary 1 to continuous-time SANs. Q in Eq. (1) can be considered as a sum of two terms. The first term is Q_l and the second term is the sum of Q_e and $\overline{Q}_e$.

Q_l is the tensor sum of N matrices and can be written as a sum of tensor products:

$$\begin{aligned} Q_l &= \oplus_{k=1}^{N} Q_l^{(k)} \\ &= \sum_{k=1}^{N} I_{n_1} \otimes I_{n_2} \otimes \cdots \otimes I_{n_{k-1}} \otimes Q_l^{(k)} \\ &\quad \otimes I_{n_{k+1}} \otimes \cdots \otimes I_{n_{N-1}} \otimes I_{n_N}, \end{aligned} \tag{6}$$

where I_{n_k} is the identity matrix of order n_k. Identity matrices have row sums of 1 and $Q_l^{(k)}$ have row sums of 0. Hence, Corollary 1 applies through Theorem 2 if the digraph G associated with the matrices $Q_l^{(k)}$ has more than one SCC.

Now, consider the second term composed of Q_e and $\overline{Q}_e$. We can omit $\overline{Q}_e$ from further consideration since $\overline{Q}_e$ contributes only to the diagonal elements of Q. Hence, it influences only the diagonal blocks in a given partitioning of Q. Once we prove that the off-diagonal blocks of a partitioning have equal row sums, the property immediately follows for its diagonal blocks since Q is a generator matrix (i.e., $Qu = 0$).

Q_e is a sum of tensor products. Hence, we can again resort to Corollary 1. However, the condition regarding equal row sums can be violated in two ways: (i) in synchronizing transition rate matrices of master automata, (ii) in synchronizing transition probability matrices of slave automata. A synchronizing transition rate matrix need not have equal row sums of 0. On the other hand, a synchronizing transition probability matrix has row sums of 1 or 0. Hence, the equal row sums property may not hold for synchronizing transition probability matrices either.

We remark that the case in which a synchronizing transition probability matrix has zero rows corresponds to an implicit functional dependency between the master automaton of the synchronizing event and the slave automaton whose synchronizing transition probability matrix has zero rows [20].

Definition 2. If a synchronizing transition probability matrix corresponding to $\mathscr{A}^{(k)}$ of a SAN has at least one zero row, then we say that $\mathscr{A}^{(k)}$ introduces an implicit functional dependency to the SAN description.

Lemma 1. *By introducing functional transitions, a SAN which contains implicit functional dependencies can be transformed to an equivalent SAN which does not contain implicit functional dependencies.*

Proof. Without loss of generality, consider a SAN of N automata and 1 synchronizing event that contains implicit functional dependencies. Let $\mathscr{A}^{(t)}$ be the master automaton of synchronizing event 1. We denote by $\mathscr{Z}^{(k)}$ the set of states of $\mathscr{A}^{(k)}$, $k \neq t$, for which the corresponding rows of $Q_1^{(k)}$ are zeros. In order to obtain an equivalent SAN that does not contain implicit functional dependencies, we replace each nonzero element $Q_1^{(t)}(i,j)$ with the function

$$f(i,j) = \begin{cases} Q_1^{(t)}(i,j), & \text{for all } k, \quad k \neq t, \ s\mathscr{A}^{(k)} \notin \mathscr{Z}^{(k)}, \\ 0, & \text{otherwise.} \end{cases}$$

We also modify each $Q_1^{(k)}$, $k \neq t$, so that if $s\mathscr{A}^{(k)} \in \mathscr{Z}^{(k)}$, then $Q_1^{(k)}(s\mathscr{A}^{(k)}, s\mathscr{A}^{(k)})$ (which is 0) becomes 1. In the same way, we redefine the transitions in $\overline{Q}_1^{(t)}$ and $\overline{Q}_1^{(k)}$, $k \neq t$. The new SAN description does not contain implicit functional dependencies.

In the general case when there are $E > 1$ synchronizing events, we apply the same kind of modification to $Q_e^{(k)}$ and $\overline{Q}_e^{(k)}$ of each event $e \in \{1, 2, \ldots, E\}$ that introduces an implicit functional dependency to the SAN description. The new SAN description does not contain implicit functional dependencies, and hence, all synchronizing transition probability matrices have equal row sums of 1. □

Next, we introduce three definitions concerning functional dependencies and suitable orderings of automata for lumpability.

Definition 3. A SAN that does not contain implicit functional dependencies is said to be in its explicit form.

Definition 4. Let $G(\mathscr{V}, \mathscr{E})$ be the digraph of a SAN in its explicit form in which the vertex $v_k \in \mathscr{V}$ represents $\mathscr{A}^{(k)}$ and the edge $(v_k, v_l) \in \mathscr{E}$ if $\mathscr{A}^{(k)}[\mathscr{A}^{(l)}]$. A reverse topological ordering of $G(\mathscr{V}, \mathscr{E})$ is said to be a proper ordering of the automata of the SAN.

The reason behind using the reverse of the topological ordering in Definition 4 is the direction of the arcs we choose in G to represent dependencies between automata.

Since each vertex in the digraph G corresponds to a unique automaton, in the next definition we use vertices of G and automata interchangeably.

Definition 5. Let $G^{\mathrm{SCC}}(\mathscr{V}^{\mathrm{SCC}}, \mathscr{E}^{\mathrm{SCC}})$ be the digraph of a SAN in its explicit form in which each vertex corresponds to an SCC of $G(\mathscr{V}, \mathscr{E})$, and the edge $(v_i^{\mathrm{SCC}}, v_j^{\mathrm{SCC}}) \in \mathscr{E}^{\mathrm{SCC}}$ if $(v_k, v_l) \in \mathscr{E}$ with $v_k \in v_i^{\mathrm{SCC}}$ and $v_l \in v_j^{\mathrm{SCC}}$. A reverse topological ordering of the digraph $G^{\mathrm{SCC}}(\mathscr{V}^{\mathrm{SCC}}, \mathscr{E}^{\mathrm{SCC}})$ when $|\mathscr{V}^{\mathrm{SCC}}| > 1$ is said to be a quasi-proper ordering of the automata of the SAN.

From Definitions 3 and 4 follows the first part of the next remark. From Definition 5 follows its second part. The theorem that follows the remark specifies sufficient conditions for the lumpability of the generator of a continuous-time SAN.

Remark 1. A proper ordering is a special case of a quasi-proper ordering. Furthermore, a quasi-proper ordering of a SAN in its explicit form exists if and only if the digraph G of the SAN has more than one SCC.

Theorem 3. *The generator underlying a SAN in its explicit form is lumpable if there exists a quasi-proper ordering of the automata and the synchronizing transition rate matrices of all automata have equal row sums. For the given quasi-proper ordering of automata, there are* $(|\mathscr{V}^{\mathrm{SCC}}| - 1)$ *lumpable partitionings, where* $\mathscr{V}^{\mathrm{SCC}}$ *is introduced in Definition* 5.

Proof. Proof of this theorem follows from Eq. (1), Corollary 1, and Remark 1. First, each local transition rate matrix has equal row sums. Since there are no implicit functional dependencies, each synchronizing transition probability matrix has equal row sums as well. Furthermore, synchronizing transition rate matrices of master automata have equal row sums by the assumption of the theorem. Second, by the assumption of the theorem regarding the existence of a quasi-proper ordering, the digraph G of the SAN has at least two SCCs. Hence, there exists at least one m in Theorem 2 such that transitions in $\mathscr{A}^{(l)}$, $l = 1, 2, \ldots, m-1$, do not depend on $s\mathscr{A}^{(k)}$, $k = m, m+1, \ldots, N$. This essentially proves that each off-diagonal block in the partitioning specified by m has equal row sums. Thus, the partitioning is lumpable. For the given quasi-proper ordering, m can assume $(|\mathscr{V}^{\mathrm{SCC}}| - 1)$ distinct values. □

As pointed out before, the equal row sums property is unlikely to be satisfied for synchronizing transition rate matrices. Fortunately, the situation is not hopeless. For some cases in which synchronizing transition rate matrices do not have equal row sums, the generator underlying the SAN can still be lumpable as we next prove.

Theorem 4. *Let* $(v_1^{\mathrm{SCC}}, v_2^{\mathrm{SCC}}, \ldots, v_S^{\mathrm{SCC}})$ *be a quasi-proper ordering of a SAN in its explicit form as in Definition* 5. *Then the generator underlying the SAN is lumpable if there exists* $s \in \{2, 3, \ldots, S\}$ *such that each* $\mathscr{A}^{(k)} \in \bigcup_{i=s}^{S} v_i^{\mathrm{SCC}}$ *satisfies one of the following conditions*:

(i) $\mathscr{A}^{(k)}$ *is not the master of any synchronizing event*;
(ii) *if* $\mathscr{A}^{(k)}$ *is the master of synchronizing event* e, *then* $Q_e^{(k)}$ *has equal row sums*;
(iii) *if* $\mathscr{A}^{(k)}$ *is the master of synchronizing event* e *and* $Q_e^{(k)}$ *does not have equal row sums, then it must be that each automaton in* $\bigcup_{i=1}^{s-1} v_i^{\mathrm{SCC}}$ *is not involved in event* e.

Proof. Assume that the automata are renumbered so that their indices in the given quasi-proper ordering are ascending. First, consider the case in which each automaton in $\bigcup_{i=s}^{S} v_i^{\mathrm{SCC}}$ satisfies either condition (i) or (ii). According to the assumption of the theorem, the SAN is given in its explicit form. Therefore, conditions (i) and (ii) imply equal row sums in synchronizing transition matrices of automata in $\bigcup_{i=s}^{S} v_i^{\mathrm{SCC}}$. Hence, from Theorem 3, the generator underlying the SAN is lumpable and the m in its proof is equal to the smallest index of the automata in $\bigcup_{i=s}^{S} v_i^{\mathrm{SCC}}$. Now, let $\mathscr{A}^{(k)} \in \bigcup_{i=s}^{S} v_i^{\mathrm{SCC}}$ neither satisfy condition (i) nor (ii). In other words, one of the synchronizing transition rate matrices of $\mathscr{A}^{(k)}$ does not have equal row sums.

Now, let $\mathscr{A}^{(k)}$ satisfy condition (iii). Without loss of generality, let $Q_e^{(k)}$ be the only synchronizing transition rate matrix that does not have equal row sums. Recall that equal row sums in the off-diagonal blocks of the partitioning of Q specified by m imply equal row sums in the diagonal blocks. Observe that the off-diagonal blocks in the

partitionings of Q_l and $\sum_{j=1,j\neq e}^{E} \otimes_{k=1}^{N} Q_j^{(k)}$ specified by m have equal row sums as we already proved. What remains is to show that the off-diagonal blocks in the partitioning of $\widetilde{Q} = \otimes_{k=1}^{N} Q_e^{(k)}$ specified by m have equal row sums. From Eq. (5), the ijth block of $\widetilde{Q}$ is given by $\widetilde{Q}_{ij} = (\prod_{k=1}^{m-1} Q_e^{(k)}(i_k, j_k)) \otimes_{k=m}^{N} Q_e^{(k)}$, where $i \leftrightarrow (i_1, i_2, \ldots, i_{(m-1)})$ and $j \leftrightarrow (j_1, j_2, \ldots, j_{(m-1)})$. If $i \neq j$, it must be that for at least one $k \in \{1, 2, \ldots, m-1\}$, $i_k \neq j_k$. From condition (iii), we have $Q_e^{(k)} = I_{n_k}$ for $k = 1, 2, \ldots, m-1$. Hence, for off-diagonal blocks, $i \neq j$ imply $\prod_{k=1}^{m-1} Q_e^{(k)}(i_k, j_k) = 0$. Consequently, each off-diagonal block in the partitioning of $\widetilde{Q}$ specified by m is zero, and therefore has equal row sums. Thus, the generator underlying the SAN is lumpable.

The generalization to the case in which $\mathscr{A}^{(k)}$ has more than one synchronizing transition rate matrix with unequal row sums and to the case in which more than one automaton in $\bigcup_{i=s}^{S} v_i^{\mathrm{SCC}}$ satisfies condition (iii) is immediate. □

The existing work on exact aggregation of SANs [3,6] consider conditions under which a subset of states of an automaton can be lumped. Hence, the partition for which m of Theorem 4 is equal to N, that is, all states of an automaton are aggregated, is a special case of the lumpability discussed in [3]. On the other hand, there are three key issues that distinguish the results of Theorems 3 and 4 from the existing work on exact aggregation [3] and performance equivalence [6] of SANs. First, Theorems 3 and 4 are applicable to SANs that may have functional transitions. Obviously, a SAN descriptor that has functional transitions can be transformed to an equivalent SAN descriptor that does not have functional transitions by introducing new synchronizing events [29]. However, such SAN representations may not be suitable for complex models. More importantly, a relatively large number of synchronizing events may increase the time required to solve the underlying MC of a SAN with an iterative solver. Second, Theorem 4 enables the identification of lumpable partitionings in which blocks are composed of multiple automata but individual automaton cannot be lumped. In other words, Theorem 4 can be used in those cases for which there is no quasi-proper ordering of the SAN with $m = N$. For instance, this happens when a cyclic functional dependency exists among $\mathscr{A}^{(m)}, \mathscr{A}^{(m+1)}, \ldots, \mathscr{A}^{(N)}$, where $1 < m < N$. The approach in [3,6] aims at identifying equivalent states in an automaton of a SAN, and hence, cannot reveal lumpable partitionings in which blocks are composed of states belonging to more than one automaton unless the lumped automata are first grouped into a single automaton and the conditions in [3,6] are applied to the grouped automaton. Finally, in contrast to the work in [3,6], we do not assume a specific Markov reward structure associated with the lumped states, and we aim at solving the original (not aggregated) system. Note also that if one wants to follow the approach in [3,6], a relatively complex algorithm must be run on the matrices of each automaton to identify its equivalent states when the physical description of the underlying model is not available. The conditions of Theorem 4 do not require this and are easy to check.

When a SAN has a quasi-proper ordering as in Theorem 4, its automata can be partitioned for some m into two subsets, $\mathscr{S}_1$ and $\mathscr{S}_2$, so that functional transitions of the automata in $\mathscr{S}_1 = \{\mathscr{A}^{(1)}, \mathscr{A}^{(2)}, \ldots, \mathscr{A}^{(m-1)}\}$ do not depend on the states of the automata in $\mathscr{S}_2 = \{\mathscr{A}^{(m)}, \mathscr{A}^{(m+1)}, \ldots, \mathscr{A}^{(N)}\}$. If the automata belonging to the two subsets were completely independent of each other, then lumpability of the SAN would be obvious and the analyses of the two subsystems corresponding to $\mathscr{S}_1$ and $\mathscr{S}_2$ could be carried out separately. However, this need not be the case. First, dependency between the two subsets exists when functional transitions of the automata in $\mathscr{S}_2$ depend on the states of the automata in $\mathscr{S}_1$. Second, dependency between the two subsets may exist through synchronizing events. The slave automata in $\mathscr{S}_2$ may depend on their masters in $\mathscr{S}_1$. In this case, Theorem 4 applies if the synchronizing transition rates of the masters functionally do not depend on the states of the slaves. An example is a SAN model of two finite queues working in tandem under the assumption that departures from the first queue are dropped when the second queue is full. Dependency through synchronizing events also takes place when there are master automata in $\mathscr{S}_2$ whose slaves are in $\mathscr{S}_1$ and each transition rate

matrix of the master automata has equal row sums. In summary, when the automata in $\mathscr{S}_2$ satisfy condition (i) of Theorem 4, synchronizing transitions that take place in the automata of $\mathscr{S}_1$ do not depend on the automata in $\mathscr{S}_2$ but may affect their state. Condition (ii) explicitly requires equal row sums and is a special case which may rarely occur in SAN models. When the automata in $\mathscr{S}_2$ satisfy condition (iii), synchronizing events that take place in these automata do not affect the states of the automata in $\mathscr{S}_1$.

In order to apply Theorem 4 to a continuous-time SAN, its automata should be put in quasi-proper ordering and renumbered accordingly. Then, for v_i^{SCC}, $i = S, S-1, \ldots, 2$, where $S = |\mathscr{V}^{\mathrm{SCC}}|$, the conditions (i), (ii), and (iii) of Theorem 4 should be exercised on each automaton in $\cup_{j=i}^{S} v_j^{\mathrm{SCC}}$. If each automaton in $\cup_{j=i}^{S} v_j^{\mathrm{SCC}}$ satisfies at least one of the three conditions, then there exists a lumpable partitioning for the given quasi-proper ordering of the automata. The value of m in the lumpable partitioning is equal to the smallest index among the automata in $\cup_{j=i}^{s} v_j^{\mathrm{SCC}}$.

To find a quasi-proper ordering of the automata in a SAN, the SCCs of the functional dependency graph of the SAN should be determined. The SCCs of $G(\mathscr{V}, \mathscr{E})$ can be found in $O(N + |\mathscr{E}|)$ comparisons assuming G is stored as an adjacency list. Since the number of vertices and the number of edges in G^{SCC} cannot exceed respectively the number of automata and the number of edges in G, a topological ordering of G^{SCC} can also be found in $O(N + |\mathscr{E}|)$. Thus, a quasi-proper ordering of a SAN can be found in $O(N + |\mathscr{E}|)$. For a given quasi-proper ordering of automata, the maximum number of automata that can be tested for the three conditions of Theorem 4 is $(N-1)$. Note that only synchronizing transition matrices are tested for the three conditions. Hence, the first condition requires at most $E(N-1)$ comparisons. Let $n_{\max} = \max_i n_i$, where $i = 1, 2, \ldots, N$. Then the second condition requires at most $E(n_{\max} - 1)$ comparisons since there can be at most E master automata in the SAN. Note that when checking for equal row sums in a synchronizing transition rate matrix of an automaton, there is no need to compute the row sums of the matrix. The negated row sums are available in the corresponding diagonal corrector matrix. The third condition of Theorem 4 requires at most $E(N-1)$ comparisons assuming that the smallest indexed automaton involved in each event of the SAN is known. Thus, the total number of comparisons required to check the conditions of Theorem 4 is $2E(N-1) + E(n_{\max} - 1)$.

Theorem 4 enables us to identify lumpable partitionings in SAN models of the mass storage problem [12], the three queues problem [16], and the pushout problem [18]. See also [22] and the references therein for examples of lumpable discrete-time SAN models. In the next section, we use the SAN model of the mass storage problem as an example to show that it is not difficult to apply Theorem 4 to a continuous-time SAN model.

3. A lumpable continuous-time SAN

As an example of a lumpable continuous-time SAN, we consider a model of a robotic tape library named as the mass storage problem. For brevity, here we give the symbolic description of the corresponding SAN model. The detailed description of the underlying physical model, its parameters, and the design decisions can be found in [12]. Results of numerical experiments with this model appear in [34].

The SAN model of the mass storage problem consists of five automata and three synchronizing events. We number the automata from 1 to 5 and the synchronizing events from 1 to 3. All local transition rate matrices have equal row sums of 0. Hence, we omit them from further consideration, but remark that transitions in $Q_l^{(2)}$ depend on the state of $\mathscr{A}^{(1)}$. Furthermore, $\mathscr{A}^{(1)}$ is not involved in the first two events. Hence, $Q_1^{(1)} = Q_2^{(1)} = I_{n_1}$. In event 3, $\mathscr{A}^{(1)}$ acts as the master, and we have

$$Q_3^{(1)} = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ \alpha_1 & 0 & \ddots & \ddots & \vdots \\ 0 & \alpha_2 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & \alpha_{n_1-1} & 0 \end{pmatrix}.$$

$\mathscr{A}^{(2)}$ is a slave automaton of event 1; but it is not involved in the other two events, and we have

$$Q_1^{(2)} = \begin{pmatrix} f_0 & f_1 & \cdots & f_{n_3-1} & 0 & \cdots & 0 \\ 0 & f_0 & f_1 & \cdots & f_{n_3-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & f_0 & \ddots & \ddots & f_{n_3-1} \\ \vdots & \ddots & \ddots & 0 & f_0 & \ddots & f_{n_3-2} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & f_0 \end{pmatrix},$$

$$Q_2^{(2)} = Q_3^{(2)} = I_{n_2}.$$

The values of the functions $f_0, f_1, \ldots, f_{n_3-1}$ depend on the states of $\mathscr{A}^{(3)}$ (and $\mathscr{A}^{(2)}$). These functions are defined so that in each row only one of the functions evaluates to 1, others evaluate to 0. Hence, $Q_1^{(2)}$ has constant row sums of 1. $\mathscr{A}^{(3)}$ is a slave automaton of event 1, acts as the master of event 2, and does not participate in event 3. We have

$$Q_1^{(3)} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix},$$

$$Q_2^{(3)} = \begin{pmatrix} 0 & \lambda & 0 & \cdots & 0 \\ \vdots & 0 & \lambda & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 & \lambda \\ 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}, \quad Q_3^{(3)} = I_{n_3}.$$

$\mathscr{A}^{(4)}$ is not involved in event 1 (i.e., $Q_1^{(4)} = I_{n_4}$); but it is a slave automaton of events 2 and 3, and we have

$$Q_2^{(4)} = \begin{pmatrix} p & \bar{p} & 0 & \cdots & 0 \\ 0 & p & \bar{p} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & 0 & p & \bar{p} \\ \bar{p} & 0 & \cdots & 0 & p \end{pmatrix},$$

$$Q_3^{(4)} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & 0 & 1 \\ 1 & 0 & \cdots & \cdots & 0 \end{pmatrix},$$

where $0 < p < 1$ and $\bar{p} = 1 - p$. Finally, $\mathscr{A}^{(5)}$ is the master automaton of event 1; but it is not involved in the other two events, and we have

$$Q_1^{(5)} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ \gamma & 0 & \cdots & 0 \end{pmatrix}, \quad Q_2^{(5)} = Q_3^{(5)} = I_{n_4}.$$

Now, let us check the lumpability conditions of Theorem 4 using the information in Table 1 and the matrices of the SAN model. First, none of the synchronizing transition probability matrices have zero rows. Hence, the SAN model of the mass storage problem is in its explicit form. Second, from the last two lines in Table 1, the digraph G of the SAN has the two edges (v_2, v_3) and (v_2, v_1). This digraph is acyclic and it has $S = N = 5$ SCCs. Therefore, there exists a proper ordering of the automata of the SAN. Any ordering in which $\mathscr{A}^{(2)}$ is placed after $\mathscr{A}^{(1)}$ and $\mathscr{A}^{(3)}$ is a proper ordering. Consider, for instance, the proper ordering $(\mathscr{A}^{(\tilde{1})}, \mathscr{A}^{(\tilde{2})}, \mathscr{A}^{(\tilde{3})}, \mathscr{A}^{(\tilde{4})}, \mathscr{A}^{(\tilde{5})})$, where $\tilde{1} = 5$, $\tilde{2} = 3$, $\tilde{3} = 1$, $\tilde{4} = 4$, and $\tilde{5} = 2$. For any $s \in \{\tilde{2}, \tilde{3}, \tilde{4}, \tilde{5}\}$, the partitioning of the generator specified by s is lumpable as we next explain.

We first remark that $\mathscr{A}^{(\tilde{5})}$ and $\mathscr{A}^{(\tilde{4})}$ satisfy condition (i) of Theorem 4 meaning neither $\mathscr{A}^{(2)}$ nor $\mathscr{A}^{(4)}$ is the master of any synchronizing event. This implies lumpability when $s \in \{\tilde{4}, \tilde{5}\}$. Second, $\mathscr{A}^{(\tilde{3})}$ satisfies condition (iii) implying lumpability when $s = \tilde{3}$. This is because $\mathscr{A}^{(1)}$ is the master of synchronizing event 3, $Q_3^{(1)}$ does not have equal row sums, and $\mathscr{A}^{(i)}$, $i = \tilde{1}, \tilde{2}$, are not involved in synchronizing event 3. Similar to $\mathscr{A}^{(\tilde{3})}$, $\mathscr{A}^{(\tilde{2})}$ also satisfies condition (iii) implying lumpability when $s = \tilde{2}$. In synchronizing event 2, $\mathscr{A}^{(3)}$ acts as the master, $Q_2^{(\tilde{3})}$ does not have equal row sums, and $\mathscr{A}^{(\tilde{1})}$ is not involved in synchronizing event 2. Thus, for the chosen proper ordering of automata,

Table 1
Summary information for the mass storage problem

Event	Master	Slave(s)	Dependencies
1	$\mathscr{A}^{(5)}$	$\mathscr{A}^{(2)}$, $\mathscr{A}^{(3)}$	
2	$\mathscr{A}^{(3)}$	$\mathscr{A}^{(4)}$	$\mathscr{A}^{(2)}[\mathscr{A}^{(3)}]$
3	$\mathscr{A}^{(1)}$	$\mathscr{A}^{(4)}$	$\mathscr{A}^{(2)}[\mathscr{A}^{(1)}]$

there are four lumpable partitionings of the generator for $s \in \{\tilde{2}, \tilde{3}, \tilde{4}, \tilde{5}\}$.

Observe that when the number of SCCs in G is equal to the number of automata, that is, when the dependency graph is acyclic, the SAN may have the largest number of lumpable partitionings, $(N-1)$, for a given quasi-proper ordering of automata. This gives more flexibility to the performance analyst in choosing a lumpable partitioning that better suits the aggregation-iterative disaggregation algorithm, which we introduce in the next section. On the other hand, when S takes its largest value, N, a larger number of comparisons is required to check the three conditions in Theorem 4. Also, when $S = N$, a larger number of quasi-proper orderings may exist that need to be tested against the conditions of the theorem as indicated by our complexity analysis.

4. AID algorithm for lumpable SANs

Assuming that the generator of a continuous-time SAN is lumpable and has a steady-state distribution, we propose Algorithm 1, which is a modified form of Koury–McAllister–Stewart's iterative aggregation–disaggregation algorithm (IAD) [31], to compute the vector π that satisfies Eq. (2). Since each block of a lumpable partitioning has equal row sums, the lumped matrix does not change from iteration to iteration. Hence, compared with the original IAD algorithm, the aggregation phase of Algorithm 1 is performed once and each subsequent iteration consists only of disaggregation. The implementation of AID for discrete-time SANs can be found in [22].

In contrast to Algorithm 1, the existing aggregation–disaggregation algorithm discussed in [4] utilizes a different approach in which aggregation at each iteration is done with respect to the states of an automaton chosen adaptively. We also remark that in the experiments of [4] the disaggregation phase of the algorithm is a power iteration, which is inferior to BGS since BGS is a preconditioned power iteration in which the preconditioning matrix is the block lower-triangular part of the coefficient matrix. Recent results [14] on the computation of the stationary vector of Markov chains show that IAD and BGS with judiciously chosen partitionings mostly outperform incomplete LU (ILU) preconditioned projection methods. Furthermore, BGS, which forms the disaggregation phase of IAD, when used with partitionings having blocks of equal order is likely to outperform IAD when the problem at hand is not ill-conditioned. Therefore, we propose AID rather than BGS for SANs when possible since the partitioning in (4) is a balanced one with equal order of blocks and the aggregate matrix needs to be formed once due to lumpability. In this way, we not only use a balanced partitioning but we also incorporate to our algorithm the gain obtained from being able to exactly solve the coupling matrix in IAD (i.e., lumped matrix).

Algorithm 1. *AID algorithm for lumpable continuous-time SANs*

1. Let $\pi^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)}, \ldots, \pi_K^{(0)})$ be a given initial approximation of π. Set it $= 1$.
2. Aggregation:
   (a) Compute the lumped matrix L of order K with ijth element $l_{ij} = e_1^T(Q_{ij}u)$.
   (b) Solve the singular system $\tau L = 0$ subject to $\sum_{i=1}^{K} \tau_i = 1$ for $\tau = (\tau_1, \tau_2, \ldots, \tau_K)$.
3. Disaggregation:
   (a) Compute the row vector
   $$z^{(\mathrm{it})} = \left(\tau_1 \frac{\pi_1^{(\mathrm{it}-1)}}{\|\pi_1^{(\mathrm{it}-1)}\|_1}, \tau_2 \frac{\pi_2^{(\mathrm{it}-1)}}{\|\pi_2^{(\mathrm{it}-1)}\|_1}, \ldots, \tau_K \frac{\pi_K^{(\mathrm{it}-1)}}{\|\pi_K^{(\mathrm{it}-1)}\|_1} \right).$$
   (b) Solve the K nonsingular systems of which the ith is given by
   $$\pi_i^{(\mathrm{it})} Q_{ii} = b_i^{(\mathrm{it})}$$
   for $\pi_i^{(\mathrm{it})}$, $i = 1, 2, \ldots, K$, where
   $$b_i^{(\mathrm{it})} = -\left(\sum_{j>i} z_j^{(\mathrm{it})} Q_{ji} + \sum_{j<i} \pi_j^{(\mathrm{it})} Q_{ji} \right).$$
4. Test $\pi^{(\mathrm{it})}$ for convergence. If the desired accuracy is attained, then stop and take $\pi^{(\mathrm{it})}$ as the steady-state vector of Q. Else set it $=$ it $+ 1$ and go to step 3.

Assuming that the automata are renumbered so that their indices in the given quasi-proper

ordering are ascending, the lumped matrix L is of order $K = \prod_{k=1}^{m-1} n_k$, where m is the smallest index of the automata in $\bigcup_{p=s}^{S} v_p^{\mathrm{SCC}}$ (see Theorem 4). In Algorithm 1, L is computed at the outset and solved once for its steady-state vector τ. In step 2(a) of the algorithm, the elements of the vector $Q_{ij}u$ are all equal; hence, we choose its first element. See the product $e_1^T(Q_{ij}u)$, where e_1 is the column vector of length $\prod_{k=m}^{N-1} n_k$ in which the first element is equal to 1 and the other elements are zero. As for the disaggregation phase (i.e., a BGS iteration), we need to look into how the right-hand sides $b_i^{(\mathrm{it})}$ at iteration it are computed. First, observe that the computation of $b_i^{(\mathrm{it})}$ involves only the off-diagonal blocks Q_{ij}, $i \neq j$. Hence, $\overline{Q}_e$ is omitted from the computation of $b_i^{(\mathrm{it})}$. Second, assuming that $(Q_e)_{ij}$ is the ijth block in the partitioning of Q_e as in Eq. (4), we have $(Q_e)_{ij} = \sum_{e=1}^{E} \xi_{ij}^{(e)} T_i^{(e)}$, where $\xi_{ij}^{(e)} = \prod_{k=1}^{m-1} Q_e^{(k)}(rQ_e^{(k)}, cQ_e^{(k)})$, $i \leftrightarrow (rQ_e^{(1)}, rQ_e^{(2)}, \ldots, rQ_e^{(m-1)})$, $j \leftrightarrow (cQ_e^{(1)}, cQ_e^{(2)}, \ldots, cQ_e^{(m-1)})$, and $T_i^{(e)} = \otimes_{k=m}^{N} Q_e^{(k)}$ (cf. Eq. (5)). Third, we have

$$
\begin{aligned}
b_i^{(\mathrm{it})} &= -\left(\sum_{j>i} z_j^{(\mathrm{it})} \left(\sum_{e=1}^{E} \xi_{ji}^{(e)} T_j^{(e)} \right) + \sum_{j<i} \pi_j^{(\mathrm{it})} \left(\sum_{e=1}^{E} \xi_{ji}^{(e)} T_j^{(e)} \right) \right) \\
&= -\left(\sum_{j>i} \sum_{e=1}^{E} \xi_{ji}^{(e)} \left(z_j^{(\mathrm{it})} T_j^{(e)} \right) + \sum_{j<i} \sum_{e=1}^{E} \xi_{ji}^{(e)} \left(\pi_j^{(\mathrm{it})} T_j^{(e)} \right) \right)
\end{aligned}
$$

for $i = 1, 2, \ldots, K$. Since $T_j^{(e)}$ is composed of $(N - m)$ tensor products, the vector–matrix multiplications $z_j^{(\mathrm{it})} T_j^{(e)}$ and $\pi_j^{(\mathrm{it})} T_j^{(e)}$ turn out to be expensive operations. Furthermore, they are performed a total of $K(K-1)E$ times during each iteration and constitute the bottleneck of the iterative solver. This situation can be improved at the cost of extra storage. Note that the subvectors $z_j^{(\mathrm{it})} T_j^{(e)}$ and $\pi_j^{(\mathrm{it})} T_j^{(e)}$ in the two summations appear in the computation of multiple $b_i^{(\mathrm{it})}$. Therefore, at iteration it, these subvectors of length $\prod_{k=m}^{N} n_k$ can be computed and stored when they are encountered for the first time for a specific pair of j and e, and then they can be scaled by $\xi_{ji}^{(e)}$ whenever necessary. Thus, when solving for $\pi^{(\mathrm{it})}$ in step 3(b) of Algorithm 1, we first compute $\xi_{ji}^{(e)}$, check if it is nonzero, and only then multiply $z_j^{(\mathrm{it})}$ or $\pi_j^{(\mathrm{it})}$ with $T_j^{(e)}$ if this product was not computed before. With such an implementation, no more than one multiplication of $z_j^{(\mathrm{it})}$ or $\pi_j^{(\mathrm{it})}$ with $T_j^{(e)}$ is performed.

In summary, the proposed solver is limited by $\max(K^2, (E+2)n)$ amount of double precision storage assuming that the lumped matrix is stored in two dimensions. The two vectors of length n are used to store the previous and current approximations of the solution.

5. Numerical experiments

We implemented Algorithm 1 in C++ as part of the software package PEPS [28]. We timed the solver on a Pentium III with 128 MB of RAM under Linux. In all experiments we use a stopping tolerance of 10^{-8} on the norm of the difference between consecutive approximations. We compare the running time of Algorithm 1, which we name as AID, with BGS. We use the recursive implementation of BGS for SANs as discussed in [34]. In order to provide a fair comparison, AID and BGS both use the same ordering of automata and partitioning of the generator. Furthermore, the implementations of both solvers use the same routines to generate and solve the diagonal blocks of the partitioning. The timing results are all in seconds.

We first consider the mass storage problem presented in Section 3. We use its four instances in [34] that we number from 1 to 4. The integer parameters of these four problems are given in Table 2. Parameter C denotes the number of states in $\mathscr{A}^{(4)}$, N_i denotes the number of states in $\mathscr{A}^{(i)}$, $i = 1, 2, 3$, and $\mathscr{A}^{(5)}$ has five states. Columns n and nz respectively correspond to the number of states and nonzeros in the generator underlying the SAN. Generators of the mass storage problem are irreducible.

In the first set of experiments, the automata are ordered as $\mathscr{A}^{(5)}$, $\mathscr{A}^{(3)}$, $\mathscr{A}^{(1)}$, $\mathscr{A}^{(4)}$, $\mathscr{A}^{(2)}$. As we indicated in Section 3, there are four lumpable partitionings for this proper ordering of automata. We partition the automata as $\mathscr{A}^{(5)}$, $\mathscr{A}^{(3)}$,

Table 2
Integer parameters of the two SAN models

	Mass storage				Tree queues			
Prob	C	N_i	n	nz	C_i	n	n_r	nz_r
1	26	6	6480	39,960	20	160,000	84,000	486,800
2	51	11	73,205	479,160	25	390,625	203,125	1,185,625
3	76	16	327,680	2,191,360	30	810,000	418,500	2,454,300
4	101	21	972,405	6,575,310	35	1,500,625	771,750	4,541,075

$\mathscr{A}^{(1)}|\mathscr{A}^{(4)}$, $\mathscr{A}^{(2)}$ so that there are $5N_1N_3$ blocks of order CN_2. For this partitioning, the size of core memory was sufficient to store the LU factors of all diagonal blocks in each experiment. Hence, diagonal blocks are generated and factorized once. Then the computed LU factors are used at each iteration to solve the K nonsingular systems in step 3(b) of Algorithm 1. The results of these experiments are given in Table 3. Column it# gives the number of iterations performed till convergence, time gives the total time to solve the problem, *db*gen gives the time to generate and factorize diagonal blocks at the outset, *L*gen gives the time to generate the lumped matrix L, *L*solve gives the time to solve L, and perit gives the time to perform one iteration of the corresponding solver. The values in column perit are calculated as (time$-$(*L*gen + *L*solve + *db*gen))/(it#). Note that for BGS, columns *L*gen and *L*solve are naturally zero. In the first problem, L is stored as a two-dimensional matrix and solved using the Grassmann–Taksar–Heyman (GTH) method (see [13,14] and the references therein). In the last three problems, L is of order 605, 1280 and 2205, respectively. Hence, it is more feasible to store L in sparse format and solve it using IAD with a balanced partitioning (if possible) having a small degree of coupling (see [13,14]). In all problems, the smallest degree of coupling for L is on the order of 10^{-2}. For this degree of coupling, the partitioning of L has five blocks of equal order. Even though step 3(a) of AID does not exist in BGS, the experiments with the particular ordering and partitioning of automata show that time per iteration in AID is smaller than that in BGS due to the gain obtained from computing the products $z_j^{(\mathrm{it})}Q_{ji}$ and $\pi_j^{(\mathrm{it})}Q_{ji}$ once at the expense of some storage space as discussed in Section 4. Furthermore, AID converges to the solution in a smaller number of iterations in all problems in agreement with expectations since it uses exact aggregation with a BGS disaggregation step. Hence, the solution time with AID is considerably smaller than that with BGS although there is extra work associated with forming and solving the aggregated system.

In the second set of experiments with the mass storage problem, the automata are ordered as $\mathscr{A}^{(5)}$, $\mathscr{A}^{(4)}$, $\mathscr{A}^{(1)}$, $\mathscr{A}^{(3)}$, $\mathscr{A}^{(2)}$. Observe that for this ordering, there are only 2 lumpable partitionings of the generator which are given by $\mathscr{A}^{(5)}$, $\mathscr{A}^{(4)}$, $\mathscr{A}^{(1)}$, $\mathscr{A}^{(3)}|\mathscr{A}^{(2)}$ and $\mathscr{A}^{(5)}|\mathscr{A}^{(4)}$, $\mathscr{A}^{(1)}$, $\mathscr{A}^{(3)}$, $\mathscr{A}^{(2)}$. Furthermore, the latter partitioning has blocks of

Table 3
Results of experiments with the mass storage problem, first ordering

Prob	Solver	it#	Time	*db*gen	*L*gen	*L*solve	Perit
1	BGS	102	2.59	0.04			0.03
	AID	34	1.00	0.04	0.03	0.00	0.03
2	BGS	106	44.79	0.68			0.42
	AID	40	13.03	0.68	0.02	0.00	0.31
3	BGS	201	417.98	8.53			2.03
	AID	47	75.01	8.53	0.06	0.12	1.41
4	BGS	323	1932.39	42.25			5.85
	AID	58	303.16	42.25	0.14	0.39	4.49

Table 4
Results of experiments with the mass storage problem, second ordering

Prob	Solver	it#	Time	*db*gen	*L*gen	*L*solve	Perit
1	BGS	33	1.28	0.04			0.04
	AID	8	0.46	0.04	0.03	0.03	0.05
2	BGS	25	14.88	0.35			0.58
	AID	7	6.94	0.35	1.14	0.76	0.67
3	BGS	23	70.63	1.49			3.01
	AID	7	42.74	1.49	10.34	5.85	3.58
4	BGS	30	293.39	4.48			9.63
	AID	7	236.25	4.48	129.14	27.10	10.79

order $n/5$ and is unfavorable due to the relatively large order of blocks for large n. Thus, we present the results of the second set of experiments in Table 4 using the former partitioning which has $5CN_1N_3$ blocks of order N_2. As in the first set of experiments, the diagonal blocks are generated and factorized once and the LU factors are stored in core memory. The lumped matrices of the four problems are of order 1080, 6655, 20,480, and 46,305, respectively. Therefore, in all problems we solve the lumped matrix using sparse IAD and employ the same kind of partitionings as in the last three problems of the first set of experiments. In step 3(b) of Algorithm 1, we use the optimized recursive BGS implementation discussed in [34] rather than the implementation described in Section 4, since the blocks are relatively small in the partitioning under consideration. In other words, the same routine is used in BGS and in the disaggregation step of AID. Together with the fact that there is overhead associated with step 3(a) of Algorithm 1, this implies slightly larger time per iteration in AID than in BGS. Observe that both solvers converge in a smaller number of iterations when compared with the results of the first set of experiments. Nevertheless, it is not surprising to see that AID still converges in a smaller number of iterations than BGS. We also remark that in the last problem, the time to generate the lumped matrix takes more than half the time to solve the problem. Hence, a very unbalanced partitioning with small order of blocks and a large lumped matrix seems to be unfavorable for large problems.

The second problem we use to test Algorithm 1 is the three queues problem that appears in [16]. This problem is an open queueing network of three finite capacity queues in which customers from queue 1 (type 1 customers) and queue 2 (type 2 customers) try to join queue 3. In the original model discussed in [16], when customers of type 1 find queue 3 full, they are blocked, whereas in the same situation customers of type 2 are lost. Here, we consider a modified version of this model in which customers of both types are lost when queue 3 is full. The network is modeled using 4 automata and 2 synchronizing events. $\mathscr{A}^{(1)}$ and $\mathscr{A}^{(2)}$ model the number of customers in queues 1 and 2, respectively. $\mathscr{A}^{(3)}$ and $\mathscr{A}^{(4)}$ model the number of type 1 and type 2 customers in queue 3, respectively. $\mathscr{A}^{(i)}$, $i = 1, 2, 3$, has C_i states and $\mathscr{A}^{(4)}$ has C_3 states. The generator underlying the SAN model of the three queues problem has a single subset of $C_1C_2C_3(C_3 + 1)/2$ essential states whereas the global state space size is $C_1C_2C_3^2$. In our experiments, we use the real valued parameters in [34]. We use four instances of the three queues problem and number them from 1 to 4. The integer parameters are given in Table 2. We set $C_1 = C_2 = C_3$ with values given in column C_i. Since the generator has transient states, we first run the state classification (SC) algorithm discussed in [20] to classify the states into recurrent and transient subsets. Columns n_r and nz_r respectively give the number of recurrent states and the number of nonzero elements in the corresponding submatrix of the generator. Alternatively, when the performance analyst has information about the particular SAN model under consideration, it may be possible to define on the global state space a reachability function that returns 1 for recurrent states and 0 for transient states, thereby enabling the identification of the subset of recurrent states

Table 5
Results of experiments with the three queues problem

C_i	Solver	it#	Time	SC	*db*gen	*L*gen	*L*solve	Perit
20	BGS	341	189.82	1.31	6.59			0.53
	AID	180	75.08	1.31	6.59	0.02	0.10	0.37
25	BGS	404	585.20	3.22	24.89			1.39
	AID	201	209.43	3.22	24.89	0.03	0.25	0.90
30	BGS	456	1312.05	6.78	72.70			2.70
	AID	221	483.73	6.78	72.70	0.06	0.65	1.85
35	BGS	502	2864.87	12.70	185.75			5.31
	AID	241	1017.49	12.70	185.75	0.09	1.21	3.39

in advance. See [18] for example SAN models and their reachability functions. In any case, once the recurrent subset of states is identified, the elements in $\pi^{(0)}$ corresponding to transient states are set to zero and omitted from further consideration when running Algorithm 1. See also [7] for various vector–tensor product multiplication algorithms that eliminate transient states from consideration and operate only on the recurrent subset of states.

The SAN model of the three queues problem is in its explicit form. There are functional transitions in synchronizing transition probability matrices of $\mathscr{A}^{(3)}$ and $\mathscr{A}^{(4)}$. Functional transitions of $\mathscr{A}^{(3)}$ depend on the state of $\mathscr{A}^{(4)}$ and those in $\mathscr{A}^{(4)}$ depend on the state of $\mathscr{A}^{(3)}$ implying a cyclic dependency. Hence, a proper ordering of the automata in this SAN does not exist. We consider the quasi-proper ordering $\mathscr{A}^{(1)}$, $\mathscr{A}^{(2)}$, $\mathscr{A}^{(3)}$, $\mathscr{A}^{(4)}$, which has two lumpable partitionings given by $\mathscr{A}^{(1)}$, $\mathscr{A}^{(2)}|\mathscr{A}^{(3)}$, $\mathscr{A}^{(4)}$ and $\mathscr{A}^{(1)}|\mathscr{A}^{(2)}$, $\mathscr{A}^{(3)}$, $\mathscr{A}^{(4)}$. We remark that in the original SAN model of the three queues problem, there exists a single lumpable partitioning having C_2 blocks of order $C_1C_3^2$. Here we experiment with the partitioning $\mathscr{A}^{(1)}$, $\mathscr{A}^{(2)}|\mathscr{A}^{(3)}$, $\mathscr{A}^{(4)}$, which has C_1C_2 blocks of order C_3^2. In the four instances of the three queues problem we consider, the lumped matrices are irreducible and of order 400, 625, 900, and 1225, respectively. We solve the lumped matrices using sparse IAD with block partitionings having degree of coupling on the order of 10^{-1}. The results of these experiments are presented in Table 5. Time spent for state classification is negligible (see column SC). The values in column time include those in SC. Numerical results show that AID converges in a smaller number of iterations than BGS. Furthermore, time per iteration in AID is smaller than that in BGS again due to the balanced nature of the partitioning. Finally, solution time with AID is less than half of that with BGS in all experiments.

6. Conclusion

In this work, easy to check conditions are given for a class of lumpable partitionings of the generator underlying a continuous-time SAN model with functional dependencies. For lumpable SANs, an efficient aggregation–iterative disaggregation algorithm that uses exact aggregation with a BGS disaggregation step is presented. Extensive experiments with two continuous-time SAN models with functional dependencies show that the proposed solver performs much better than the highly competitive BGS for the same ordering and partitioning of automata. It is also observed that some orderings and partitionings of automata lead to faster convergence than others. Future work may focus on trying to identify them.

Acknowledgements

We thank the anonymous referees for their detailed remarks and suggestions, which led to an improved manuscript.

References

[1] P. Buchholz, Exact and ordinary lumpability in finite Markov chains, Journal of Applied Probability 31 (1994) 59–75.

[2] P. Buchholz, Hierarchical Markovian models: Symmetries and reduction, Performance Evaluation 22 (1995) 93–110.
[3] P. Buchholz, Equivalence relations for stochastic automata networks, in: W.J. Stewart (Ed.), Computations with Markov Chains, Kluwer, Boston, MA, 1995, pp. 197–215.
[4] P. Buchholz, An aggregation\disaggregation algorithm for stochastic automata networks, Probability in the Engineering and Informational Sciences 11 (1997) 229–253.
[5] P. Buchholz, Projection methods for the analysis of stochastic automata networks, in: B. Plateau, W.J. Stewart, M. Silva, (Eds.), Proceedings of the 3rd International Workshop on the Numerical Solution of Markov Chains, Prensas Universitarias de Zaragoza, Spain, 1999, pp. 149–168.
[6] P. Buchholz, Exact performance equivalence: An equivalence relation for stochastic automata, Theoretical Computer Science 215 (1999) 263–287.
[7] P. Buchholz, G. Ciardo, S. Donatelli, P. Kemper, Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models, INFORMS Journal on Computing 12 (2000) 203–222.
[8] P. Buchholz, Multilevel solutions for structured Markov chains, SIAM Journal on Matrix Analysis and Applications 22 (2000) 342–357.
[9] R.H. Chan, W.K. Ching, Circulant preconditioners for stochastic automata networks, Numerische Mathematik 87 (2000) 35–57.
[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, The MIT Press, Cambridge, MA, 1990.
[11] M. Davio, Kronecker products and shuffle algebra, IEEE Transactions on Computers C-30 (1981) 116–125.
[12] T. Dayar, O.I. Pentakalos, A.B. Stephens, Analytical modeling of robotic tape libraries using stochastic automata, Technical Report TR-97-189, CESDIS, NASA/GSFC, Greenbelt, Maryland, January 1997.
[13] T. Dayar, W.J. Stewart, On the effects of using the Grassmann–Taksar–Heyman method in iterative aggregation–disaggregation, SIAM Journal on Scientific Computing 17 (1996) 287–303.
[14] T. Dayar, W.J. Stewart, Comparison of partitioning techniques for two-level iterative solvers on large, sparse Markov chains, SIAM Journal on Scientific Computing 21 (2000) 1691–1705.
[15] S. Donatelli, Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space, Performance Evaluation 18 (1993) 21–36.
[16] P. Fernandes, B. Plateau, W.J. Stewart, Efficient descriptor-vector multiplications in stochastic automata networks, Journal of the ACM 45 (1998) 381–414.
[17] P. Fernandes, B. Plateau, W.J. Stewart, Optimizing tensor product computations in stochastic automata networks, RAIRO, Operations Research 32 (3) (1998) 325–351.
[18] P. Fernandes, R.J. Hessel, B. Plateau, W.J. Stewart, PEPS-2000 user manual, June 2000; available online at http://www-apache.imag.fr/software/peps/PEPSman2000.ps.gz.
[19] J.-M. Fourneau, F. Quessette, Graphs and stochastic automata networks, in: W.J. Stewart (Ed.), Computations with Markov Chains, Kluwer, Boston, MA, 1995, pp. 217–235.
[20] O. Gusak, T. Dayar, J.-M. Fourneau, Stochastic automata networks and near complete decomposability, Technical Report BU-CE-0016, Department of Computer Engineering, Bilkent University, Ankara, Turkey, October 2000; available online at ftp://ftp.cs.bilkent.edu.tr/pub/tech-reports/2000/BU-CE-0016.ps.z.
[21] O. Gusak, T. Dayar, J.-M. Fourneau, Stochastic automata networks and near complete decomposability, SIAM Journal on Matrix Analysis and Applications 23 (2001) 581–599.
[22] O. Gusak, T. Dayar, J.-M. Fourneau, Iterative disaggregation for a class of lumpable discrete-time stochastic automata networks, Performance Evaluation (submitted for publication).
[23] J.R. Kemeny, J.L. Snell, Finite Markov Chains, Van Nostrand, New York, 1960.
[26] B. Plateau, On the stochastic structure of parallelism and synchronization models for distributed algorithms, in: Proceedings of the ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems, Austin, Texas, 1985, pp. 147–154.
[27] B. Plateau, K. Atif, Stochastic automata network for modeling parallel systems, IEEE Transactions on Software Engineering 17 (1991) 1093–1108.
[28] B. Plateau, J.-M. Fourneau, K.-H. Lee, PEPS: A package for solving complex Markov models of parallel systems, in: R. Puigjaner, D. Ptier, (Eds.), Modeling Techniques and Tools for Computer Performance Evaluation, Spain, 1988, pp. 291–305.
[29] B. Plateau, J.-M. Fourneau, A methodology for solving Markov models of parallel systems, Journal of Parallel and Distributed Computing 12 (1991) 370–387.
[30] M. Siegle, Structured Markovian performance modeling with automatic symmetry exploitation, in: Short Papers and Tool Descriptions of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Vienna, Austria, 1994, pp. 77–81.
[31] G.W. Stewart, W.J. Stewart, D.F. McAllister, A two-stage iteration for solving nearly completely decomposable Markov chains, in: G.H. Golub, A. Greenbaum, M. Luskin, (Eds.), Recent Advances in Iterative Methods, IMA Vol. Math. Appl. 60, Springer-Verlag, New York, 1994, pp. 201–216.
[32] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, Princeton, NJ, 1994.
[33] W.J. Stewart, K. Atif, B. Plateau, The numerical solution of stochastic automata networks, European Journal of Operational Research 86 (1995) 503–525.
[34] E. Uysal, T. Dayar, Iterative methods based on splittings for stochastic automata networks, European Journal of Operational Research 110 (1998) 166–186.