ANALYSIS OF LARGE MARKOV CHAINS USING STOCHASTIC AUTOMATA NETWORKS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Oleg Gusak

July, 2001

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Asst. Prof. Dr. Murat Alanyalı

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Tuğrul Dayar (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Asst. Prof. Dr. Ezhan Karaşan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Sibel Tarı

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

ABSTRACT

ANALYSIS OF LARGE MARKOV CHAINS
USING STOCHASTIC AUTOMATA NETWORKS

Oleg Gusak
Ph.D. in Computer Engineering
Advisor: Assoc. Prof. Tuğrul Dayar
July, 2001

This work contributes to the existing research in the area of analysis of finite Markov chains (MCs) modeled as stochastic automata networks (SANs). First, this thesis extends the near complete decomposability concept of Markov chains to SANs so that the inherent difficulty associated with solving the underlying MC can be forecasted and solution techniques based on this concept can be investigated. A straightforward approach to finding a nearly completely decomposable (NCD) partitioning of the MC underlying a SAN requires the computation of the nonzero elements of its global generator. This is not feasible for very large systems even in sparse matrix representation due to memory and execution time constraints. In this thesis, an efficient decompositional solution algorithm to this problem that is based on analyzing the NCD structure of each component of a given SAN is introduced. Numerical results show that the given algorithm performs much better than the straightforward approach.

Second, this work specifies easy to check lumpability conditions for the generator of a SAN. When there exists a lumpable partitioning induced by the tensor representation of the generator, it is shown that an efficient iterative aggregation-disaggregation algorithm (IAD) may be employed to compute the steady state distribution of the MC underlying the SAN model. The results of experiments with continuous-time and discrete-time SAN models show that the proposed algorithm performs better than the highly competitive block Gauss-Seidel (BGS) in terms of both the number of iterations and the time to converge to the solution.

Finally, the performance of the IAD algorithm on continuous-time SANs having relatively large blocks in lumpable partitionings is investigated. To overcome difficulties associated with solving large diagonal blocks at each iteration of the IAD algorithm, the recursive implementation of BGS for SANs is employed. The performance of IAD is compared with that of BGS. The results of experiments show that it is possible to tune IAD so that it outperforms BGS.

Key words: Markov chain, Stochastic automata network, Near complete decomposability, lumpability, iterative aggregation-disaggregation.

ÖZET

BÜYÜK MARKOV ZİNCİRLERİNİN RASSAL
ÖZDEVİNİMLİ AĞ KULLANILARAK ÇÖZÜMLEMESİ

Oleg Gusak
Bilgisayar Mühendisliği, Doktora
Tez Yöneticisi: Doç. Dr. Tuğrul Dayar
Temmuz, 2001

Bu çalışma, rassal özdevinimli ağ olarak modellenen sonlu Markov zincirlerinin çözümlemesi alanında var olan araştırmaya katkıda bulunmaktadır. İlk olarak, bu tez Markov zincirlerinin neredeyse tamamen bölünebilirlik kavramını rassal özdevinimli ağlara genişleterek, alttaki Markov zincirinin çözülmesinde aslında var olan zorluğun kestirilmesini ve bu kavrama dayalı çözüm tekniklerinin araştırılmasını mümkün kılmaktadır. Bir rassal özdevinimli ağın altındaki Markov zincirinin neredeyse tamamen bölünebilir bloklara ayrışmış halini bulmanın basit yolu, sisteme karşı gelen matrisin sıfırdan farklı elemanlarının hesap edilmesini gerektirir. Bu, çok büyük sistemler için bellek ve uygulama zamanı sınırlamalarından dolayı seyrek matris gosterimiyle dahi mümkün değildir. Bu tezde, bu problem için, verilen bir rassal özdevinimli ağın herbir bileşenini çözümlemeye dayalı, bir etkili ayrıştırmalı çözüm algoritması sunulmaktadır. Sayısal sonuçlar, verilen algoritmanın basit yöntemden çok daha iyi randıman verdiğini göstermektedir.

İkinci olarak, bu çalışma bir rassal özdevinimli ağa karşı gelen matris için kontrol edilmesi kolay birleştirilebilirlik koşulları vermektedir. Sisteme karşı gelen matrisin tensör gosteriminin neden olduğu birleştirilebilir bir bölünme var olduğunda, rassal özdevinimli ağ modelinin altındaki Markov zincirinin değişmez durum dağılımının etkili bir dolaylı birleştirme-ayrıştırma algoritması kullanılarak bulunabileceği gösterilmektedır. Sürekli-zamanlı ve kesintili-zamanlı rassal özdevinimli ağ modelleri üzerinde yapılan deneylerin sonuçları, önerilen algoritmanın son derece çetin bir rakip olan blok Gauss-Seidel'den hem ardışık tekrar sayısında hem de çözüme yakınsama için geçen süre yönünden daha iyi randıman verdiğini göstermektedir.

Son olarak, dolaylı birleştirme-ayrıştırma algoritmasının başarımı, birleştirilebilir bölünmelerde nispi olarak büyük bloklara sahip sürekli-zamanlı rassal özdevinimli ağlar üzerinde araştırılmaktadır. Dolaylı birleştirme-ayrıştırma algoritmasının herbir ardışık tekrarında, büyük blokları çözmede karşılaşılan güçlükleri aşmak için rassal özdevinimli ağlar için blok Gauss-Seidel'ın özyinelemeli uygulaması kullanılmaktadır. Dolaylı birleştirme-ayrıştırma algoritmasının başarımı blok Gauss-Seidel'inki ile karşılaştırılmaktadır. Deney sonuçları, dolaylı birleştirme-ayrıştırmayı blok Gauss-Seidel'i geçecek şekilde ayarlamanın mümkün olduğunu göstermektedir.

Anahtar kelimeler: Markov zinciri, rassal özdevinimli ağlar, neredeyse tamamen bölünebilirlik, birleştirilebilirlik, dolaylı birleştirme-ayrıştırma.

ACKNOWLEDGMENTS

I would like to express my sincere thanks and gratitude to my advisor Tuğrul Dayar for his enormous efforts and time spent on contributing to my knowledge and for his invaluable guidance in my research.

Part of the work presented in this thesis was accomplished through a joint research project with Laboratorie PRiSM at the University of Versailles-St. Quentin, France. I would like to thank Jean-Michel Fourneau for his contribution, comments and suggestions on this work. Also, I would like to thank Franck Quessette for his support and help during my stay at Versailles and for his comments and suggestions in our conversations on SAN formalism.

I would like to thank Brigitte Plateau for inviting me to Laboratorie ID-IMAG in Grenoble during my second visit to France in year 2000 to present the results of this work and for our discussions on this research that gave me a much better grasp of the investigated problem.

I would like to thank the members of my Ph.D. advisory committee Murat Alanyalı and Ozgür Ulusoy for their valuable suggestions and guidance in my Ph.D. study. I also thank them and the members of my Ph.D. jury Ezhan Karaşan and Sibel Tarı for their careful reading and comments on this thesis that led to an improved manuscript.

I am very grateful to my parents for understanding of my desire to pursue a doctoral degree and their constant support and encouragement that was the major source for my inspiration to conduct this research.

I thank Bilkent University and the Department of Computer Engineering for full financial support (tuition payment and teaching assistantship) granted to me for the whole course of my study and for financial support of my conference visits at which the results of this work were presented. I am also thankful to Bilkent University for maintaining high level education standards and providing high quality research and living facilities that made my work more efficient.

Finally, my infinite thanks go to my friends and other people whom I met while studying at Bilkent for their help, encouragement and support. I would like to thank the people of Turkey for their hospitality and help which has made this a stay to remember.

To my parents Galina Todorova and Yurij Gusak.

To the memory of my grandparents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In most cases, a system under investigation can be described in terms of states and transitions among them. If the time spent in any state of the system is exponentially distributed, the system can be modeled as a Markov process [12, 57]. If the state space of a Markov process is discrete, the Markov process is called a Markov chain (MC) [4, 29, 35, 40, 44, 62]. A continuous-time Markov chain (CTMC) is a continuous-time stochastic process $X(t)$ that satisfies the memoryless (Markov) property. In other words, for all integers n and for any sequence $t_0 < t_1 < \ldots < t_n < t$ the following relation holds

$$\begin{aligned}\text{Prob}\{X(t) \leq x | X(t_0) = x_0, X(t_1) = x_1, \ldots, X(t_n) = x_n\} \\ = \text{Prob}\{X(t) \leq x | X(t_n) = x_n\},\end{aligned}$$

where x_k, $k = 0, 1, \ldots, n$, denotes the state of the Markov process at time t_k.

For a discrete-time Markov chain (DTMC), the state of the system is observed at a discrete set of times. Hence, the time parameters t_k can be omitted. As in the continuous-time case, the conditional probability p_{ij} of making a transition to state $x_{n+1} = j$ from state $x_n = i$ is independent of the states $x_0, x_1, \ldots, x_{n-1}$. The conditional probabilities p_{ij} are called single-step (or one-step) transition probabilities [4, 62]. If p_{ij} are independent of n, the corresponding DTMC is said to be homogeneous. In this work, we consider homogeneous DTMCs and CTMCs having finite state space.

A finite DTMC is described by the one-step transition probability matrix

P in which the ijth entry is equal to p_{ij} and $\sum_j p_{ij} = 1$ for all i. Similar to a DTMC, a CTMC is represented by the transition rate matrix Q in which q_{ij} corresponds to the number of transitions from state i to state j per unit time and $\sum_j q_{ij} = 0$ for all i.

When transitions between states of a MC are known, the steady state analysis of the underlying system amounts to computing the steady state probability vector π of the MC, where π_i is the limiting probability of finding the MC in state i in the long run. Assuming that the DTMC has a single irreducible subset of states and is aperiodic (see for instance [44, 62] for classification of states), π can be obtained from the system of linear equations

$$\pi P = \pi, \; ||\pi||_1 = 1. \tag{1.1}$$

Hence, π is also a stationary distribution. When the system is modeled as a CTMC, π can be obtained from the system of linear equations

$$\pi Q = 0, \; ||\pi||_1 = 1. \tag{1.2}$$

Introduced by A.A. Markov in 1907, over the years Markovian modeling has become a well established theory. Nowadays, MCs are used for the analysis of stochastic systems arising in different application areas, and in particular, for modeling computer and communication systems.

Complexity of the current applications in communications and computer systems is constantly increasing. Consequently, the complexity and the size of the Markov chains employed in the analysis of these systems is an obstacle. In the next section, we review basic methods that are used for the analysis of large MCs.

1.1 Methods for the analysis of large Markov chains

Various techniques have been developed for the analysis of large MCs. For a large Markov chain having special structural properties, it may be possible to avoid the generation of the underlying state space. For instance, for product form queueing networks [3, 31, 37], stationary probabilities can be obtained as closed-form expressions. Another class of MCs having special structure lend themselves to matrix-geometric solution methods [45, 46].

Assuming that a large MC does not fall into the class of MCs having special structural properties, two major difficulties need to be overcome during the analysis of the underlying system. First, difficulties arise at the modeling stage when a large system must be described in terms of states and transitions among them. Difficulties of the second type arise when computing the stationary probability vector of the underlying Markov chain.

To overcome difficulties when modeling large Markovian systems, the computational low-level model that is the transition matrix of a MC should be separated from the high-level model description [4, 35]. This must be done due to the following reasons. First, the high-level description gives a more clear understanding of the modeled system to the researcher. Second, compactness of the high-level description can be utilized at the analysis step by storing in core memory larger MCs than can be stored with a low-level description. Third, a compact description avoids the generation of the low-level model and errors associated with this process. Finally, a compact high-level description may reveal properties of the MC that lead to a more efficient analysis. It should be emphasized that high-level descriptions are usually applicable in MCs having more or less repetitive (regular) structure. Otherwise, utilization of a high-level formalism is not useful and brings only additional overhead to the analysis of the model.

One of the high-level descriptions that is widely used for the generation of large MCs is the generalized stochastic Petri net (GSPN) formalism [1,2,48,56]. It was originally introduced by C.A. Petri in 1962 in [49]. A Petri net (PN) is a directed bipartite graph whose nodes are places and transitions. Arcs of the graph are divided into two classes. Input arcs lead from input places to transitions. Output arcs lead from transitions to output places. A finite number of tokens is associated with a PN. Tokens are distributed among places and each place may contain an arbitrary number of tokens. A marking or a state of a PN is a possible assignment of tokens to the places of the PN.

A transition of a PN is enabled and can be fired if each of its input places contains at least one token. When a transition is fired, tokens are moved from input places to output places according to arcs associated with the transition. Change in the distribution of tokens among the places of a PN after firing a transition corresponds to a state change in the PN.

In a GSPN, each transition has an associated firing time. The firing time may be zero or exponentially distributed. Transitions that have zero firing time can be fired immediately when they are enabled, whereas an enabled transition with a nonzero firing time is triggered according to the exponential distribution having the parameter associated with the transition. Recent results [11, p. 79] show that hierarchical representations arising in queueing network and superposed stochastic Petri Net formalisms [7,10,11,14,21,39] lend themselves naturally to distributed steady state analysis.

Another modeling paradigm called Stochastic Automata Networks (SANs) [6,8,13,18,23–25,27,28,50–55,62,63,67] can be employed to model large finite MCs. SANs provide a methodology for modeling large systems with interacting components. The main idea is to decompose the system of interest into its components and to model each component separately. Once this is done, interactions and dependencies among components can be brought into the picture and the model finalized. With this decompositional approach, the global system ends up having as many states as the product of the number of states of the individual components. The benefit of the SAN approach is twofold. First, each component can be modeled much easier compared to the global system due to state space reduction. Second, space required to store the description of components is minimal compared to the case in which transitions from each global state are stored explicitly. However, all this happens at the expense of increased analysis time [6,8,13,18,24,28,63,67].

Similar to the SAN formalism, the multilevel decomposition introduced in [9] uses tensor products to represent the transition matrix of a MC. In this approach, the transition matrix is partitioned into blocks, and each block of the partitioning is represented by a sum of tensor products.

Somewhat distant from the methods described so far are stochastic bounding techniques. With these methods, lower and upper bounds on the stationary probability vector of the original MC or parts of it are computed [60, 65]. Generally, the approach is coupled with some kind of state space reduction technique so that the measure of interest can be obtained much easier than using the original MC (see for instance [47]).

There are other techniques that can be used to cope with the state space explosion problem in large MCs. A detailed review of such techniques can be found in [36]. In this work, we concentrate on the numerical solution of MCs modeled as SANs.

In the next section, we briefly mention existing analysis methods for SANs and discuss what can be done to improve them.

1.2 Analysis of MCs modeled as SANs

Different solution methods with applications to SANs have been studied [6, 8, 13, 18, 24, 28, 62, 63, 67]. The existence of an efficient vector-descriptor multiplication algorithm [24] hints at employing iterative methods for computing the stationary probability vector of the underlying MC. See chapter 3 in [62] for the description of iterative methods for MCs. In [67], iterative methods based on splittings (i.e., Jacobi, Gauss-Seidel, successive over-relaxation) and their block versions are introduced for SANs. Results with iterative aggregation-disaggregation (IAD) [19, 20, 41, 59, 61, 64] type solvers for SANs appear in [6].

An important issue in choosing an efficient iterative solver for SANs is the conditioning [43] associated with the underlying Markov chain. Recent numerical experiments [20] show that two-level iterative solvers perform very well with nearly completely decomposable (NCD) partitionings [17] having balanced block sizes when the MC to be solved for its stationary probability vector is ill-conditioned. Recall that a DTMC is said to be NCD if its one-step transition probability matrix can be symmetrically permuted to a block form in which all the off-diagonal blocks have relatively small elements compared to the elements in the diagonal blocks [62, p. 286].

It should be emphasized that iterative aggregation-disaggregation based on NCD partitionings has certain rate of convergence guarantees [61] that may be useful for very large MCs. On the other hand, standard iterative methods applied to an ill-conditioned MC converge slowly. Hence, information about NCDness of a MC can be used to suggest the appropriate iterative solver. Thus, one of the directions that is worth investigating is the NCD analysis of MCs modeled as SANs.

The compact SAN description can be also employed for studying structural properties of the underlying MC that may lead to more efficient analysis. For instance, if each block in a partitioning of the MC has constant row sums, i.e., the MC underlying the SAN model is lumpable [38], the analysis can be significantly simplified. Lumpability is especially important for discrete-time SANs, which are usually evaded by researchers because of the relatively high density of nonzeros in their transition probability matrices.

In the first part of this thesis, we extend the concept of near complete decomposability to SANs so that the inherent difficulty associated with solving the underlying MC can be forecasted and solution techniques based on this concept can be investigated. In doing this, we utilize the graph theoretical ideas for SANs given in [28].

In the second part, we specify lumpability conditions for SANs and introduce an efficient iterative aggregation-disaggregation (IAD) algorithm for lumpable SANs. We also model a wireless communication system as a discrete-time SAN, show that it is lumpable, and demonstrate how difficulties associated with the analysis of discrete-time SANs can be overcome using the introduced algorithm for lumpable SANs.

In the next section, we discuss the organization of the thesis and briefly describe the contents of each chapter.

1.3 Layout of the thesis

In the first part of the next chapter, we introduce the SAN paradigm, review its basic concepts, and give examples of continuous-time SANs. In its second part, we make some assumptions regarding the description of a continuous-time SAN model and show what can be done if the given model does not satisfy the assumptions.

In chapter 3, we design a discrete-time SAN model of a current application in wireless communications. We first introduce the descriptor of a discrete-time SAN. Then we show how the SAN model described in [68] can be improved and extended to a more general case by introducing an additional type of service to the system.

In chapter 4, we present a three step algorithm that finds an NCD partitioning of the MC underlying a SAN based on a user specified decomposability parameter without computing the global generator matrix. In doing this, we proceed step by step introducing definitions, stating propositions, making remarks, and illustrating with small examples the ideas on which our algorithm is based. We also provide a summary of the complexity analysis of the NCD partitioning algorithm and numerical results with the algorithm on three applications.

In chapter 5, we investigate block partitionings of a matrix that is a sum of tensor products. Using these properties, we derive lumpability conditions for discrete-time and continuous-time SANs and introduce an efficient IAD algorithm for lumpable SANs.

In chapter 6, we analyze the discrete-time SAN model of the wireless communication system using IAD for lumpable SANs and present the results of numerical experiments with IAD and Block-Gauss Seidel (BGS) [58] on various continuous-time SAN models. In the end of the chapter, we discuss the case of lumpable SANs having unfavorable partitionings and its implications on the IAD algorithm.

Chapter 7 contains concluding remarks. The appendix includes definitions and properties of tensor operators, the continuous-time SAN models used in the numerical experiments, the description of the state classification (SC) algorithm whose output is used by the NCD partitioning algorithm and the IAD algorithm for lumpable SANs, the NCD partitioning algorithm and its complexity analysis, and the IAD algorithm for lumpable discrete-time SANs.

Chapter 2

SAN formalism

In the first section of this chapter we introduce the SAN formalism, give definitions of its components and discuss their characteristics. The section ends with two examples of continuous-time SAN models. In the second section, we specify our assumptions on the description of a SAN, define the equivalence of two SAN descriptions and show how a SAN that does not satisfy the assumptions can be transformed to an equivalent SAN that meets the requirements.

2.1 Overview

In a SAN (see [62], 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 nonnegative real valued function that depends on the state of other automata. 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 i (denoted by $\mathcal{A}^{(i)}$) are modeled by local transition rate matrix $Q_l^{(i)}$. When the components of the system do not interact among each other, the underlying continuous-time MC corresponding to the global system can be obtained by the tensor sum of the local transition rate matrices of the automata. We refer to the tensor representation associated with the CTMC as the descriptor of the SAN.

Each of E synchronizing events introduces two types of matrices to the SAN description. These matrices are called the synchronizing event matrix and the diagonal corrector matrix. For automaton i and synchronizing event j, we have the synchronizing event matrix $Q_{e_j}^{(i)}$ and the diagonal corrector matrix $\bar{Q}_{e_j}^{(i)}$ both of order equal to that of the local transition rate matrix $Q_l^{(i)}$. The automaton that triggers a synchronizing event is called the master, the others that get affected are called slaves. Matrices associated with synchronizing events are either transition rate matrices (corresponding to master automata) or transition probability matrices (corresponding to slave automata). When they are rate matrices, each diagonal element in the diagonal corrector matrix is the negated sum of the off-diagonal elements in the corresponding synchronizing event matrix. When they are transition probability matrices, each diagonal element of the corrector matrix is the sum of the corresponding row elements in the synchronizing event matrix.

Synchronizing events introduce additional tensor products to the descriptor thereby complicating the SAN formalism. Since a tensor sum may be written as a sum of tensor products, it is possible to express the descriptor as a sum of Ordinary Tensor Products (OTPs) in the absence of functional transitions. When functional transitions are present, the descriptor is formed of Generalized Tensor Products (GTPs, see [24], for instance).

Summarizing the above, we can express the descriptor of a SAN as

$$Q = Q_l + Q_e + \bar{Q}_e, \tag{2.1}$$

where

$$Q_l = \bigoplus_{i=1}^{N} Q_l^{(i)}, \quad Q_e = \sum_{j=1}^{E} \bigotimes_{i=1}^{N} Q_{e_j}^{(i)}, \quad \bar{Q}_e = \sum_{j=1}^{E} \bigotimes_{i=1}^{N} \bar{Q}_{e_j}^{(i)},$$

$\otimes$ is the tensor product operator, $\oplus$ is the tensor sum operator (see appendix 8.1 for the definitions of the tensor operators). When there are functional transitions, tensor operations become generalized tensor operations.

Assuming that automaton i has n_i states, the global system has $n = \prod_{i=1}^{N} n_i$ states. The global state s of the system maps to the state vector $(s\mathcal{A}^{(1)}, s\mathcal{A}^{(2)}, \ldots, s\mathcal{A}^{(N)})$; that is, $s \leftrightarrow (s\mathcal{A}^{(1)}, s\mathcal{A}^{(2)}, \ldots, s\mathcal{A}^{(N)})$, where $s\mathcal{A}^{(k)}$ denotes the state of $\mathcal{A}^{(k)}$.

An important issue in the analysis of a SAN is the concept of functional dependency among the automata of the SAN.

Definition 2.1 *We denote by $\mathcal{A}^{(i)}[\mathcal{A}^{(k)}]$ a functional dependency between automata $\mathcal{A}^{(i)}$ and $\mathcal{A}^{(k)}$ when the value of at least one functional transition of $\mathcal{A}^{(i)}$ depends on the state of $\mathcal{A}^{(k)}$. We say, $\mathcal{A}^{(i)}$ functionally depends on $\mathcal{A}^{(k)}$. More generally, $\mathcal{A}^{(i)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(k)}]$ indicates that values of the functional transitions of $\mathcal{A}^{(i)}$ depend on $s\mathcal{A}^{(1)}, s\mathcal{A}^{(2)}, \ldots, s\mathcal{A}^{(k)}$.*

Before we define cyclic functional dependencies among automata of a SAN, we introduce the definition of the dependency graph of a SAN.

Definition 2.2 *The dependency graph $G(\mathcal{V}, \mathcal{E})$ of a SAN is the directed graph (digraph) associated with the automata $\mathcal{A}^{(i)}$, $i = 1, 2, \ldots, N$ of the SAN in which the vertex $v_i \in \mathcal{V}$ represents $\mathcal{A}^{(i)}$ and the edge $(v_i, v_k) \in \mathcal{E}$ if $\mathcal{A}^{(i)}[\mathcal{A}^{(k)}]$.*

Definition 2.3 *We say that there is a cyclic functional dependency among the automata of a SAN if a topological ordering of its dependency graph does not exist.*

Detailed description of the topological ordering algorithm for digraphs can be found in [15, pp. 485-487].

We illustrate the SAN formalism on two simple continuous-time SAN models that have respectively two and three automata. Observe that neither of the SAN models have cyclic functional dependencies.

Example 2.1 *Consider a SAN [62, pp. 470–472] that is formed of two automata (N=2) having 2 and 3 states and two synchronizing events (E=2). Automaton 1 is given by*

$$Q_l^{(1)} = \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & 0 \end{pmatrix},\ Q_{e_1}^{(1)} = \begin{pmatrix} 0 & 0 \\ \lambda_2 & 0 \end{pmatrix},\ \bar{Q}_{e_1}^{(1)} = \begin{pmatrix} 0 & 0 \\ 0 & -\lambda_2 \end{pmatrix},$$

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

and automaton 2 is given by

$$Q_l^{(2)} = \begin{pmatrix} -\mu_1 & \mu_1 & 0 \\ 0 & -\mu_2 & \mu_2 \\ 0 & 0 & 0 \end{pmatrix},\ Q_{e_1}^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},\ \bar{Q}_{e_1}^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

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

From equation (2.1), the descriptor of the SAN can be obtained as

$$\begin{aligned} Q &= Q_l + Q_e + \bar{Q}_e \\ &= \bigoplus_{i=1}^{2} Q_l^{(i)} + \sum_{j=1}^{2} \bigotimes_{i=1}^{2} Q_{e_j}^{(i)} + \sum_{j=1}^{2} \bigotimes_{i=1}^{2} \bar{Q}_{e_j}^{(i)} \\ &= Q_l^{(1)} \oplus Q_l^{(2)} + Q_{e_1}^{(1)} \otimes Q_{e_1}^{(2)} + Q_{e_2}^{(1)} \otimes Q_{e_2}^{(2)} + \bar{Q}_{e_1}^{(1)} \otimes \bar{Q}_{e_1}^{(2)} + \bar{Q}_{e_2}^{(1)} \otimes \bar{Q}_{e_2}^{(2)} = \end{aligned}$$

$$\begin{pmatrix} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\mu_2) & \mu_2 & 0 & \lambda_1 & 0 \\ \mu_3 & 0 & -(\lambda_1+\mu_3) & 0 & 0 & \lambda_1 \\ \lambda_2 & 0 & 0 & -(\lambda_2+\mu_1) & \mu_1 & 0 \\ \lambda_2 & 0 & 0 & 0 & -(\lambda_2+\mu_2) & \mu_2 \\ \lambda_2+\mu_3 & 0 & 0 & 0 & 0 & -(\lambda_2+\mu_3) \end{pmatrix}$$

Example 2.2 *In this example we consider a continuous-time SAN that consists of 3 automata ($N = 3$) and 2 synchronizing events ($E = 2$). The second automaton has three states and each of the other two automata has two states. The local transition rate matrix of the third automaton has the functional transition f. We have $f = 3$ when $\mathcal{A}^{(1)}$ is in state 1, and $f = 5$ when $\mathcal{A}^{(1)}$ is in state 2. The master of synchronizing event 1 is $\mathcal{A}^{(3)}$ and the master of synchronizing event 2 is $\mathcal{A}^{(2)}$. The matrices are*

$$Q_l^{(1)} = \begin{pmatrix} -2 & 2 \\ 1 & -1 \end{pmatrix}, Q_l^{(2)} = \begin{pmatrix} -2 & 2 & 0 \\ 2 & -5 & 3 \\ 1 & 3 & -4 \end{pmatrix}, Q_l^{(3)} = \begin{pmatrix} -f & f \\ 0 & 0 \end{pmatrix},$$

$$Q_{e_1}^{(1)} = \bar{Q}_{e_1}^{(1)} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, Q_{e_2}^{(1)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \bar{Q}_{e_2}^{(1)} = I,$$

$$Q_{e_1}^{(2)} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \bar{Q}_{e_1}^{(2)} = I, Q_{e_2}^{(2)} = \begin{pmatrix} 0 & 0 & 5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \bar{Q}_{e_2}^{(2)} = \begin{pmatrix} -5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$Q_{e_1}^{(3)} = \begin{pmatrix} 0 & 0 \\ 5 & 0 \end{pmatrix}, \bar{Q}_{e_1}^{(3)} = \begin{pmatrix} 0 & 0 \\ 0 & -5 \end{pmatrix}, Q_{e_2}^{(3)} = \bar{Q}_{e_2}^{(3)} = I.$$

The generator matrix of the underlying MC is given by

$$Q = \begin{pmatrix} -12 & 3 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 5 & 0 \\ 0 & -14 & 0 & 2 & 5 & 0 & 0 & 2 & 0 & 0 & 0 & 5 \\ 2 & 0 & -10 & 3 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 5 & 2 & 0 & -12 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 \\ 1 & 0 & 3 & 0 & -9 & 3 & 0 & 0 & 0 & 0 & 2 & 0 \\ 5 & 1 & 0 & 3 & 0 & -11 & 0 & 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 5 & 0 & -13 & 5 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 5 & 0 & -8 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & -11 & 5 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & -6 & 0 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 3 & 0 & -10 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 3 & 0 & -5 \end{pmatrix}.$$

2.2 Continuous-time SAN description

There is no standard specification for the description of a continuos-time SAN model. In this section, we state definitions and propositions that enable us to transform a continuous-time SAN description to one that is more convenient to work with when developing the algorithms introduced in chapters 4 and 5.

Before we introduce definitions and propositions regarding the description of a SAN, we define the equivalence of two SAN descriptions.

Definition 2.4 *Two SAN descriptions are said to be equivalent to each other if and only if their global generator matrices given by equation (2.1) are equal.*

Without loss of generality, we restrict ourselves to the case in which row sums of synchronizing transition probability matrices are either 0 or 1.

Definition 2.5 *A SAN description is said to be proper if and only if each synchronizing transition probability matrix has row sums of 0 or 1.*

The SAN descriptions of the three applications we consider in the numerical experiments are proper. However, in a given SAN description, row sums between 0 and 1 can very well be present in synchronizing transition rate matrices. Proposition 2.1 shows what should be done when such a case is encountered.

Proposition 2.1 *The description of a SAN that is not proper can be transformed to an equivalent SAN description that is proper.*

Proof. Without loss of generality, consider a SAN description of N automata and one synchronizing event. There are two possible cases. In the first case, row sums of the synchronizing transition probability matrix $Q_{e_1}^{(k)}$ corresponding to slave automaton k are all equal to some constant β such that $0 < \beta < 1$. This is the trivial case; we can replace $Q_{e_1}^{(k)}$ with $\hat{Q}_{e_1}^{(k)} = \frac{1}{\beta} Q_{e_1}^{(k)}$ and $Q_{e_1}^{(m)}$ with $\hat{Q}_{e_1}^{(m)} = \beta Q_{e_1}^{(m)}$, where m is the index of the master automaton of the synchronizing event. Row sums of the transformed matrix $\hat{Q}_{e_1}^{(k)}$ are 1. In the second case, row sums of $Q^{(k)}_{e_1}$ are not equal and some are between 0 and 1. This implies that transition rates of the master automaton m of the synchronizing event depend on the state of automaton k. Therefore, it is possible to replace $Q^{(k)}_{e_1}$ with a matrix that has row sums of 0 or 1 by introducing functional transitions to $Q^{(m)}_{e_1}$ as follows. Let β_l, $l = 1, 2, \ldots, n_k$, be the sum of row l in $Q^{(k)}_{e_1}$. We replace $Q^{(k)}_{e_1}$ with $\hat{Q}^{(k)}_{e_1}$ in which $\hat{Q}^{(k)}_{e_1}(i,j) = Q^{(k)}_{e_1}(i,j)/\beta_i$ if $0 < \beta_i < 1$, else $\hat{Q}^{(k)}_{e_1}(i,j) = Q^{(k)}_{e_1}(i,j)$, for $j = 1, 2, \ldots, n_k$. We also replace $Q^{(m)}_{e_1}$ with $\hat{Q}^{(m)}_{e_1}$ in which $\hat{Q}^{(m)}_{e_1}(i,j) = \beta_l Q^{(m)}_{e_1}(i,j)$ if $0 < \beta_l < 1$, else $\hat{Q}^{(m)}_{e_1}(i,j) = Q^{(m)}_{e_1}(i,j)$, for $i, j = 1, 2, \ldots, n_m$ when $\mathcal{A}^{(k)}$ is in state l. The transformed matrix $\hat{Q}^{(k)}_{e_1}$ has row sums of 0 or 1.

Given a synchronizing event, the above modifications must be made for each of its synchronizing transition probability matrices that has row sums between 0 and 1. This implies that nonzero elements in the synchronizing event transition rate matrix of the corresponding master automaton may need to be scaled multiple times with values that depend on the state of multiple slave automata. After modifying the synchronizing event matrices, the corresponding diagonal corrector matrices must also be modified accordingly. The new SAN description has synchronizing transition probability matrices with row sums of 0 or 1, and therefore is proper. The generalization to E (> 1) synchronizing events is straightforward. $\square$

Observe that the transformation of a SAN description discussed in the proof of Proposition 2.1 may cause the number of functional elements in the synchronizing transition rate matrices of automata to increase. However, the number of synchronizing events as well as the nonzero structure of the synchronizing transition matrices of automata remain unchanged.

Now we introduce a definition related to the separability of synchronizing transition rates from local transition rates.

Definition 2.6 *Synchronizations are said to be separable from local transitions in a given SAN description if and only if for any synchronizing event t whose master is automaton m and $i, j = 1, 2, \ldots, n_m$, $Q^{(m)}_{e_t}(i,j) \neq 0$ implies $Q^{(m)}_{l}(i,j) = 0$.*

Definition 2.6 may seem to be specifying an artificial condition at first, yet the condition is satisfied by the three applications we consider. As we shall see in chapter 4, this property enables the preprocessing of local transition rate matrices separately from synchronizing transition matrices which significantly improves the complexity of the NCD partitioning algorithm we propose. Even though the three SAN descriptions we consider have separable synchronizations, one may very well encounter those that do not satisfy this property. Proposition 2.2 shows that a SAN description whose synchronizations are not separable can be handled in the framework discussed in this paper.

Proposition 2.2 *The description of a SAN having synchronizations that are not separable from local transitions can be transformed to an equivalent SAN description whose synchronizations are separable from local transitions.*

Proof. Assume that the given SAN description does not satisfy the condition in Definition 2.2. Without loss of generality, let t be the event, m its master, and (i,j) the indices of the problematic element. Decompose $Q_l^{(m)}$ into three terms as

$$Q_l^{(m)} = R_l^{(m)} + Q_l^{(m)}(i,j)u_iu_j^T - Q_l^{(m)}(i,j)u_iu_i^T,$$

where u_i is the ith column of the identity matrix. Here $R_l^{(m)}$ is a transition rate matrix, the second term is a matrix with a single nonzero transition rate at element (i,j) and the third term is the diagonal corrector of the second term. Now, let $R_l^{(m)}$ be the local transition rate matrix of automaton m and introduce the new synchronizing event v with master automaton m; $Q_{e_v}^{(m)}(= Q_l^{(m)}(i,j)u_iu_j^T)$ is the rate matrix associated with automaton m and synchronizing event v, and $\bar{Q}_{e_v}^{(m)}(= -Q_l^{(m)}(i,j)u_iu_i^T)$ is its diagonal corrector. All other matrices corresponding to synchronizing event v are equal to identity. Now, recall the following identity from tensor algebra

$$A \oplus (Q_l^{(m)} + Q_{e_v}^{(m)} + \bar{Q}_{e_v}^{(m)}) \oplus B = (A \oplus Q_l^{(m)} \oplus B) + (I \otimes Q_{e_v}^{(m)} \otimes I) + (I \otimes \bar{Q}_{e_v}^{(m)} \otimes I)$$

and compare its right-hand side with equation (2.1). The new SAN description has separable synchronizations.

The generalization to the cases when event t has more than one problematic element and the SAN description has more than one synchronizing event that are not separable from local transitions is straightforward. $\square$

The number of synchronizing events in the new SAN description obtained through the transformation discussed in the proof of Proposition 2.2 is larger than the number of synchronizing events in the original SAN. The difference in the number of synchronizing events corresponds to the number of the synchronizing events in the original SAN that are not separable. Nevertheless, assuming that identity matrices are not stored explicitly, the described transformation does not increase the number of nonzeros in the transformed SAN description.

Our next definition related to the SAN description involves the number of nonzero elements in synchronizing transition rate matrices. Without loss of generality, we restrict ourselves to the case where all synchronizing events in a SAN are simple.

Definition 2.7 *Synchronizations in a given SAN description are said to be simple if and only if for any synchronizing event t whose master is automaton m, $Q_{e_t}^{(m)}$ has only one nonzero element.*

In a SAN description whose synchronizations are simple, each synchronizing event can be characterized by a value that corresponds to the synchronizing transition rate of the event. In chapter 4, we show how to take advantage of this property. In most of the cases, we will not encounter SAN descriptions whose synchronizations are simple. The next proposition shows how SAN descriptions that do not satisfy the condition of Definition 2.7 can be handled in the framework of our approach.

Proposition 2.3 *The description of a SAN having synchronizing events that are not simple can be transformed to an equivalent SAN description whose synchronizing events are simple.*

Proof. Assume that the given SAN description does not satisfy the condition in Definition 2.7. Without loss of generality, let t be the event, m its master, and nz the number of nonzeros in $Q_{e_t}^{(m)}$. Decompose $Q_{e_t}^{(m)}$ into nz simple synchronizing transition rate matrices thereby creating nz new synchronizing events with master automaton m. The slave automata of the new synchronizing events are the slave automata of synchronizing event t. The transition probability matrices and their diagonal correctors associated with the new slave automata are respectively equal to the transition probability matrix and its diagonal corrector associated with the slave automata for synchronizing event t. All other matrices corresponding to the new synchronizing events are equal to identity. The new SAN description has simple synchronizations.

If the SAN description has more than one synchronizing event that are not simple, the decomposition presented in this proof must be applied to each synchronizing event that does not satisfy the condition of Definition 2.7. □

Application of the transformation described in the proof of Proposition 2.3 to a SAN description whose synchronizing events are not simple leads to an increase in the number of synchronizing events. The number of the simple synchronizing events in the new SAN description is equal to the number of nonzero elements in the synchronizing transition rate matrices of the original SAN. Note that the described transformation does not change the synchronizing transition probability matrices and their diagonal correctors. Hence, it is possible to keep the number of nonzero elements in the new SAN description the same as in the original SAN description.

2.3 Conclusion

In this chapter we presented basic definitions of SAN formalism, its components and their description. We have shown that, in a SAN, a Markovian system of loosely connected components is modeled by a stochastic automaton for each component. In a continuous-time SAN, the description of each automaton consists of a local transition rate matrix and two transition matrices per each synchronizing event taking place in the system.

We have also seen that transitions in the matrices of automata can be constant or functional. In the latter case, the value of a function depends on the state of other automata. Functional dependencies among automata can be characterized by the dependency graph G that may be acyclic or may contain cycles. As we will see in chapters 4 and 5, cyclic dependencies among automata introduce additional problems to the analysis of SANs.

We also made assumptions regarding the description of a SAN which is suitable to work with when developing the algorithms in chapters 4 and 5. We showed that if the description of a SAN does not satisfy the requirements in section 2.2, then it can be transformed to an equivalent SAN description that meets the requirements.

In the next chapter, we introduce a wireless ATM system and model it as a discrete-time SAN.

Chapter 3

Discrete-time SAN model of a wireless ATM system

Similar to continuous-time SANs (see section 2.1), a discrete-time system of N components can be modeled by a single stochastic automaton for each component. When there are E synchronizing events in the system, automaton k has the corresponding transition probability matrix $P_{e_j}^{(k)}$ that represents the contribution of $\mathcal{A}^{(k)}$ to synchronization $j \in \{1, 2, \ldots, E\}$ (see [26, p. 333]). The underlying discrete-time MC (DTMC) corresponding to the global system can be obtained from

$$P = \sum_{j=1}^{E} \bigotimes_{k=1}^{N} P_{e_j}^{(k)}. \tag{3.1}$$

We refer to the tensor representation in equation (3.1) associated with the DTMC as the descriptor of the discrete-time SAN.

The original formula introduced in [50] to study discrete-time SANs is given by

$$P = \bigotimes_{k} L^{(k)} + \sum_{s} (\bigotimes_{k} R_s^{(k)} - \bigotimes_{k} N_s^{(k)}), \tag{3.2}$$

where $L^{(k)}$ are the local transition matrices, $R_s^{(k)}$ are the synchronization matrices, and $N_s^{(k)}$ are the normalization matrices. Note that this formula is the same as the formula for continuous-time SANs (2.1) in which the tensor sum is replaced with a tensor product for the local transitions.

We consider the form of the descriptor in equation (3.1) rather than (3.2) since it is compact and easier to work with. Note that (3.2) can be transformed to (3.1) as

$$P_{e_j}^{(k)} = \begin{cases} L^{(k)}, & 1 \leq k \leq N,\ j = 1 \\ R_s^{(k)}, & 1 \leq k \leq N,\ 1 \leq s \leq S,\ j = s+1, \\ -N_s^{(k)}, & k = 1,\ 1 \leq s \leq S,\ j = S+s+1, \\ N_s^{(k)}, & 2 \leq k \leq N,\ 1 \leq s \leq S,\ j = S+s+1, \end{cases}$$

where S is the number of synchronizing events in (3.2) and $E = 2S + 1$.

The underlying DTMC of a discrete-time SAN is generally a dense matrix unlike the CTMC corresponding to a continuous-time SAN. The matrices $P_{e_j}^{(k)}$ are relatively dense compared to their continuous-time counterparts implying large number of floating-point arithmetic operations in the generalized descriptor-vector multiply algorithm (see [24, p. 404]) used in iterative solution methods. Hence, even though the DTMC need not be generated and stored during SAN analysis, discrete-time SANs are often evaded by researchers.

In the next 3 sections, we introduce the wireless ATM model that is investigated. In doing so, we first describe the system, then introduce the basic model with two types of services, and then add a third service giving a more general model.

3.1 Description of the system

The application that we consider arises in wireless asynchronous transfer mode (ATM) networks [30]. In [68], a multiservices resource allocation policy (MRAP) is developed to integrate two types of service over a time division multiple access (TDMA) system in a mobile communication environment. These are the constant bit rate (CBR) service for two types of voice calls (i.e., handover calls from neighboring cells and new calls) and the available bit rate (ABR) service for data transfer. See [30] for a detailed description of CBR and ABR services. A single cell and a single carrier frequency is modeled.

The TDMA frame is assumed to have C slots. Handover CBR requests have priority over new CBR calls and they respectively arrive with probabilities p_h and p_n. We do not consider multiple handover or new CBR call arrivals during a TDMA frame since the associated probabilities with these events are small.

Each CBR call takes up a single slot of a TDMA frame but may span multiple TDMA frames, whereas each data packet is small enough to be served in a single TDMA slot. When all the slots are full, incoming CBR calls are rejected. The number of CBR calls that may terminate in a given TDMA frame depends on the number of active CBR calls, but can be at most M, and hence is modeled as a truncated binomial process with parameter p_s.

Data are queued in a FIFO buffer of size B and has the least priority. The arrival of data packets is modeled as an on-off process. The process moves from the on state to the off state with probability α and from the off state to the on state with probability β. The load offered to the system is defined as $\lambda = \beta/(\alpha + \beta)$. Assuming that the time interval between two consecutive on periods is t, the burstiness of such an on-off process is described by the square coefficient of variation, $S_C = Var(t)/[E(t)]^2$. In terms of λ and S_C, $\beta = 2\lambda(1 - \lambda)/(S_C + 1 - \lambda)$ and $\alpha = \beta(1 - \lambda)/\lambda$.

When the on-off process is in the on state, we assume that $i \in \{0, 1, 2, 3\}$ data packets may arrive with probability p_{di}. Then the mean arrival rate of data packets is defined as $\rho = \sum_{i=1}^{3} i \times p_{di}$. Hence, the global mean arrival rate of data packets is given by $\Gamma = \lambda\rho$. If the number of arriving data packets exceeds the free space in the buffer plus the number of free slots in the current TDMA frame, then the excess packets are blocked. The arrival process of data and the service process of CBR calls we consider are quite general and subsume those in [68]. Furthermore, when compared to the model in [68], our SAN model is scalable since its number of synchronizing events is independent of C.

The performance measures of interest are the dropping probability of handover CBR calls, the blocking probability of new CBR calls, and the blocking probability of data packets. Note that dropping refers to the rejection of an existing call, whereas blocking refers to the rejection of a new call or packet.

We model this system as a discrete-time SAN in which state changes occur at TDMA frame boundaries. Regarding the events that take place in the system, we make the following assumption. Data packet and call arrivals to the system happen at the beginning of a frame, and data packet transmissions finish and CBR calls terminate at the end of the frame. Since each data packet is transmitted in a single slot, in a particular state of the system we never see slots occupied by data packets.

3.2 The basic SAN model

The basic SAN model consists of 3 automata and 3 synchronizing events. For convenience, we number synchronizing events starting from 0. States of all automata are numbered starting from 0 as well. We denote the state index of automaton k by $s\mathcal{A}^{(k)}$; $\mathcal{A}^{(1)}$ represents the data source, $\mathcal{A}^{(2)}$ represents the current TDMA frame, and $\mathcal{A}^{(3)}$ represents the data buffer. We define the three synchronizing events e_0, e_1, e_2 that correspond to respectively 0, 1, 2 CBR arrivals during the current TDMA frame. Synchronizing event e_a happens with probability γ_a, $a \in \{0, 1, 2\}$, where $\gamma_0 = \bar{p}_n \bar{p}_h$, $\gamma_1 = p_n \bar{p}_h + p_h \bar{p}_n$, $\gamma_2 = p_h p_n$, and $\bar{q} = 1 - q$ when $q \in [0, 1]$.

$\mathcal{A}^{(1)}$ has two states that correspond to the on and off states of the data source. Transitions in this automaton happen independently of the other automata. Hence, we have

$$P_{e_0}^{(1)} = P_{e_1}^{(1)} = P_{e_2}^{(1)} = \begin{bmatrix} \bar{\beta} & \beta \\ \alpha & \bar{\alpha} \end{bmatrix}.$$

The current TDMA frame is modeled by $\mathcal{A}^{(2)}$ that has $(C + 1)$ states. If $s\mathcal{A}^{(2)} = i$, then the current TDMA frame has i active CBR connections. The contribution of $\mathcal{A}^{(2)}$ to synchronization e_a, $a \in \{0, 1, 2\}$, is given by the matrix

$P_{e_a}^{(2)}$ of order $(C+1)$ with ijth element

$$p_{ij} = \begin{cases} \gamma_a \begin{pmatrix} i+a \\ i+a-j \end{pmatrix} p_s^{i+a-j}(1-p_s)^j, & i+a \le C,\ 0 \le i+a-j < M \\ \gamma_a \begin{pmatrix} C \\ C-j \end{pmatrix} p_s^{C-j}(1-p_s)^j, & i+a > C,\ 0 \le i+a-j < M \\ \gamma_a \sum_{k=j}^{C} \begin{pmatrix} i+a \\ i+a-k \end{pmatrix} p_s^{i+a-k}(1-p_s)^k, & i+a \le C,\ 0 \le i+a-j = M \\ \gamma_a \sum_{k=j}^{C} \begin{pmatrix} C \\ C-k \end{pmatrix} p_s^{C-k}(1-p_s)^k, & i+a > C,\ 0 \le i+a-j = M \\ 0, & \text{otherwise} \end{cases}$$

for $i,j = 0,1,\ldots,C$.

The data buffer is modeled by $\mathcal{A}^{(3)}$ that has $(B+1)$ states. If $s\mathcal{A}^{(3)} = i$, then the buffer has i data packets. Transitions of this automaton depend on $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ and we have

$$P_{e_0}^{(3)} = \begin{bmatrix} g_1 & p_{-1} & p_{-2} & p_{-3} & & & & & & \\ g_1 & p_0 & p_{-1} & p_{-2} & p_{-3} & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & & & \\ g_1 & p_{C-1} & p_{C-2} & p_{C-3} & p_{C-4} & \cdots & p_{-3} & & & \\ & p_C & p_{C-1} & p_{C-2} & p_{C-3} & \cdots & p_{-2} & p_{-3} & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & p_C & p_{C-1} & p_{C-2} & p_{C-3} & \cdots & p_{-2} & g_2 \\ & & & & p_C & p_{C-1} & p_{C-2} & \cdots & p_{-1} & g_2 \\ & & & & & p_C & p_{C-1} & \cdots & p_0 & g_2 \\ & & & & & & p_C & \cdots & p_1 & g_2 \end{bmatrix}.$$

The matrices $P_{e_1}^{(3)}$ and $P_{e_2}^{(3)}$ have the same (functional) nonzero structure as that of $P_{e_0}^{(3)}$, but different contents. For synchronization e_a, the probability p_l is obtained from

$$p_l = \begin{cases} p_{d(FS(a)-l)}, & s\mathcal{A}^{(1)} = 1,\ 0 \le (FS(a)-l) \le 3 \\ 1, & s\mathcal{A}^{(1)} = 0,\ FS(a) = l \\ 0, & \text{otherwise} \end{cases}, \tag{3.3}$$

where $FS(a) = [C - (s\mathcal{A}^{(2)} + a)]^+$ denotes the number of free slots in the current TDMA frame taking into account up to $a \in \{0,1,2\}$ possible CBR arrivals. The values g_1 and g_2 are given by

$$g_1 = \sum_{l=s\mathcal{A}^{(3)}}^{C} p_l, \quad g_2 = \sum_{l=B-s\mathcal{A}^{(3)}}^{3} p_{(-l)}.$$

The stochastic one-step transition probability matrix of the underlying Markov chain is given by

$$R = \sum_{a=0}^{2} \bigotimes_{k=1}^{3} P_{e_a}^{(k)}.$$

3.3 A SAN model for VBR traffic

In this section, we consider a SAN model of a wireless ATM system that accepts variable bit rate (VBR) calls. See [30] for a detailed description of VBR service. We assume that an arrival associated with VBR traffic can be either a new call or a handover just like CBR arrivals. Handover VBR requests have priority over new VBR calls and they respectively arrive with probabilities p_{vh} and p_{nh}. We do not consider multiple handover or new VBR call arrivals during a TDMA frame since the associated probabilities with these events are small.

A VBR connection is characterized by a state of high intensity and a state of low intensity. In the state of high intensity, the VBR source transmits data with its highest rate, whereas in the state of low intensity, its transmission rate is reduced. The reduced transmission rate in the low intensity state in fact means that at some instances of time the slot in a TDMA frame allocated to the VBR connection will not be used. Hence, slots reserved for VBR traffic in a TDMA frame can be used by ABR traffic in two ways. First, a slot may have been reserved for VBR traffic for which there is no active VBR connection. Second, there is a VBR connection associated with the given slot, but the connection is in the low intensity state and nothing is transmitted during this TDMA frame.

A VBR connection moves from the state of high intensity to the state of low intensity with probability α_v and from the state of low intensity to the state of high intensity with probability β_v. In terms of $\lambda_v = \beta_v/(\alpha_v + \beta_v)$ and its square coefficient of variation S_{C_v}, we have $\beta_v = 2\lambda_v(1-\lambda_v)/(S_{C_v}+1-\lambda_v)$ and

$\alpha_v = \beta_v(1-\lambda_v)/\lambda_v$ (see section 3.1). For the low intensity state, we introduce the parameters p_{empty} and p_{busy}. The former is the probability of transition from the state when the slot allocated to the VBR connection is busy to the state when the slot is empty. The latter is the probability of transition from the empty state to the busy state. We assume that when a VBR connection (either a new call or a handover) is set up, it is in the high intensity state. On the other hand, the connection can terminate in any state of the VBR source. We also assume that when a VBR connection changes its state from high intensity to low intensity, it enters the busy state.

The number of VBR calls that may terminate in a given TDMA frame depends on the number of active VBR calls and the duration of each VBR call is assumed to be a geometric process with parameter p_{vs}. VBR arrivals to the system are assumed to happen at the beginning of a TDMA frame, and state changes of a VBR connection after it is set up are assumed to take place at the end of a frame.

The performance measures of interest are the dropping probability of handover VBR calls and the blocking probability of new VBR calls. We model each slot reserved for VBR traffic in the current TDMA frame by a single automaton of 4 states. State 0 of the automaton corresponds to the case of an idle slot, i.e., the VBR connection is not active. State 1 corresponds to the state of high intensity, states 2 and 3 correspond to the state of low intensity. In particular, state 2 indicates that the slot is busy and state 3 indicates that it is empty.

Similar to CBR arrivals, in a given TDMA frame we can have at most two VBR requests arriving simultaneously. Therefore, we define the 3 synchronizing events f_0, f_1, f_2 that correspond to respectively 0, 1, 2 VBR arrivals. Note that VBR calls arrive to the system but not to a particular slot. Hence, if a TDMA frame has V slots reserved for VBR traffic numbered from 1 to V, the transition probability matrix P_{f_b} that corresponds to synchronizing event f_b, $b \in \{0,1,2\}$, is given by

$$P_{f_b} = \tilde{\gamma}_b \bigotimes_{k=1}^{V} P_{f_b}^{(k)} = \tilde{\gamma}_b P_{f_b}^{(1)} \bigotimes \left(\bigotimes_{k=2}^{V} P_{f_b}^{(k)} \right).$$

Here $P_{f_b}^{(k)}$ is the contribution of $\mathcal{A}^{(k)}$ (corresponding to the kth VBR slot) to synchronizing event f_b, $\tilde{\gamma}_b$ is the probability of b VBR arrivals during the current TDMA frame, and $\tilde{\gamma}_0 = \bar{p}_{vn}\bar{p}_{vh}$, $\tilde{\gamma}_1 = p_{vn}\bar{p}_{vh} + p_{vh}\bar{p}_{vn}$, and $\tilde{\gamma}_2 = p_{vn}p_{vh}$. Thus, only one synchronizing event matrix of synchronizing event f_b needs to be scaled by $\tilde{\gamma}_b$. Hence, for convenience we define

$$P_{f_b}^{(k)} = \begin{cases} \tilde{\gamma}_b P_b^{(k)}, & k = 1 \\ P_b^{(k)}, & 1 < k \leq V \end{cases},$$

where $P_b^{(k)}$ is the transition probability matrix describing the evolution of the kth TDMA slot when there are b VBR arrivals.

The transition probability matrix $P_0^{(k)}$ is given by

$$P_0^{(k)} = \begin{bmatrix} 1 & & & \\ p_{vs} & \bar{p}_{vs}\bar{\alpha}_v & \bar{p}_{vs}\alpha_v & \\ p_{vs} & \bar{p}_{vs}\beta_v & \bar{p}_{vs}\bar{\beta}_v\bar{p}_{empty} & \bar{p}_{vs}\bar{\beta}_v p_{empty} \\ p_{vs} & \bar{p}_{vs}\beta_v & \bar{p}_{vs}\bar{\beta}_v p_{busy} & \bar{p}_{vs}\bar{\beta}_v\bar{p}_{busy} \end{bmatrix}.$$

When a single VBR request arrives to the system, only one automaton among those that are in the idle state (i.e., state 0) should change its state. We choose the automaton that is in the idle state and that has the smallest index. If all V automata are in active states (i.e., there are V active VBR connections), the incoming VBR call is rejected. Observe that if an automaton is in one of the three active states (i.e., states 1-3), the transition probabilities out of the active state are the same as those for 0 VBR arrivals (see rows 2-4 of matrix $P_0^{(k)}$). Thus, the transition probability matrix $P_1^{(k)}$ is given by

$$P_1^{(k)} = \begin{bmatrix} 1 - g_3(k)\bar{p}_{vs} & g_3(k)\bar{p}_{vs}\bar{\alpha}_v & g_3(k)\bar{p}_{vs}\alpha_v & \\ p_{vs} & \bar{p}_{vs}\bar{\alpha}_v & \bar{p}_{vs}\alpha_v & \\ p_{vs} & \bar{p}_{vs}\beta_v & \bar{p}_{vs}\bar{\beta}_v\bar{p}_{empty} & \bar{p}_{vs}\bar{\beta}_v p_{empty} \\ p_{vs} & \bar{p}_{vs}\beta_v & \bar{p}_{vs}\bar{\beta}_v p_{busy} & \bar{p}_{vs}\bar{\beta}_v\bar{p}_{busy} \end{bmatrix},$$

where

$$g_3(k) = \begin{cases} 1, & s\mathcal{A}^{(k)} = 0 \text{ and } \sum_{l=1}^{k-1} \tilde{s}\mathcal{A}^{(l)} = k \\ 0, & \text{otherwise} \end{cases}, \quad \tilde{s}\mathcal{A}^{(l)} = \begin{cases} 0, & s\mathcal{A}^{(l)} = 0 \\ 1, & \text{otherwise} \end{cases}.$$

The difference between synchronizing events f_1 and f_2 is that when two VBR requests arrive, two automata among those that are in the idle state should change their states. Hence, we only have to redefine the function $g_3(k)$. The new function $g_4(k)$ should return 1 for the two automata that are in the idle state and that have the smallest indices.

$$g_4(k) = \begin{cases} 1, & s\mathcal{A}^{(k)} = 0 \text{ and } (\sum_{l=1}^{k-1} \tilde{s}\mathcal{A}^{(l)} = k \text{ or } \sum_{l=1}^{k-1} \tilde{s}\mathcal{A}^{(l)} = k-1) \\ 0, & \text{otherwise} \end{cases} .$$

Thus, the transition probability matrix $P_2^{(k)}$ is $P_1^{(k)}$ in which $g_3(k)$ is replaced by $g_4(k)$.

Similar to the basic SAN model, the stochastic one-step transition probability matrix of the underlying Markov chain is given by

$$S = \sum_{b=0}^{2} \bigotimes_{k=1}^{V} P_{f_b}^{(k)}.$$

3.4 The combined SAN model

The SAN model of the combined system consists of $(3+V)$ automata. The first 3 are the automata of the basic SAN model and the last V are the automata dedicated to VBR arrivals. We remark that the automata of the SAN that handle CBR and VBR arrivals are mutually independent. Hence, in the combined model the set of synchronizing events that correspond to new call and handover arrivals is given by the Cartesian product $E_{CBR} \times E_{VBR}$, where $E_{CBR} = \{e_0, e_1, e_2\}$ and $E_{VBR} = \{f_0, f_1, f_2\}$. Therefore, we denote by s_{ab} the synchronizing event that is triggered by a CBR arrivals and b VBR arrivals. Then the synchronizing transition probability matrix of automaton $\mathcal{A}^{(k)}$, $k \in \{1, 2, \ldots, V+3\}$, and event s_{ab} for $a, b \in \{0, 1, 2\}$ is given by

$$P_{s_{ab}}^{(k)} = \begin{cases} P_{e_a}^{(k)}, & 1 \le k \le 3 \\ P_{f_b}^{(k-3)}, & 4 \le k \le V+3 \end{cases}$$

under the following conditions.

Each TDMA frame of the combined system that gives CBR, VBR, and ABR service consists of C slots reserved for CBR traffic and V slots reserved for VBR traffic. ABR traffic can be pushed into any reserved, but unused slots. Hence, data packets can be transmitted in the idle slots among the C reserved for CBR traffic, in the idle slots among the V reserved for VBR traffic, and in those slots among the V that are in the empty state. Since data packets can use effectively a maximum of $(C+V)$ slots, each C in the matrix $P_{e_0}^{(3)}$ and the expression g_1 should be replaced by $(C+V)$. Furthermore, the probability p_l in equation (3.3) and the function $FS(a)$ should be redefined as follows:

$$p_l = \begin{cases} p_{d(FS(a,b)-l)}, & s\mathcal{A}^{(1)} = 1, \ 0 \leq (FS(a,b)-l) \leq 3 \\ 1, & s\mathcal{A}^{(1)} = 0, \ FS(a,b) = l \\ 0, & \text{otherwise} \end{cases},$$

where a and b are the indices of synchronizing event s_{ab},

$$FS(a,b) = [C-(s\mathcal{A}^{(2)}+a)]^+ + [V-(\sum_{k=4}^{V+3}\delta(s\mathcal{A}^{(k)}=1)+\sum_{k=4}^{V+3}\delta(s\mathcal{A}^{(k)}=2)+b)]^+,$$

and δ denotes the Kronecker delta.

Finally, the underlying Markov chain of the combined system is given by

$$P = \sum_{a=0}^{2}\sum_{b=0}^{2}\bigotimes_{k=1}^{V+3} P_{s_{ab}}^{(k)}.$$

3.5 Conclusion

In this chapter we introduced a discrete-time SAN model for a wireless ATM system. First, we showed that in a discrete-time SAN, each automaton can be modeled by a single transition probability matrix for each event taking place in the system. The discrete-time Markov chain of the overall system is the sum of the tensor products of the matrices of automata corresponding to each synchronizing event.

The wireless ATM system is able to process CBR, VBR, and ABR arrivals. The first two types are connection oriented calls that arrive rarely but may last for a relatively long time. On the other hand, ABR calls that are essentially data packets do not require the establishment of a connection, but arrive more frequently compared to the first two types of requests. The system consists of three main components corresponding to a data source that has two states and controls the arrival of data packets, a TDMA frame that transmits the three types of requests, and a data buffer that stores arriving data packets that cannot be transmitted in the given TDMA frame. We assumed that in the TDMA frame duration at most one new CBR or VBR call and one CBR or VBR handover can arrive. In other words, in the given TDMA frame there can be 0, 1, 2 CBR or VBR arrivals. Hence, there are 9 synchronizing events in the system. In addition to the three main components modeled by 3 automata, there are V automata that model the V TDMA slots dedicated to VBR connections. Matrices of all automata except the automaton corresponding to the data source have functional transitions. Furthermore, the dependency graph of the SAN is acyclic.

The next chapter consists of the SAN analysis part of the thesis. Therein, we introduce the NCD partitioning algorithm that computes NCD partitions of a SAN from the description of its automata [33]. We also present the complexity analysis of the algorithm and discuss its efficiency on three continuous-time SAN models.

Chapter 4

SANs and near complete decomposability

The SAN formalism introduced in section 2.1 provides a compact description of a Markovian system having a large number of states. The existence of an efficient vector-descriptor multiplication algorithm enables the analysis of large MCs modeled as SANs that practically cannot be handled by conventional techniques. However, in some cases these advanced techniques may not be sufficient for efficient analysis. Standard iterative methods applied to NCD Markov chains converge to an approximate solution very slowly when solving the system of linear equations (1.2). Fortunately, these MCs can be analyzed more efficiently using an iterative aggregation-disaggregation technique as we will show.

In this chapter, we present the NCD partitioning algorithm that determines the NCD connected components (CCs) of the MC underlying a SAN for a given decomposability parameter. Since we deal with large MCs, we aim at computing the NCD CCs without generating the transition matrix underlying the system under consideration. In the next section, we introduce the definition of an NCD MC. In section 4.2, we discuss the NCD partitioning algorithm of three steps. In section 4.3, we provide its complexity analysis and in the subsequent section present results of experiments with three SAN models.

4.1 Nearly completely decomposable MCs

NCD MCs [43] are irreducible stochastic matrices that can be symmetrically permuted [17] to the block form

$$P_{n\times n} = \begin{pmatrix} P_{11} & P_{12} & \dots & P_{1K} \\ P_{21} & P_{22} & \dots & P_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ P_{K1} & P_{K2} & \dots & P_{KK} \end{pmatrix}$$

in which the nonzero elements of the off-diagonal blocks are small compared with those of the diagonal blocks [62, p. 286]. Hence, it is possible to represent an NCD MC as

$$P = \mathrm{diag}(P_{11}, P_{22}, \dots, P_{KK}) + F,$$

where the diagonal blocks P_{ii} are square and possibly of different order and $\|F\|_\infty$, where $\|F\|_\infty = \max_i \sum_j F(i,j)$, is relatively small. The quantity $\|F\|_\infty$ is referred to as the degree of coupling and is taken to be a measure of the decomposability of P. When the chain is NCD, it has eigenvalues close to 1, and the poor separation of the unit eigenvalue implies a slow rate of convergence for standard matrix iterative methods [19, p. 290]. Hence, NCD MCs are said to be ill-conditioned [43, p. 258]. We should remark that the definition of NCDness is given through a discrete-time Markov Chain. The underlying CTMC of a SAN can be transformed through uniformization [62, p. 24] to a DTMC for the purpose of computing its stationary probability vector as in

$$P = I + \frac{1}{\alpha} Q, \tag{4.1}$$

where $\alpha \geq \max_{1\leq i\leq n} |Q(i,i)|$. To preserve NCDness in this transformation, α must be chosen as $\max_{1\leq i\leq n} |Q(i,i)|$.

An NCD partitioning of P corresponding to a user specified decomposability parameter ϵ can be computed as follows (see [17] for details). First, construct an undirected graph whose vertices are the states of P by introducing an edge between vertices i and j if $P(i,j) \geq \epsilon$ or $P(j,i) \geq \epsilon$, and then identify its connected components[1] (CCs). Each CC forms a subset of the NCD partitioning. Notice that for a given value of ϵ, the maximum number of subsets in a

[1] Not to be mixed with the word component we have been using so far to mean subsystem.

computed partitioning is unique.

In the next section, we present a three step algorithm that computes an NCD partitioning of a MC modeled as a SAN from the description of its automata using a user specified decomposability parameter.

4.2 NCD partitioning algorithm for SANs

The following is our proposed solution algorithm that computes NCD partitionings of the MC underlying a SAN without generating Q (or P).

ALGORITHM 1

NCD partitioning of MC underlying SAN for given ϵ

Step 1. $Q \rightarrow P$ transformation

Step 2. Preprocessing synchronizing events

Step 3. Constructing NCD connected components

Step 1 computes the scalar α in equation (4.1) that describes the transformation of the global generator Q to a DTMC P through uniformization. In the next subsection, we show how this can be achieved efficiently by inspecting the diagonal elements in local transition rate matrices and the nonzero elements in diagonal corrector matrices.

Step 2 considers the locations of off-diagonal nonzero elements in the global generator Q. Off-diagonal nonzero elements in local transition rate matrices cannot contribute to the same nonzero element in Q due to the fact that these matrices form a tensor sum. Hence, their analysis is straightforward. However, off-diagonal nonzero elements in synchronizing transition rate matrices may contribute to the same nonzero element in Q since these matrices form a sum of tensor products. Therefore, it is necessary to identify those synchronizing events that may influence the NCD partitioning of the MC underlying the SAN by contributing to the value of the same nonzero element in Q. In subsection 4.2.2, we explain how this is done.

Finally, Step 3 determines the NCD CCs by analyzing local transition rate matrices and matrices corresponding to synchronizing events identified in Step 2 using ϵ and the value of α computed in Step 1. This is discussed in subsection 4.2.3.

4.2.1 $Q \rightarrow P$ transformation

The CTMC Q can be transformed to a DTMC P using equation (4.1) after $\alpha = \max_{1 \leq i \leq n} |Q(i,i)|$ is computed. It is Algorithm 1.1 in appendix 8.4.1 that computes α using local transition rate matrices, diagonal corrector matrices, and dependencies among automata.

Since Q is a CTMC, we have $Q(i,i) = -\sum_{j \neq i} Q(i,j)$ for $i = 1, 2, \ldots, n$. Note also that only the off-diagonal elements in P contribute to NCDness. Regarding the off-diagonal elements in Q, which determine the off-diagonal elements in P, we make the following observations.

Remark 4.1 *Each nonzero local transition rate in a SAN contributes to a different off-diagonal element in Q; two or more nonzero local transition rates cannot contribute to the same off-diagonal element in Q.*

This observation follows immediately from the term Q_l in equation (2.1) and the definition of tensor sum [5].

Remark 4.2 *A nonzero off-diagonal element in Q for a SAN with separable synchronizations is formed either of a nonzero local transition rate or of nonzero synchronizing transition rates but not of both.*

This observation follows from the definition of the SAN descriptor in equation (2.1) and Definition 2.6.

Remark 4.3 *A nonzero off-diagonal element that is formed of synchronizing transition rates in Q for a SAN can be represented as*

$$\sum_{j, e_j \in \mathcal{E}^*} q_{e_j}^{(m)} \prod_{k=1, k \neq m}^{N} q_{e_j}^{(k)}. \qquad (4.2)$$

Here $q_{e_j}^{(m)}$ *is the synchronizing transition rate in* $Q_{e_j}^{(m)}$*, where* m *is the index of the master automaton of event* j*;* $q_{e_j}^{(k)}$ *is a particular transition probability in* $Q_{e_j}^{(k)}$*, where* k $(\neq m)$ *is the index of a slave automaton of event* j*. Finally,* $\mathcal{E}^*$ *is the set of synchronizing events that contribute to the off-diagonal element of interest in* Q*, and* $|\mathcal{E}^*| \leq E$*. We have omitted the row and column indices from* $q_{e_j}^{(m)}$ *and* $q_{e_j}^{(k)}$*, since only the form of equation (4.2) is important for the current discussion.*

From Remarks 4.1–4.3 and equations (2.1), (4.1), and (4.2), P without its main diagonal, denoted by P^*, can be computed as

$$P^* = \bigoplus_{i=1}^{N} \left(\frac{1}{\alpha} Q_l^{(i)} \right) + \sum_{j=1}^{E} \bigotimes_{i=1}^{N} \hat{Q}_{e_j}^{(i)}, \tag{4.3}$$

where

$$\hat{Q}_{e_j}^{(i)} = \begin{cases} \frac{1}{\alpha} Q_{e_j}^{(i)} & \text{if } \mathcal{A}^{(i)} \text{ is the master of event } j \\ Q_{e_j}^{(i)} & \text{otherwise} \end{cases} .$$

Note that only rate matrices are scaled. What remains to be done is to compute α. To that effect, we state two propositions.

Proposition 4.1 *The diagonal element with maximum magnitude of* Q *for a SAN that does not have synchronizations and functional dependencies is given by the sum of diagonal elements with maximum magnitude of the local transition rate matrices. Thus,*

$$\alpha = \sum_{i=1}^{N} \max_{1 \leq k \leq n_i} |Q_l^{(i)}(k,k)|.$$

Proof. Recall Remark 4.1. Also note that the rates of transitions out of state k of automaton i all appear in the same row of Q, and consequently they contribute the value $Q_l^{(i)}(k,k)$ to the diagonal element of Q in that row. Since the state space of the global system is all possible combinations of the automata states, there is a global state in Q for which the off-diagonal row sums of all automata are maximized. □

Remark 4.4 *Dependencies among automata may arise either as explicit functions whose values depend on the states of automata other than the ones in* *which they are defined or implicitly by the existence of zero rows in synchronizing event matrices associated with slave automata. The latter case corresponds to the disabling of the synchronized transition when the slave automaton is in local state corresponding to the zero row.*

From now on, by dependencies we refer to both explicit and implicit dependencies as discussed in Remark 4.4. As an extension to Proposition 4.1, we have the next one.

Proposition 4.2 *The diagonal element with maximum magnitude of Q for a SAN that does not have dependencies can be obtained from*

$$\alpha = \sum_{i=1}^{N} \max_{1 \le k \le n_i} \left| \left(Q_l^{(i)}(k,k) + \sum_{j, e_j \in \mathcal{M}_i} \bar{Q}_{e_j}^{(i)}(k,k) \right) \right|, \tag{4.4}$$

where $\mathcal{M}_i$ is the set of synchronizing events whose master is automaton i.

Proof. Observe that local and synchronizing transitions of a master automaton that emanate from the same local state appear in the same row of Q. The rest of the proof follows from an argument similar to that for Proposition 4.1. Note that equation (4.4) is also valid for the case in which synchronizing transition probabilities are less than 1. In this case, the synchronizing transition rate in the master automaton is split according to transition probabilities in slave automata. However, these fractional synchronizing transition rates still appear in the same row of Q; that is, they contribute to a diagonal element of Q the corresponding diagonal element of the diagonal corrector matrix. $\square$

Example 4.1 *Now let us show the computation of the diagonal element with maximum magnitude of Q for the SAN in Example 2.1 with* $(\lambda_1, \lambda_2, \mu_1, \mu_2, \mu_3) = (2, 3, 3, 2, 1)$. *According to Proposition 4.2, we have*

$$\begin{aligned} \alpha &= \max_{1 \le k \le 2} |Q_l^{(1)}(k,k) + \bar{Q}_{e_1}^{(1)}(k,k)| + \max_{1 \le k \le 3} |Q_l^{(2)}(k,k) + \bar{Q}_{e_2}^{(2)}(k,k)| \\ &= \max\{\lambda_1, \lambda_2\} + \max\{\mu_1, \mu_2, \mu_3\} = \lambda_2 + \mu_1 = 6, \end{aligned}$$

which can be verified on

$$Q = \begin{pmatrix} -5 & 3 & 0 & 2 & 0 & 0 \\ 0 & -4 & 2 & 0 & 2 & 0 \\ 1 & 0 & -3 & 0 & 0 & 2 \\ 3 & 0 & 0 & -6 & 3 & 0 \\ 3 & 0 & 0 & 0 & -5 & 2 \\ 4 & 0 & 0 & 0 & 0 & -4 \end{pmatrix}.$$

Example 4.2 *Consider the SAN presented in Example 4.1 with the following modification:* $Q_{e_1}^{(2)}(1,1) = 0$ *and* $\bar{Q}_{e_1}^{(2)}(1,2) = 0$. *Note that the modified SAN still does not have functional transitions defined explicitly. However, the rate of synchronizing event 1 is in fact a function, and it depends on the state of automaton 2 (see Remark 4.4). When automaton 2 is in state 1, the synchronizing transition rate is 0 since it is disabled due to the modification, else it is 3. It is not possible to apply Proposition 4.2 to this SAN, since it would produce the incorrect result* $\alpha = 6$ *which can be seen from*

$$Q' = \begin{pmatrix} -5 & 3 & 0 & 2 & 0 & 0 \\ 0 & -4 & 2 & 0 & 2 & 0 \\ 1 & 0 & -3 & 0 & 0 & 2 \\ 0 & 0 & 0 & -3 & 3 & 0 \\ 3 & 0 & 0 & 0 & -5 & 2 \\ 4 & 0 & 0 & 0 & 0 & -4 \end{pmatrix}.$$

For SANs having dependencies, equation (4.4) cannot be used. A naive solution is to compute explicitly each diagonal element of Q and to find the element with maximum magnitude. However, this is expensive. To reduce the complexity, we propose to partition automata into dependency sets.

Definition 4.1 *Let* $G'(\mathcal{V},\mathcal{E})$ *be a digraph in which* v_i *corresponds to* $\mathcal{A}^{(i)}$ *and* $(v_i, v_j) \in \mathcal{E}$ *if transitions in* $\mathcal{A}^{(i)}$ *depend on the state of* $\mathcal{A}^{(j)}$ *either explicitly or implicitly as discussed in Remark 4.4. Then, the dependency sets of a SAN, denoted by* $\mathcal{D}_k$, $k = 1, 2, \ldots, N_{\mathcal{D}}$, *are the connected components of the dependency graph* G'.

We assume that for each automaton of the SAN, the set of automata with which it is involved in a functional dependency relationship is known. Regarding implicit dependencies that originate from synchronizations (see Remark 4.4), we make the following observation. If the diagonal corrector matrix $\bar{Q}_{e_j}^{(i)}$ of slave automaton i has at least one row with a zero diagonal element, then the master automaton of event j is dependent on the state of automaton i. Note that we do not need to scan each synchronizing event diagonal corrector matrix to detect such dependencies. Each row of a diagonal corrector matrix that is a probability transition matrix can have at most one nonzero element per row. Hence, if the number of nonzeros in $\bar{Q}_{e_j}^{(i)}$ is less than the order of the matrix (i.e., n_i), then the master automaton of event j depends on automaton i.

We refer to

$$\max\_\mathcal{D}_k = \max \left| \bigoplus_{i, \mathcal{A}^{(i)} \in \mathcal{D}_k} \mathrm{diag}(Q_l^{(i)}) + \sum_{j, e_j \in \mathcal{M}_{\mathcal{D}_k}} \bigotimes_{i, \mathcal{A}^{(i)} \in \mathcal{D}_k} \mathrm{diag}(\bar{Q}_{e_j}^{(i)}) \right| \tag{4.5}$$

as the maximum of the dependency set $\mathcal{D}_k$, where diag returns a vector consisting of the diagonal elements of its matrix argument, and $\mathcal{M}_{\mathcal{D}_k}$ is the set of synchronizing events whose masters are in $\mathcal{D}_k$.

Proposition 4.3 *The diagonal element with maximum magnitude of Q for a SAN can be obtained from*

$$\alpha = \sum_{k=1}^{N_{\mathcal{D}}} \max\_\mathcal{D}_k. \tag{4.6}$$

Proof. From the definition of dependency sets, automata in a dependency set are independent of automata in other dependency sets. Now, consider a new SAN description of $N_{\mathcal{D}}$ automata and E synchronizing events. In the new description, $\mathcal{A}^{(\mathcal{D}_k)}$, $k = 1, 2, \ldots, N_{\mathcal{D}}$, corresponds to the dependency set $\mathcal{D}_k$, $Q_l^{(\mathcal{D}_k)} = \bigoplus_{i, \mathcal{A}^{(i)} \in \mathcal{D}_k} Q_l^{(i)}$, $Q_{e_j}^{(\mathcal{D}_k)} = \bigotimes_{i, \mathcal{A}^{(i)} \in \mathcal{D}_k} Q_{e_j}^{(i)}$, and $\bar{Q}_{e_j}^{(\mathcal{D}_k)} = \bigotimes_{i, \mathcal{A}^{(i)} \in \mathcal{D}_k} \bar{Q}_{e_j}^{(i)}$, where $j = 1, 2, \ldots, E$. We define $\mathcal{A}^{(\mathcal{D}_k)}$ as the master of synchronizing event j if the master automaton of event j in the original SAN description is a member of $\mathcal{D}_k$. By construction, the new SAN does not have dependencies. Hence, we can apply Proposition 4.2. Substituting in equation (4.4) the matrices $Q_l^{(\mathcal{D}_k)}$ and $\bar{Q}_{e_j}^{(\mathcal{D}_k)}$, we obtain equation (4.6). □

Propositions 4.1, 4.2, and 4.3 are valid for irreducible MCs underlying SANs. When transient states and/or multiple essential subsets of states are present, the diagonal element with maximum magnitude given by equation (4.6) may not belong to the essential subset of interest. In the following, we refer to the states other than the ones in the essential subset of interest as uninteresting.

In the presence of uninteresting states, we can compute α by finding the maximums of all $N_{\mathcal{D}}$ dependency sets (see equations (4.5) and (4.6)). For dependency set $\mathcal{D}_k$, this task amounts to the enumeration of $\prod_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} n_i$ states and an equal number of floating-point comparisons. Now, observe that to max_$\mathcal{D}_k$ of the dependency set $\mathcal{D}_k$ corresponds a state S_k. Hence, if the global state s that corresponds to $S_1, S_2, \ldots, S_{N_{\mathcal{D}}}$ maps into the essential subset of interest, then α given by equation (4.6) is taken as the diagonal element with maximum magnitude. However, if s is an uninteresting state, we omit from further consideration the element that corresponds to max_$\mathcal{D}_k$ for $k = 1, 2, \ldots, N_{\mathcal{D}}$, and proceed as in the while-loop of Algorithm 1.1 in appendix 8.4.1.

In the first step, for $k = 1, 2, \ldots, N_{\mathcal{D}}$ we find the next largest value denoted by next_max_$\mathcal{D}_k$ from equation (4.5) and the corresponding state $\tilde{S}_k$. In order to find next_max_$\mathcal{D}_k$ rapidly, the vectors

$$\left| \bigoplus_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} \mathrm{diag}(Q_l^{(i)}) + \sum_{j,e_j\in\mathcal{M}_{\mathcal{D}_k}} \bigotimes_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} \mathrm{diag}(\bar{Q}_{e_j}^{(i)}) \right|,$$

where $k = 1, 2, \ldots, N_{\mathcal{D}}$, should be stored as sorted. In the second step, we find t such that next_max_$\mathcal{D}_t \geq$ next_max_$\mathcal{D}_k$ for $k = 1, 2, \ldots, N_{\mathcal{D}}$. Finally, we replace max_$\mathcal{D}_t$ with next_max_$\mathcal{D}_t$, S_t with $\tilde{S}_t$, and omit the element corresponding to next_max_$\mathcal{D}_t$ from further consideration. If the updated global state s maps to a state in the essential subset of interest, then α given by equation (4.6) is taken as the diagonal element with maximum magnitude. Else we go back to the first step. Since finite MCs have at least one recurrent state in each essential subset, the algorithm is terminating.

Our final remark is about the special case of a SAN with a single dependency set; that is, $N_{\mathcal{D}} = 1$ and $\mathcal{D}_1 = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N)}\}$. In this case, finding $\alpha =$ max_$\mathcal{D}_1$ amounts to enumerating all diagonal elements of Q since we have the equality $\bigoplus_{i,\mathcal{A}^{(i)}\in\mathcal{D}_1} \mathrm{diag}(Q_l^{(i)}) + \sum_{j,e_j\in\mathcal{M}_{\mathcal{D}_k}} \bigotimes_{i,\mathcal{A}^{(i)}\in\mathcal{D}_1} \mathrm{diag}(\bar{Q}_{e_j}^{(i)}) = \mathrm{diag}(Q)$. Therefore, for a SAN with a single dependency set, there is no need to sort and store diag(Q) as suggested. When finding the maximum of diag(Q), we test an element of diag(Q) only if its index corresponds to a state in the essential subset of interest. Although it is implemented, this case is omitted from the presentation of Algorithm 1.1 in appendix 8.4.1.

Example 4.3 *This example shows the computation of the diagonal element with maximum magnitude of Q for the SAN model of Example 2.2. The SAN has two dependency sets: $\mathcal{D}_1 = \{\mathcal{A}^{(1)}, \mathcal{A}^{(3)}\}$ and $\mathcal{D}_2 = \{\mathcal{A}^{(2)}\}$. Note that $\mathcal{A}^{(3)}$ functionally depends on the state of $\mathcal{A}^{(1)}$ due to functional transition f as well as due to synchronizing event 1 (see $\bar{Q}_{e_1}^{(1)}$). Hence, the diagonal element with maximum magnitude of Q is comprised of two terms. The maximum of $\mathcal{D}_1$ is given by*

$$\begin{aligned}\max\_\mathcal{D}_1 &= \max\left| diag(Q_l^{(1)}) \bigoplus diag(Q_l^{(3)}) + diag(\bar{Q}_{e_1}^{(1)}) \bigotimes diag(\bar{Q}_{e_1}^{(3)})\right| \\ &= \max\left|\begin{pmatrix} -2-f \\ -2-0 \\ -1-f \\ -1-3 \end{pmatrix} + \begin{pmatrix} 0 \\ -5 \\ 0 \\ 0 \end{pmatrix}\right| = \max\left|\begin{pmatrix} -5 \\ -2 \\ -6 \\ -4 \end{pmatrix} + \begin{pmatrix} 0 \\ -5 \\ 0 \\ 0 \end{pmatrix}\right| = 7.\end{aligned}$$

On the other hand, $\mathcal{D}_2$ is a singleton, and therefore the maximum of $\mathcal{D}_2$ is given by

$$\max\_\mathcal{D}_2 = \max\left| diag(Q_l^{(2)}) + diag(\bar{Q}_{e_2}^{(2)})\right| = \max\left|\begin{pmatrix} -2 \\ -5 \\ -4 \end{pmatrix} + \begin{pmatrix} -5 \\ 0 \\ 0 \end{pmatrix}\right| = 7.$$

Since the underlying MC is irreducible, $\alpha = \max\_\mathcal{D}_1 + \max\_\mathcal{D}_2 = 14$ as verified on Q (see Example 2.2).

As pointed out at the beginning of this subsection, an NCD partitioning of P that corresponds to a user specified decomposability parameter ϵ is determined by the off-diagonal elements in P. Having found α, we can obtain P^* by scaling each transition rate matrix of the SAN with $1/\alpha$ (see equation (4.3)).

4.2.2 Preprocessing synchronizing events

As it is mentioned in Remark 4.3, transition rates from different synchronizing event matrices may sum up to form a nonzero element in the generator matrix Q. Hence, in some cases it may not be possible to determine the value of an off-diagonal element in Q by inspecting each automaton separately. The aim of Step 2 in Algorithm 1 is to find sets of those synchronizing events that may influence the NCD partitioning of Q. We name these sets as potential sets of synchronizing events.

The potential sets are disjoint, and their union is a subset of the set of synchronizing events. It is Algorithm 1.2 in appendix 8.4.2 that finds these potential sets using synchronizing event matrices, ϵ, and α computed in Step 1. The output of Algorithm 1.2 is $N_{\mathcal{P}}$ potential sets denoted by $\mathcal{P}_r$, $r = 1, 2, \ldots, N_{\mathcal{P}}$.

There are two cases in which synchronizing events may influence the NCD partitioning of Q. First, a simple synchronizing event has the corresponding transition rate greater than or equal to $\alpha\epsilon$. Second, a set of synchronizing events contribute to the same element in Q, and the sum of the synchronizing transition rates of the events in the set is greater than or equal to $\alpha\epsilon$.

In the first case, each synchronizing event with transition rate greater than or equal to $\alpha\epsilon$ forms a potential set that is a singleton. When the transition rate of a synchronizing event is a function, its value can be evaluated only on the global state space. This can be done in Step 3 of Algorithm 1 when NCD CCs of the SAN are formed. Hence, if the synchronizing transition rate is a function and the maximum value of the function is not known in advance, then the corresponding synchronizing event also forms a potential set that is a singleton.

Regarding the second case, we make the following observation. The position of a synchronizing transition rate in Q is uniquely determined by all synchronizing transition matrices that correspond to the synchronizing event. This can be seen from equation (2.1). Hence, we have the following proposition.

Proposition 4.4 *In a SAN with simple synchronizations, the set $\mathcal{E}^*$ of synchronizing events contribute to the same nonzero element of Q if and only if there exists at least one nonzero element with the same indices in the matrices $Q^{(i)}_{e_j}$ for all $e_j \in \mathcal{E}^*$ and $i = 1, 2, \ldots, N$.*

Proof. The proof follows from equation (2.1), definition of tensor product, Definitions 2.6 and 2.7. □

Example 4.4 *Consider Example 2.1. We remark that synchronizing events 1 and 2 are simple. By inspecting the synchronizing event matrices of $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$, we see that $Q^{(1)}_{e_1}$ and $Q^{(1)}_{e_2}$ have a nonzero element with the same indices (2,1), and $Q^{(2)}_{e_1}$ and $Q^{(2)}_{e_2}$ have a nonzero element with the same indices (3,1). Hence, transition rates that correspond to e_1 (i.e., λ_2) and e_2 (i.e., μ_3) contribute to the same element of Q (see element $Q(6,1)$).*

Those synchronizing events that are not classified as potential sets of singletons must be tested for the condition in Proposition 4.4. The test of two events, t and u, for the condition requires the comparison of the indices of nonzero elements in $Q^{(i)}_{e_t}$ and $Q^{(i)}_{e_u}$ for $i = 1, 2, \ldots, N$; that is, we test N pairs of matrices. For k events, the number of matrix pairs that need to be tested is $Nk(k-1)/2$. Note that for three events, t, u, and v, the fact that the pairs $(Q^{(i)}_{e_t}, Q^{(i)}_{e_u})$ and $(Q^{(i)}_{e_u}, Q^{(i)}_{e_v})$ each have at least one nonzero element with the same indices for $i = 1, 2, \ldots, N$ does not imply that the events t and v also satisfy the condition. In other words, the condition is not transitive. This further complicates the test for the condition in Proposition 4.4.

In order to avoid excessive computation associated with the test, we consider the set of synchronizing events $\mathcal{P}$ as a potential set if for all $e_u \in \mathcal{P}$ there exists $e_v \in \mathcal{P}$ such that the condition in Proposition 4.4 is satisfied for synchronizing events u and v, and the sum of transition rates of synchronizing events in $\mathcal{P}$ is greater than or equal to $\alpha\epsilon$. According to this definition, we form potential sets as follows. Let $\mathcal{L}$ be the set of synchronizing events that are not classified as potential sets of singletons. We choose event $e_v \in \mathcal{L}$, remove it from $\mathcal{L}$, and test e_v with each event in $\mathcal{L}$ for the condition in Proposition 4.4. Let $\mathcal{K}$ be the set of events that satisfy this condition. Then, if the sum of the transition rates of synchronizing event v and those in $\mathcal{K}$ is greater than or equal to $\alpha\epsilon$, we remove the events that are in $\mathcal{K}$ from $\mathcal{L}$ and form the potential set $\mathcal{P} = \{e_v\} \cup \mathcal{K}$. We repeat this procedure for all events in $\mathcal{L}$ until $\mathcal{L} = \emptyset$.

Example 4.5 *Let us consider the application of Algorithm 1.2 to the SAN in Example 2.2 for which* $\alpha = 14$. *Let* $\epsilon = 0.3$ *implying* $\alpha\epsilon = 4.2$. *The transition rate of the master automaton of simple synchronizing event 1 is 5 and greater than* $\alpha\epsilon$. *See* $Q_{e_1}^{(3)}(2,1)$ *in Example 2.2. Hence, the first potential set,* $\mathcal{P}_1$, *consists of synchronizing event 1 only. The second synchronizing event of the SAN also forms a potential set. See* $Q_{e_2}^{(2)}(1,3)$ *for justification. Thus,* $\mathcal{P}_1 = \{e_1\}$ *and* $\mathcal{P}_2 = \{e_2\}$. *Now, consider the case in which* $\epsilon = 0.4$ *implying* $\alpha\epsilon = 5.6$. *Both transition rates of synchronizing events 1 and 2 are less than* $\alpha\epsilon$. *Hence, we have to test these two events for the condition in Proposition 4.4; that is, we check if each of the three pairs of matrices* $(Q_{e_1}^{(1)}, Q_{e_2}^{(1)})$, $(Q_{e_1}^{(2)}, Q_{e_2}^{(2)})$, *and* $(Q_{e_1}^{(3)}, Q_{e_2}^{(3)})$ *have at least one nonzero element with the same indices. However, the condition in Proposition 4.4 is not satisfied. Thus, the number of potential sets for the case of* $\epsilon = 0.4$ *is zero. This implies that neither of the synchronizing events influence the NCD partitioning of the underlying MC. Therefore, synchronizing events of the SAN are omitted from further consideration in Step 3 of Algorithm 1 when* $\epsilon = 0.4$.

4.2.3 Constructing NCD connected components

As indicated in Remark 4.2, a nonzero element in the global generator of a SAN originates either from a local transition rate or from one or more synchronizing transition rates. Hence, NCD CCs of the underlying MC are determined by (i) constant local transition rates that are greater than or equal to $\alpha\epsilon$, (ii) functional local transition rates that can take values greater than or equal to $\alpha\epsilon$, or (iii) transition rates of synchronizing events that are in the potential sets $\mathcal{P}_r$, $r = 1, 2, \ldots, N_{\mathcal{P}}$. These three different possibilities are implemented in Algorithm 1.3 that appears in appendix 8.4.3. The input parameters of the algorithm are local transition rate and synchronizing event matrices, ϵ, α computed by Algorithm 1.1, and potential sets formed by Algorithm 1.2. The output of Algorithm 1.3 is the set of NCD CCs of the underlying MC.

First, we consider possibility (i) in which local transition rates are constant, and assume that $Q = Q_l$ (see equation (2.1)). Using $\alpha\epsilon$, we can find the NCD CCs of $Q_l^{(i)}$, $i = 1, 2, \ldots, N$. Let $\mathcal{C}^{(i)}$ be the set of NCD CCs of $Q_l^{(i)}$, where a member of $\mathcal{C}^{(i)}$, denoted by $\mathtt{c}^{(i)}$, is a partition of states from $\mathcal{A}^{(i)}$. Let $\mathcal{B}$ and $\mathcal{H}$ be sets in which each member of either set is also a set. In other words, $\mathcal{B}$ as well as $\mathcal{H}$ is a set of sets. We define the binary operator $\odot$ between the two sets $\mathcal{B}$ and $\mathcal{H}$ as $\mathcal{B} \odot \mathcal{H} = \{\mathtt{b} \times \mathtt{h} \mid \mathtt{b} \in \mathcal{B}, \mathtt{h} \in \mathcal{H}\}$, where $\times$ is the ordinary Cartesian product operator. Then, based on the graph interpretation of the tensor sum operator discussed in [28], the set of NCD CCs is given by $\mathcal{C} = \mathcal{C}^{(1)} \odot \mathcal{C}^{(2)} \odot \cdots \odot \mathcal{C}^{(N)}$. Observe that if $\mathcal{C}^{(i)}$, $i = 1, 2, \ldots, N$, are singletons, then $\mathcal{C}$ is a singleton as well; that is, the underlying MC is not NCD for given ϵ. One can take advantage of the same property when there are only K $(< N)$ $\mathcal{C}^{(i)}$ that are singletons. In this case, we renumber the automata so that these K sets assume indices from $(N - K + 1)$ to N. Then these K sets can be replaced with the set $\mathcal{C}^{[N-K+1]} = \{\{1, 2, \ldots, n_{N-K+1}\} \times \{1, 2, \ldots, n_{N-K}\} \times \cdots \times \{1, 2, \ldots, n_N\}\}$.

Now we bring into the picture functional local transition rates and consider possibility (ii). Let us assume that the automata of the given SAN can be reordered and renumbered so that transitions of automaton i depend (if at all) on the states of higher indexed automata, but they do not depend on the states of lower indexed automata (see [24] for details). Since Cartesian product is associative, $\odot$ is also associative, and one can rewrite the expression for $\mathcal{C}$ as

$$\mathcal{C} = \left(\mathcal{C}^{(1)} \odot \left(\mathcal{C}^{(2)} \odot \cdots \odot \left(\mathcal{C}^{(N-1)} \odot \mathcal{C}^{(N)}\right) \cdots\right)\right). \tag{4.7}$$

Given $\mathcal{C}^{[k]} = \left(\mathcal{C}^{(k)} \odot \left(\mathcal{C}^{(k+1)} \odot \cdots \odot \left(\mathcal{C}^{(N-1)} \odot \mathcal{C}^{(N)}\right) \cdots\right)\right)$, the union of all members of $\mathcal{C}^{[k]}$ is a set that is equivalent to the product state space of $\mathcal{A}^{(k)}$, $\mathcal{A}^{(k+1)}, \ldots, \mathcal{A}^{(N)}$. Therefore, taking into account the assumed ordering of automata, functional transition rates of $\mathcal{A}^{(k)}$ can be evaluated and NCD CCs of $\mathcal{C}^{[k]}$ can be updated accordingly. More formally, let $Q_l^{(k)}(s_k, \tilde{s}_k)$ be a functional element, i.e., $Q_l^{(k)}(s_k, \tilde{s}_k) = f$. Then the NCD CCs $\mathtt{c}^{[k]}$,$\tilde{\mathtt{c}}^{[k]} \in \mathcal{C}^{[k]}$ must be joined if $(s_k, s_{k+1}, \ldots, s_N) \in \mathtt{c}^{[k]}$, $(\tilde{s}_k, s_{k+1}, \ldots, s_N) \in \tilde{\mathtt{c}}^{[k]}$, and $f(s_k, s_{k+1}, \ldots, s_N) \geq \alpha\epsilon$.

Example 4.6 *We illustrate possibilities (i) and (ii) on the SAN of Example 2.2 by omitting synchronizing events 1 and 2. Synchronizing events are treated* *in possibility (iii). We set* $\epsilon = 0.3$ *implying* $\alpha\epsilon = 4.2$ *and assume that the automata are ordered as* $\mathcal{A}^{(2)}$, $\mathcal{A}^{(3)}$, $\mathcal{A}^{(1)}$. *First, we find the NCD CCs of all local transition rate matrices as in possibility (i) treating functional transition rates as zero. Inspection of local transition rate matrices shows that local transition rates of all automata are less than* $\alpha\epsilon$. *Hence, we have* $\mathcal{C}^{(1)} = \{\{1_1\}, \{2_1\}\}$, $\mathcal{C}^{(2)} = \{\{1_2\}, \{2_2\}, \{3_2\}\}$, *and* $\mathcal{C}^{(3)} = \{\{1_3\}, \{2_3\}\}$. *The subscripts in the states enable us to distinguish between states with identical indices but that belong to different automata. According to equation (4.7), we form the NCD CCs of* $Q_l^{(3)} \bigoplus Q_l^{(1)}$, *i.e.,* $\mathcal{C}^{(3)} \odot \mathcal{C}^{(1)} = \{\{(1_3, 1_1)\}, \{(1_3, 2_1)\}, \{(2_3, 1_1)\}, \{(2_3, 2_1)\}\}$. *Then we continue with possibility (ii). The value of the functional transition rate* $Q_l^{(3)}(1, 2)$ $(= f)$ *depends on the state of* $\mathcal{A}^{(1)}$ *only. Hence, we can evaluate* f *when* $\mathcal{C}^{(3)} \odot \mathcal{C}^{(1)}$ *is formed. The functional transition rate* f *evaluates to 5, which is larger than* $\alpha\epsilon$, *when* $\mathcal{A}^{(1)}$ *is in state 2. Therefore, we join* $\{(1_3, 2_1)\}$ *and* $\{(2_3, 2_1)\}$. *Finally, the NCD CCs of* Q_l *are given by*

$$
\begin{aligned}
\mathcal{C} &= \mathcal{C}^{(2)} \odot (\mathcal{C}^{(3)} \odot \mathcal{C}^{(1)}) \\
&= \{\{1_2\}, \{2_2\}, \{3_2\}\} \odot \{\{(1_3, 1_1)\}, \{(1_3, 2_1), (2_3, 2_1)\}, \{(2_3, 1_1)\}\} \\
&= \{\{(1_2, 1_3, 1_1)\}, \{(1_2, 2_3, 1_1)\}, \{(2_2, 1_3, 1_1)\}, \{(2_2, 2_3, 1_1)\}, \\
&\quad \{(3_2, 1_3, 1_1)\}, \{(3_2, 2_3, 1_1)\}, \{(1_2, 1_3, 2_1), (1_2, 2_3, 2_1)\} \\
&\quad \{(2_2, 1_3, 2_1), (2_2, 2_3, 2_1)\}, \{(3_2, 1_3, 2_1), (3_2, 2_3, 2_1)\}\}
\end{aligned}
$$

Now we consider possibility (iii). When possibilities (i) and (ii) are handled, the union of all members in $\mathcal{C}$ is a set that corresponds to the global state space of the SAN. The transition rate of synchronizing event t can be taken into account as follows. Let $(s_1, s_2, \ldots, s_N) \in \mathsf{c}$ and $(\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_N) \in \tilde{\mathsf{c}}$, where $\mathsf{c}, \tilde{\mathsf{c}} \in \mathcal{C}$. Then c and $\tilde{\mathsf{c}}$ must be joined if $\prod_{i=1}^N Q_{e_t}^{(i)}(s_i, \tilde{s}_i) \geq \alpha\epsilon$. Since the global state space of the SAN is usually very large, it may take a significant amount of time to find all pairs c and $\tilde{\mathsf{c}}$ that satisfy this condition. Fortunately, the situation can be improved. Let p, $1 < p \leq N$, be the smallest index among automata involved in event t, i.e., $Q_{e_t}^{(i)} = I_{n_i}$ for $i = 1, 2, \ldots, p-1$. We rewrite the first two terms of equation (2.1) as

$$\bigoplus_{i=1}^{N} Q_l^{(i)} + \sum_{j=1}^{E} \bigotimes_{i=1}^{N} Q_{e_j}^{(i)} = \left(\bigoplus_{i=1}^{p-1} Q_l^{(i)} \right) \bigoplus Q_l^{[p]} + \left(\bigotimes_{i=1}^{p-1} I_{n_i} \right) \bigotimes Q_{e_t}^{[p]} + \sum_{j=1, j\neq t}^{E} \bigotimes_{i=1}^{N} Q_{e_j}^{(i)}, \quad (4.8)$$

where $Q_l^{[p]} = \bigoplus_{i=p}^{N} Q_l^{(i)}$, and $Q_{e_t}^{[p]} = \bigotimes_{i=p}^{N} Q_{e_t}^{(i)}$. From the definition of tensor sum (see appendix 8.1), the first two terms of expression (4.8) can be written as

$$\left(\bigoplus_{i=1}^{p-1} Q_l^{(i)}\right) \bigoplus Q_l^{[p]} + \left(\bigotimes_{i=1}^{p-1} I_{n_i}\right) \bigotimes Q_{e_t}^{[p]} = \left(\bigoplus_{i=1}^{p-1} Q_l^{(i)}\right) \bigoplus \left(Q_l^{[p]} + Q_{e_t}^{[p]}\right). \quad (4.9)$$

From (4.9), it can be seen that the transition rate of synchronizing event t can be taken into account on the smaller state space $\mathcal{C}^{(p)} \odot \mathcal{C}^{(p+1)} \odot \cdots \odot \mathcal{C}^{(N)}$. The same idea can be extended to the potential sets formed in Step 2. In other words, if for $\mathcal{P}_r$, there exists σ_r, $1 < \sigma_r \leq N$, such that $Q_{e_j}^{(i)} = I_{n_i}$ for $i = 1, 2, \ldots, \sigma_r - 1$ and all $e_j \in \mathcal{P}_r$, then transition rates of synchronizing events in $\mathcal{P}_r$ can be taken into account when the set $\mathcal{C}^{[\sigma_r]} = \mathcal{C}^{(\sigma_r)} \odot \mathcal{C}^{(\sigma_r+1)} \odot \cdots \odot \mathcal{C}^{(N)}$ is formed. We remark that for the assumed ordering of automata, all functional transitions that may be present in synchronizing transition matrices of events in $\mathcal{P}_r$ can be evaluated when $\mathcal{C}^{[\sigma_r]}$ is formed.

Example 4.7 *We continue with the illustration of Algorithm 1 on the SAN of Example 2.2. For $\epsilon = 0.3$, each of the two synchronizing events of the SAN is classified as a potential set. We assume the same ordering of automata, i.e., $\mathcal{A}^{(2)}$, $\mathcal{A}^{(3)}$, $\mathcal{A}^{(1)}$. After renumbering the automata, let the new indices of the automata be $\tilde{1}$, $\tilde{2}$, $\tilde{3}$, respectively. For the given ordering of automata, the smallest index among automata involved in event 1 as well as in event 2 is $\tilde{1}$. Hence, the transition rates of events 1 and 2 can be taken into account when $\mathcal{C}^{[\tilde{1}]} = \mathcal{C}$ is formed. Due to the transition rate of synchronizing event 1, we join the NCD CCs that have the members $(1_2, 2_3, 1_1)$ and $(3_2, 1_3, 1_1)$, $(2_2, 2_3, 1_1)$ and $(1_2, 1_3, 1_1)$, $(3_2, 2_3, 1_1)$ and $(1_2, 1_3, 1_1)$. Similarly, due to synchronizing event 2, we join the NCD CCs that have the members $(1_2, 1_3, 1_1)$ and $(3_2, 1_3, 2_1)$, $(1_2, 1_3, 2_1)$ and $(3_2, 1_3, 1_1)$, $(1_2, 2_3, 1_1)$ and $(3_2, 2_3, 2_1)$, $(1_2, 2_3, 2_1)$ and $(3_2, 2_3, 1_1)$. For justification, see $\mathcal{C}$ formed in the Example 4.6 and the SAN in Example 2.2.*

Our next remark is about cyclic dependencies. When the automata of a SAN have cyclic dependencies, they cannot be ordered as discussed. Such cases can be handled as follows.

Definition 4.2 *The SCC graph $G^{SCC}(\mathcal{V}^{SCC}, \mathcal{E}^{SCC})$ of a SAN is the digraph in which each vertex corresponds to an SCC of $G(\mathcal{V}, \mathcal{E})$, and the edge $(v_i^{SCC}, v_j^{SCC}) \in \mathcal{E}^{SCC}$ if $(v_k, v_l) \in \mathcal{E}$ with $v_k \in v_i^{SCC}$ and $v_l \in v_j^{SCC}$.*

Observe that $G^{SCC}(\mathcal{V}^{SCC}, \mathcal{E}^{SCC})$ is acyclic by construction. Assuming that automata of the SAN are ordered topologically with respect to G^{SCC}, let p be the smallest index among cyclically dependent automata. Then we can evaluate all functional transitions in the cyclically dependent automata when $\mathcal{C}^{[p]}$ is formed. This observation is omitted from Algorithm 1.3 presented in appendix 8.4.3. The special case in which a cyclic dependency is created by transitions in the synchronizing transition matrices of a particular event can be handled in the same way as discussed in possibility (iii). There, the potential set $\mathcal{P}_r$, $r \in \{1, 2, \ldots, N_{\mathcal{P}}\}$, is taken into account when $\mathcal{C}^{[\sigma_r]}$ is formed. Assuming that the automata are ordered topologically with respect to G^{SCC}, all functions in the matrices of synchronizing events that belong to $\mathcal{P}_r$ can be evaluated when $\mathcal{C}^{[\sigma_r]}$ is formed.

Our final remark is about a SAN with more that one essential subset of states and/or transient states. As in subsection 4.2.2, we refer to the states other than the ones in the essential subset of interest as uninteresting. For $1 < i \leq N$, we do not have a one-to-one mapping between the global state space and the union of all members in $\mathcal{C}^{[i]}$. Hence, we cannot say whether a member of $\mathsf{c}^{[i]} \in \mathcal{C}^{[i]}$ maps to a state in the essential subset of interest or to an uninteresting state. Therefore, the decomposition of $\mathcal{C}$ as in (4.7) that allows us to handle functional local transition rates and synchronizing transition rates on a smaller state space cannot be used. This is because one or both of the members that belong to the joined NCD CCs may map to an uninteresting state.

For a SAN with uninteresting states, possibilities (ii) and (iii) should be considered on the global state space. Hence, the NCD CCs $\mathsf{c}, \tilde{\mathsf{c}} \in \mathcal{C}$ should be joined only if the members under consideration from each of the two sets map into the essential subset of interest. When we compute $\mathcal{C} = \mathcal{C}^{(1)} \odot \mathcal{C}^{(2)} \odot \cdots \odot \mathcal{C}^{(N)}$, uninteresting states must also be omitted from consideration. From the definition of the binary operator $\odot$, if s_i and $\tilde{s}_i$ are in the same NCD CC of $\mathcal{C}^{(i)}$, then it must be that $(s_1, s_2, \ldots, s_{i-1}, s_i, s_{i+1} \ldots, s_N)$ and $(s_1, s_2, \ldots, s_{i-1}, \tilde{s}_i, s_{i+1} \ldots, s_N)$ are in the same NCD CC of $\mathcal{C}$. When uninteresting states are present, we exercise the additional constraint that $(s_1, s_2, \ldots, s_{i-1}, s_i, s_{i+1} \ldots, s_N)$ and $(s_1, s_2, \ldots, s_{i-1}, \tilde{s}_i, s_{i+1} \ldots, s_N)$ must belong to the essential subset of interest.

In the next subsection, we summarize for Algorithm 1 the detailed space and time complexity analysis that appears in appendix 8.5, and apply the results to Example 2.2.

4.3 Complexity analysis of the NCD partitioning algorithm

The core operation performed by an algorithm that finds the NCD CCs of a MC is floating-point comparison. Hence, we provide the number of floating-point comparisons performed in Algorithm 1. Regarding the algorithm's storage requirements, we remark that its three steps are executed sequentially. Hence, the maximum amount of memory required by Algorithm 1 is upper bounded by an integer array of length $O(n)$.

As in appendix 8.5, we assume that the MC underlying the SAN is irreducible. In Step 1, the number of floating-point comparisons is given by $\sum_{k=1}^{N_D} \prod_{i, \mathcal{A}^{(i)} \in \mathcal{D}_k} n_i$. For the best case in which each dependency set is a singleton, the number of floating-point comparisons reduces to $\sum_{i=1}^{N} n_i$. On the other hand, if all automata form a single dependency set, we have the upper bound $\prod_{i=1}^{N} n_i = n$. In Step 2, the lower bound on the number of floating-point comparisons is E, and it corresponds to the case in which the transition rate of each simple synchronizing event is greater than or equal to $\alpha\epsilon$. The upper bound is equal to $\frac{1}{2}E(E+1)$ floating-point comparisons. This number of floating-point comparisons is achieved when the transition rate of each simple synchronizing event is less than $\alpha\epsilon$ and the transition rates of synchronizing events do not sum up in Q. The number of floating-point comparisons in Step 3 depends strongly on the number of functional transitions and synchronizing events as well as the automata ordering. Assuming that in Step 2 of Algorithm 1 synchronizing event r is classified as the potential set $\mathcal{P}_r$, $r = 1, 2, \ldots, E$, and the automata are ordered as discussed in possibility (iii) in subsection 4.2.3, the number of floating-point comparisons in Step 3 is given by $\sum_{i=1}^{N} nz_l^{(i)} + \sum_{i=1}^{N-1} nf_i \prod_{j=i+1}^{N} n_j + \sum_{r=1}^{E} \prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$, where $nz_l^{(i)}$ is the number of nonzero off-diagonal elements in $Q_l^{(i)}$, nf_i is the number of functional transitions in $Q_l^{(i)}$, $nz_{e_r}^{(j)}$ is the number of nonzeros in $Q_{e_r}^{(i)}$, and m_r is the index of the master automaton of event r. Finally, the number of floating-point comparisons performed in Algorithm 1 is given by $E + \sum_{i=1}^{N}(n_i + nz_l^{(i)}) + \sum_{i=1}^{N-1} nf_i \prod_{j=i+1}^{N} n_j + \sum_{r=1}^{E} \prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$ in the best case, and $n + \frac{1}{2}E(E+1) + \sum_{i=1}^{N} nz_l^{(i)} + \sum_{i=1}^{N-1} nf_i \prod_{j=i+1}^{N} n_j + \sum_{r=1}^{E} \prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$ in the worst case.

Step 3 of Algorithm 1 also incurs floating-point multiplications when synchronizing events are handled. Computation of a single nonzero transition originating from synchronizing event r requires $(N - \sigma_r)$ floating-point multiplications. For synchronizing event r, we compute $\prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$ elements. Hence, the maximum number of floating-point multiplications in Algorithm 1.3 is $\sum_{r=1}^{E}[(N - \sigma_r) \prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}]$. Observe that this expression is almost the same as the last term of the expression for the number of floating-point comparisons performed in Algorithm 1. Hence, assuming that the time it takes to perform floating-point multiplication and floating-point comparison are of the same order, the time complexity of Algorithm 1 is roughly the number of floating-point comparisons.

Example 4.8 *As an example, we calculate the number of floating-point comparisons performed by Algorithm 1 to find an NCD partitioning of the MC underlying the SAN in Example 2.2. We use the same input parameters for Algorithm 1 as in subsection 4.2.3; that is,* $\epsilon = 0.3$ *and the automata are ordered as* $\mathcal{A}^{(2)}$, $\mathcal{A}^{(3)}$, $\mathcal{A}^{(1)}$. *The SAN in Example 2.2 has two dependency sets,* $\mathcal{D}_1 = \{\mathcal{A}^{(1)}, \mathcal{A}^{(3)}\}$ *and* $\mathcal{D}_2 = \{\mathcal{A}^{(2)}\}$. *Hence, Step 1 of Algorithm 1 takes* $n_1 n_3 + n_2 = 7$ *floating-point comparisons. The diagonal element with maximum magnitude of the SAN is 14 and* $\alpha\epsilon = 4.2$. *This SAN has two simple synchronizing events. Transition rates of the master automata of these events are greater than* $\alpha\epsilon$. *Hence, each synchronizing event is classified as a potential* *set, and the number of floating-point comparisons in Step 2 is 2.*

In Step 3, we first find the NCD CCs of local transition rate matrices. This operation takes 7 floating-point comparisons and corresponds to the number of off-diagonal nonzero elements in the local transition rate matrices. The value of the functional transition rate in $Q_l^{(3)}$ depends on the state of $\mathcal{A}^{(1)}$. Hence, for the given ordering of automata, the value of the function can be evaluated when $\mathcal{C}^{(3)} \odot \mathcal{C}^{(1)}$ is formed (see Algorithm 1.3). The number of these evaluations is equal to the number of states in $\mathcal{A}^{(1)}$. Hence, the number of floating-point comparisons due to the functional transition rate is 2. Recall that in Step 2, each synchronizing event is classified as a potential set. Hence, transition rates of both events must be taken into consideration in Step 3. For the given ordering of automata, let the new indices of the automata be $\tilde{1}$, $\tilde{2}$, and $\tilde{3}$, respectively. For potential set 1, we have $\sigma_1 = \tilde{1}$, and for potential set 2, we have $\sigma_2 = \tilde{1}$. Hence, the number of floating-point comparisons due to synchronizing events of the SAN is $nz_{e_1}^{(1)} nz_{e_1}^{(2)} + nz_{e_2}^{(1)} nz_{e_2}^{(3)} = 7$. Finally, the total number of floating-point comparisons performed in Algorithm 1 is 25. The number of floating-point multiplications performed to process synchronizing events 1 and 2 is $(N-1)(nz_{e_1}^{(1)} nz_{e_1}^{(2)} + nz_{e_2}^{(1)} nz_{e_2}^{(3)}) = 14$. When the global generator is stored in sparse format, the total number of floating-point comparisons performed by the straightforward algorithm that finds the NCD CCs is 57, which is almost two times as large as the corresponding value of Algorithm 1.

4.4 Numerical experiments with the NCD partitioning algorithm

We implemented the SC algorithm (see appendix 8.3) and Algorithm 1 in C++ [66] as part of the software package PEPS [54]. We ran all the experiments on a SUN UltraSparcstation 10 with 128 MBytes of RAM. To verify the NCD partitionings obtained for a given SAN, we compared our results with the straightforward approach of generating in core the submatrix of Q corresponding to the essential subset of states obtained using the SC algorithm and finding its NCD CCs. We remark that the same data structure for NCD CCs is used in Algorithm 1 and the straightforward approach.

The input parameters of Algorithm 1 are the user specified decomposability parameter ϵ, the vector output by the SC algorithm in which states corresponding to the essential subset of interest are marked, and a file in PEPS format that contains the description of the SAN under consideration. The only modification we introduced to the PEPS descriptor file format is a matrix appended to the end which describes the dependency among automata. This matrix is in the same sparse matrix format used for automata matrices in PEPS. Note that the new descriptor file format is compatible with the previous version of PEPS. We remark that the first step of Algorithm 1 (i.e., $Q \to P$ transformation) does not introduce any explicit changes to the original input description of the SAN. In other words, ϵ is multiplied by α once and rates are compared with $\alpha\epsilon$ on the fly. Hence, upon termination of the algorithm, the description of the SAN remains unchanged and can be used in further processing. The only modification that we make on the SAN is the transformation of each synchronizing event to the simple form (if the SAN is not already in that form). Note that this transformation is taken into account in the reported results.

Table 4.1: Results of the resource sharing problem (U, S).

n	n_{ess}	nz_{ess}	SC	ϵ	$\lvert CCs \rvert$	NCD_S	Gen.	NCD_N
32,768	16,384	210,664	0.69	0.04	1	0.46	0.61	0.04
(15,7)				0.08	16,384	0.44		0.04
65,536	39,203	563,491	1.57	0.04	1	1.06	1.64	0.13
(16,8)				0.08	39,203	1.03		0.12
131,072	65,536	960,858	3.29	0.04	1	2.15	3.06	0.22
(17,8)				0.08	65,536	2.12		0.21
262,144	155,382	2,514,678	7.24	0.04	1	4.96	8.02	0.57
(18,9)				0.08	155,382	4.83		0.55
524,288	262,144	4,319,100	15.31	0.04	1	9.76	15.03	0.85
(19,9)				0.08	262,144	9.83		0.91
1,048,576	616,666	11,102,426	33.43	0.04	1	22.03		
(20,10)				0.08	616,666	22.31		
2,097,152	1,048,576	19,188,796	70.58	0.04	1	44.98		
(21,10)				0.08	1,048,576	45.45		
4,194,304	2,449,868	48,587,212	152.56	0.04	1	100.79		
(22,11)				0.08	2,449,868	101.71		
8,388,608	4,194,304	84,438,360	319.53	0.04	1	205.24		
(23,11)				0.08	4,194,304	206.98		

As test problems, we use three SAN models in appendices 8.2.1, 8.2.2, and 8.2.3. We name them resource sharing, three queues, and mass storage. The SAN model of the resource sharing problem has single dependency set. In our experiments, we used $\lambda_i = 0.04$ and $\mu_i = 0.03$ for $i = 1, 2, \ldots, U$ (see appendix 8.2.1 for detailed description of the parameters of the SAN model). The three queues problem has two dependency sets, $\mathcal{D}_1 = \{\mathcal{A}^{(1)}, \mathcal{A}^{(3_1)}, \mathcal{A}^{(3_2)}\}$ and $\mathcal{D}_2 = \{\mathcal{A}^{(1)}\}$. In our experiments, we used $\lambda_1 = 0.4$, $\lambda_2 = 0.3$, $\mu_1 = 0.6$, $\mu_2 = 0.5$, $\mu_{3_1} = 0.7$, and $\mu_{3_2} = 0.2$. Detailed description of the three queues problem can be found in appendix 8.2.2. The SAN model of the mass storage system (see appendix 8.2.3) has three dependency sets, $\mathcal{D}_1 = \{\mathcal{A}^{(\mathrm{n}_1)}, \mathcal{A}^{(\mathrm{n}_2)}, \mathcal{A}^{(\mathrm{n}_3)}\}$, $\mathcal{D}_2 = \{\mathcal{A}^{(\tilde{C})}\}$, and $\mathcal{D}_3 = \{\mathcal{A}^{(erl)}\}$. We used $R = 5$, $H = 0.95$, $L = 0.75$, and the values of the other parameters in [18] except C, n_1, n_2, n_3. The detailed description of the parameters appears in [18].

Results of experiments for the resource sharing, three queues, and mass storage problems are presented respectively in Tables 4.1, 4.2, and 4.3. All timing results are in seconds. In these tables, n denotes the number of states in the global state space of the particular SAN under consideration, n_{ess} denotes the number of states in the essential subset when the underlying MC is reducible, nz_{ess} denotes the number of nonzero elements in the submatrix of Q corresponding to the essential subset of states, and SC denotes the time for state classification. For each problem, we indicate in parentheses under n the values of the integer parameters used. The column ϵ denotes the value of the decomposability parameter used and $|CCs|$ denotes the number of NCD CCs corresponding to ϵ when transient states are removed. The column NCD_S contains timing results for Algorithm 1. The columns Gen. and NCD_N respectively contain timing results to generate in core the submatrix of Q corresponding to the essential subset of states and to naively compute its NCD partitioning for given ϵ after the SC algorithm is executed. We have varied the value of ϵ in each problem to see how the performance of Algorithm 1 changes for different number of NCD CCs.

We remark that the difference between the time required to generate in core the submatrix of Q corresponding to the essential subset of states for a given SAN and the time to find the corresponding NCD partitionings using Algorithm

1 is noticeable. Compare columns Gen. and NCD_S in Tables 4.1–4.3, and also compare the sum of columns Gen. and NCD_N with column NCD_S. Moreover, there are cases in each of the three tables for which it is not possible to generate in core the submatrix of Q corresponding to the essential subset of states on the particular architecture. Hence, the straightforward approach of finding NCD partitionings is relatively more restricted with memory and is slower than using Algorithm 1.

Table 4.2: Results of the three queues problem (C_1, C_2, C_3).

n	n_{ess}	nz_{ess}	SC	ϵ	$\|CCs\|$	NCD_S	Gen.	NCD_N
37,800	20,160	112,242	0.44	0.10	1	0.06	0.21	0.02
(12,14,15)				0.22	364	0.12		0.04
				0.25	2,520	0.07		0.04
				0.35	20,160	0.05		0.04
68,850	36,720	207,279	0.82	0.10	1	0.10	0.39	0.05
(18,17,15)				0.22	544	0.22		0.09
				0.25	4,590	0.13		0.07
				0.35	36,720	0.11		0.06
202,400	106,260	608,474	2.63	0.10	1	0.30	1.24	0.17
(23,22,20)				0.22	924	0.68		0.28
				0.25	10,120	0.38		0.24
				0.35	106,260	0.33		0.23
390,000	202,800	1,168,676	4.91	0.10	1	0.56	2.40	0.32
(26,24,25)				0.22	1,200	1.37		0.54
				0.25	15,600	0.71		0.47
				0.35	202,800	0.58		0.45
756,000	390,600	2,264,460	9.83	0.10	1	1.03	4.58	0.62
(30,28,30)				0.22	1,652	2.46		1.04
				0.25	25,200	1.37		0.92
				0.35	390,600	1.12		0.90
1,414,875	727,650	4,239,795	19.04	0.10	1	1.88	8.37	1.16
(35,33,35)				0.22	2,277	4.60		1.94
				0.25	40,425	2.46		1.71
				0.35	727,650	2.02		1.56
4,050,000	2,070,000	12,143,950	56.91	0.10	1	5.02		
(40,50,45)				0.22	4,200	12.66		
				0.25	90,000	6.74		
				0.35	2,070,000	5.53		
6,875,000	3,506,250	20,632,250	96.37	0.10	1	8.63		
(50,55,50)				0.22	5,445	21.85		
				0.25	137,500	11.57		
				0.35	3,506,250	9.18		
9,150,625	4,658,500	27,445,825	131.34	0.10	1	11.25		
(55,55,55)				0.22	5,995	33.04		
				0.25	166,375	14.24		
				0.35	4,658,500	12.44		

The time spent for state classification does not involve any floating-point operations, whereas the time spent to generate in core the submatrix of Q corresponding to the essential subset of states primarily involves floating-point arithmetic operations. The overhead associated with evaluating functions slows down both tasks dramatically. Compare columns SC and Gen. in Tables 4.1–4.3 with columns NCD_S and NCD_N. The time spent by the SC algorithm is larger than the time spent by Algorithm 1 in all experiments. This is not surprising since the former is based on finding SCCs while the latter is based on finding CCs. The difference is more pronounced when there are multiple dependency sets for which Algorithm 1 can bring in considerable savings.

The resource sharing problem is the most difficult of the three problems considered since it has a single dependency set, is reducible, and contains a significant number of functional transitions. Hence, the time to find its NCD CCs using Algorithm 1 is the largest for a given problem size. Compare column NCD_S in Table 4.1 with those in Tables 4.2 and 4.3. However, even for this problem, Gen. is larger than NCD_S since we are able to take advantage of the constant transition values in automata matrices which makes Algorithm 1 worthwhile to use.

The case of $|CCs| = 1$ corresponds to smaller ϵ and implies the largest number of nonzeros taken into account from automata matrices in Algorithm 1 and from the submatrix of Q corresponding to the essential subset of states in the naive NCD partitioning algorithm. The case of $|CCs| = n_{ess}$ corresponds to larger ϵ and implies larger temporary data structures being used by both algorithms when determining NCD CCs. Hence, for increasing ϵ, the results in columns NCD_S and NCD_N either increase then decrease (Table 4.2) or they decrease then increase (Table 4.3). NCD_S for intermediate ϵ values for the mass storage example seem to have benefited significantly from its larger number of dependency sets, irreducibility, and the improvements introduced by possibilities (ii) and (iii) discussed in subsection 4.2.3.

Table 4.3: Results of the mass storage problem (C, N_1, N_2, N_3).

$n(= n_{ess})$	nz_{ess}	SC	ϵ	$\lvert CCs\rvert$	NCD_S	Gen.	NCD_N
30,240	199,440	0.46	0.01	1	0.05	0.43	0.05
(10,12,14,12)			0.05	5	0.03		0.05
			0.15	10	0.02		0.05
			0.22	720	0.03		0.05
			0.25	22,680	0.04		0.06
			0.40	30,240	0.05		0.08
69,120	462,720	1.08	0.01	1	0.13	1.05	0.12
(10,16,18,16)			0.05	5	0.07		0.12
			0.15	10	0.06		0.13
			0.22	960	0.07		0.14
			0.25	56,160	0.14		0.16
			0.40	69,120	0.14		0.15
184,320	1,232,640	2.95	0.01	1	0.16	2.78	0.35
(38,16,16,16)			0.05	5	0.11		0.33
			0.15	10	0.11		0.37
			0.22	2,880	0.12		0.34
			0.25	149,760	0.27		0.41
			0.40	184,320	0.30		0.43
372,680	2,524,060	6.21	0.01	1	0.45	6.12	0.67
(30,22,22,22)			0.05	5	0.27		0.66
			0.15	10	0.26		0.69
			0.22	3,080	0.33		0.69
			0.25	304,920	0.62		0.79
			0.40	372,680	0.71		0.85
760,000	5,130,000	13.09	0.01	1	0.45	12.35	1.38
(90,20,20,20)			0.05	5	0.37		1.33
			0.15	10	0.34		1.40
			0.22	7,600	0.58		1.32
			0.25	646,000	1.21		1.71
			0.40	760,000	1.22		1.82
1,572,160	10,773,920	28.90	0.01	1	2.19		
(35,34,34,34)			0.05	5	1.34		
			0.15	10	1.32		
			0.22	5,440	1.88		
			0.25	1,340,960	3.11		
			0.40	1,572,160	3.48		
3,510,000	23,985,000	65.93	0.01	1	2.08		
(126,30,30,30)			0.05	5	1.73		
			0.15	10	1.72		
			0.22	15,600	3.31		
			0.25	3,042,000	5.78		
			0.40	3,510,000	6.29		
5,573,750	38,220,000	108.92	0.01	1	3.53		
(126,35,35,35)			0.05	5	2.88		
			0.15	10	2.84		
			0.22	18,200	5.80		
			0.25	4,777,500	9.32		
			0.40	5,573,750	10.01		
9,280,000	63,800,000	189.80	0.01	1	5.91		
(140,40,40,40)			0.05	5	5.14		
			0.15	10	5.07		
			0.22	23,200	10.18		
			0.25	8,120,000	17.31		
			0.40	9,280,000	17.96		

4.5 Conclusion

In this chapter, we designed the NCD partitioning algorithm that computes NCD CCs of a SAN for a user specified decomposability parameter ϵ from the description of its components. The algorithm consists of three steps. Since the definition of NCDness is given through the transition probability matrix of a DTMC, the MC underlying a continuous-time SAN should be transformed to the discrete-time form using equation (4.1). In other words, in the first step the diagonal element with maximum magnitude of the SAN under consideration, α, is computed. In order to do it efficiently, we partition the automata of the SAN into dependency sets and find the maximum diagonal element of each dependency set. The diagonal element with maximum magnitude of the SAN is obtained as the sum of the maximum diagonal elements of the dependency sets.

In the second step, those synchronizing events that may influence the NCD partitioning of the SAN are selected. Synchronizing events influence the NCD partitioning in two ways. First, the transition rate of a simple synchronizing event is greater than or equal to $\alpha\epsilon$. Second, a set of synchronizing events contributes to the same nonzero element in the generator underlying the SAN and the sum of the transition rates of these synchronizing events is greater than or equal to $\alpha\epsilon$. The sets of selected events are named as potential sets.

The aim of the third step is to compute the NCD CCs using the description of the automata and the information obtained in steps 1 and 2. In subsection 4.2.3, we showed that it is relatively easy to find the NCD CCs of a SAN that does not have synchronizing events. Using the operator $\odot$, we gave a compact expression for the NCD CCs of a SAN. We pointed out that the operator $\odot$ is associative, and hence, NCD CCs of a SAN can be computed in a sequential manner starting from the last automaton. We also described how synchronizing events that form the potential sets can be taken into account. Finally, we discussed implementation issues for the case of cyclic dependencies.

The complexity analysis presented in section 4.3 shows that the time and space complexities of the NCD partitioning algorithm depends on the number of automata, the number of synchronizing events, the number of functions, the number of essential states, the sparsity of automata matrices, the number of dependency sets, and the ordering of automata. These results are verified with numerical experiments on three continuous-time SAN models. The experiments also showed that the developed algorithm performs much better than a straightforward algorithm that computes NCD CCs of a continuous-time MC from its transition rate matrix.

In the next chapter, we concentrate on finding the stationary probability vector of a MC modeled as a SAN. Taking advantage of the compact SAN description and the block structure induced by the tensor product, we derive conditions that allow us to identify lumpable SANs from the description of automata. For lumpable SANs, we introduce an iterative aggregation-disaggregation algorithm and discuss its implementation.

Chapter 5

SANs and lumpability

Various methods have been developed for solving the underlying MC of a SAN for its stationary probability vector, π. In this chapter we contribute to the existing results in this area. Continuing the research in [6] and [67], we take advantage of the block partitioning of the MC underlying a SAN model. First, we derive conditions for having equal row sums in the blocks of a matrix that is a sum of tensor products. Then, we extend the derived conditions to the descriptor of a SAN. We also introduce an efficient iterative aggregation-dissagregation algorithm for lumpable SANs

5.1 Block structure of the tensor product and its properties

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

$$A = \bigotimes_{k=1}^{N} A^{(k)}, \tag{5.1}$$

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 (see appendix 8.1), for any $m \in \{2, 3, \ldots, N\}$ the matrix A can be partitioned into K^2 blocks of the same order as

$$A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1K} \\ A_{21} & A_{22} & \ldots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \ldots & A_{KK} \end{pmatrix}, \tag{5.2}$$

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

$$A_{ij} = \xi_{ij} \bigotimes_{k=m}^{N} A^{(k)} = \left(\prod_{k=1}^{m-1} A^{(k)}(rA^{(k)}, cA^{(k)}) \right) \bigotimes_{k=m}^{N} A^{(k)}, \tag{5.3}$$

$i \leftrightarrow (rA^{(1)}, rA^{(2)}, \ldots, rA^{(m-1)})$, 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. Similar to the notation for the functional dependency among automata of a SAN (see Definition 2.1), 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 equation (5.2).

Theorem 5.1 *Each block A_{ij} in equation (5.2) has equal row sums for any $m \in \{2, 3, \ldots, N\}$ if the matrices $A^{(k)}$, $k = 1, 2, \ldots, N$, in equation (5.1) 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 equation (5.2) 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 and j, and u represents the column vector of 1's with appropriate length. The value $A^{(k)}(rA^{(k)}, cA^{(k)})$ in equation (5.3) 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, $\bigotimes_{k=m}^{N} A^{(k)}$ may very well depend on $rA^{(l)}$ for some $l \in \{1, 2, \ldots, m-1\}$.

Hence, it suffices to show that $(\xi_{ij} \bigotimes_{k=m}^{N} A^{(k)})u = l_{ij}u$ for some constant value l_{ij} that depends only on i and m. We are dropping m from l_{ij} since m is fixed for the chosen partitioning. By the assumption regarding equal row sums in the statement of the theorem, we have $A^{(k)}u = \nu^{(k)}u$ for $k = 1, 2, \ldots, N$ and for some constant value $\nu^{(k)}$ that depends only on k. Then

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

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 equation (5.2) under the assumed ordering of the matrices retains the equal row sums property, and $l_{ij} = \xi_{ij} \prod_{k=m}^{N} \nu^{(k)}$. □

Similar to the dependency graph of a SAN (see Definition 2.2) we associate the dependency graph $G(\mathcal{V}, \mathcal{E})$ with the matrices $A^{(k)}$, $k = 1, 2, \ldots, N$, in which the vertex $v_k \in \mathcal{V}$ represents $A^{(k)}$ and the edge $(v_k, v_l) \in \mathcal{E}$ 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.

Now, we state a more relaxed version of Theorem 5.1 for the case of cyclic functional dependencies.

Theorem 5.2 *There exists $m \in \{2, 3, \ldots, N\}$ and an ordering and renumbering of matrices $A^{(k)}$, $k = 1, 2, \ldots, N$, such that each block A_{ij}, $i, j = 1, 2, \ldots, K$, in equation (5.2) 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 $\mathcal{S}_1 = \{A^{(1)}, A^{(2)}, \ldots, A^{(m-1)}\}$ and $\mathcal{S}_2 = \{A^{(m)}, A^{(m+1)}, \ldots, A^{(N)}\}$. Let $\mathcal{S}_2$ be formed of cyclically dependent matrices (i.e., $N - m > 0$). Note that it is possible for the matrices in $\mathcal{S}_1$ to depend on $rA^{(k)}$, where $A^{(k)} \in \mathcal{S}_2$, or vice versa, but both type of functional dependencies cannot be present simultaneously. That is, the matrices in $\mathcal{S}_1$ and the matrices in $\mathcal{S}_2$ cannot be mutually dependent; otherwise we could not have two partitions $\mathcal{S}_1$ and $\mathcal{S}_2$. Now, let $\mathcal{S}_2[\mathcal{S}_1]$; 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 $\mathcal{S}_1[\mathcal{S}_2]$ were the case, one could exchange $\mathcal{S}_2$ and $\mathcal{S}_1$.

The equal row sums property of each block A_{ij} follows from two arguments. First, the scalars ξ_{ij} in equation (5.3) 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 $\mathcal{S}_1$ is formed of cyclically dependent matrices (implying $m > 1$), each ξ_{ij} is a well defined constant. Second, the matrices $(\xi_{ij} \bigotimes_{k=m}^{N} A^{(k)})$ in equation (5.3) 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 equation (5.2) has equal row sums for the particular value of m.

When there are $S > 2$ SCCs $\mathcal{S}_p$, $p = 1, 2, \ldots, S$, in the digraph associated with the matrices $A^{(k)}$, the theorem also holds since there are no cyclic functional dependencies among the $\mathcal{S}_p$ and they can be reordered and then renumbered so that for $p = 2, 3, \ldots, S$ $\mathcal{S}_p[\mathcal{S}_o]$ 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 equation (5.2) has equal row sums. $\Box$

Next, we state a result that extends Theorems 5.1 and 5.2 to a square matrix given as the sum of E tensor products.

Corollary 5.1 *If there exists the same value of m for which each tensor product $\bigotimes_{k=1}^{N} B_e^{(k)}$ in $B = \sum_{e=1}^{E} \bigotimes_{k=1}^{N} B_e^{(k)}$, where $B_e^{(k)}$ is of order n_k for $e = 1, 2, \ldots, E$, satisfies the conditions of Theorems 5.1 or 5.2, then each block B_{ij}, $i, j = 1, 2, \ldots, K$, in the partitioning of B specified by m as in equation (5.2) has equal row sums.*

When B is a stochastic or generator matrix that satisfies the conditions of Corollary 5.1, B is said to be lumpable [38, p. 124].

In the next section, we extend the conditions of having equal row sums in the blocks of a matrix that is a sum of tensor products to the descriptor of a continuous-time SAN. Then we introduce conditions for which the underlying MC of a SAN is lumpable.

5.2 Lumpable continuous-time SANs

Q in equation (2.1) 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 $\bar{Q}_e$. Q_l is the tensor sum of N matrices and can be written as a sum of tensor products as follows:

$$Q_l = \bigoplus_{k=1}^{N} Q_l^{(k)} = \sum_{k=1}^{N} I_{n_1} \otimes I_{n_2} \otimes \ldots \otimes I_{n_{k-1}} \otimes Q_l^{(k)} \otimes I_{n_{k+1}} \otimes \ldots \otimes I_{n_{N-1}} \otimes I_{n_N}, \quad (5.4)$$

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 5.1 applies through Theorem 5.2 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 $\bar{Q}_e$. We can omit $\bar{Q}_e$ from further consideration since $\bar{Q}_e$ 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 5.1. 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, assuming that the SAN description is proper (see Definition 2.5), 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 (see Remark 4.4). The next lemma shows that implicit functional dependencies are equivalent to explicit functional transitions.

Lemma 5.1 *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 $\mathcal{A}^{(t)}$ be the master automaton of synchronizing event 1. We denote by $\mathcal{Z}^{(k)}$ the set of states of $\mathcal{A}^{(k)}$, $k \neq t$, for which the corresponding rows of $Q_{e_1}^{(k)}$ are zeros. In order to obtain an equivalent SAN that does not contain implicit functional dependencies, we replace each nonzero element $Q_{e_1}^{(t)}(i,j)$ with the function

$$f(i,j) = \begin{cases} Q_{e_1}^{(t)}(i,j), & \text{for all } k, k \neq t, s\mathcal{A}^{(k)} \notin \mathcal{Z}^{(k)} \\ 0, & \text{otherwise} \end{cases} .$$

We also modify each $Q_{e_1}^{(k)}$, $k \neq t$, so that if $s\mathcal{A}^{(k)} \in \mathcal{Z}^{(k)}$, then $Q_{e_1}^{(k)}(s\mathcal{A}^{(k)}, s\mathcal{A}^{(k)})$ (which is 0) becomes 1. In the same way, we redefine the transitions in $\bar{Q}_{e_1}^{(t)}$ and $\bar{Q}_{e_1}^{(k)}$, $k \neq t$. 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 $Q_{e_j}^{(k)}$ and $\bar{Q}_{e_j}^{(k)}$ of each event $j \in \{1, 2, \ldots, E\}$ 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. $\square$

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

Definition 5.1 *A SAN that does not contain implicit functional dependencies is said to be in its explicit form.*

Definition 5.2 *A proper ordering of the automata of a SAN is a reverse topological ordering of its dependency graph.*

The reason behind using the reverse of the topological ordering in Definition 5.2

is the direction of the arcs we choose in G to represent dependencies between automata (see Definition 2.2).

When there are cyclic functional dependencies, a proper ordering of the automata of a SAN does not exist. In that case, using Definition 4.2 for the SCC digraph $G^{SCC}(\mathcal{V}^{SCC}, \mathcal{E}^{SCC})$ of the SAN, we search for a quasi-proper ordering of the automata of the SAN.

Definition 5.3 *A quasi-proper ordering of the automata of the SAN is a reverse topological ordering of its SCC digraph $G^{SCC}(\mathcal{V}^{SCC}, \mathcal{E}^{SCC})$ when $|\mathcal{V}^{SCC}| > 1$.*

From Definitions 5.1 and 5.2 follows the first part of the next proposition. From Definition 5.3 follows its second part. The theorem that follows the proposition specifies sufficient conditions for the lumpability of the generator of a continuous-time SAN.

Remark 5.1 *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 5.3 *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 $(|\mathcal{V}^{SCC}| - 1)$ lumpable partitionings, where $\mathcal{V}^{SCC}$ is introduced in Definition 5.3.*

Proof. Proof of this theorem follows from equation (2.1), Corollary 5.1, and Remark 5.1. 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 5.2 such that transitions in $\mathcal{A}^{(l)}$, $l = 1, 2, \ldots, m-1$, do not depend on $s\mathcal{A}^{(k)}$, $k = m, m+1, \ldots, N$. 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 $(|\mathcal{V}^{SCC}| - 1)$ 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 5.4 *Let $(v_1^{SCC}, v_2^{SCC}, \ldots, v_S^{SCC})$ be a quasi-proper ordering of a SAN in its explicit form as in Definition 5.3. Then the generator underlying the SAN is lumpable if there exists $s \in \{2, 3, \ldots, S\}$ such that each $\mathcal{A}^{(k)} \in \bigcup_{i=s}^{S} v_i^{SCC}$ satisfies one of the following conditions:*

(i) $\mathcal{A}^{(k)}$ is not the master of any synchronizing event;

(ii) if $\mathcal{A}^{(k)}$ is the master of synchronizing event t, then $Q_{e_t}^{(k)}$ has equal row sums;

(iii) if $\mathcal{A}^{(k)}$ is the master of synchronizing event t and $Q_{e_t}^{(k)}$ does not have equal row sums, then it must be that each automaton in $\bigcup_{i=1}^{s-1} v_i^{SCC}$ is not involved in event t.

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 $\bigcup_{i=s}^{S} v_i^{SCC}$ 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 $\bigcup_{i=s}^{S} v_i^{SCC}$. Hence, from Theorem 5.3, the generator underlying the SAN is lumpable and the m in its proof is equal to the smallest index of the automata in $\bigcup_{i=s}^{S} v_i^{SCC}$.

Now, let $\mathcal{A}^{(k)}$ satisfy condition (iii). Without loss of generality, let $Q_{e_t}^{(k)}$ 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 $\sum_{j=1,j\neq t}^{E} \bigotimes_{k=1}^{N} Q_{e_j}^{(k)}$ 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 $\tilde{Q} = \bigotimes_{k=1}^{N} Q_{e_t}^{(k)}$ specified by m have equal row sums. From equation (5.3), the ijth block of $\tilde{Q}$ is given by $\tilde{Q}_{ij} = (\prod_{k=1}^{m-1} Q_{e_t}^{(k)}(i_k, j_k)) \bigotimes_{k=m}^{N} Q_{e_t}^{(k)}$, 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 $Q_{e_t}^{(k)} = I_{n_k}$ for $k = 1, 2, \ldots, m-1$. Hence, for off-diagonal blocks, $i \neq j$ imply $\prod_{k=1}^{m-1} Q_{e_t}^{(k)}(i_k, j_k) = 0$. Consequently, each off-diagonal block in the partitioning of $\tilde{Q}$ 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 $\mathcal{A}^{(k)}$ has more than one synchronizing transition rate matrix with unequal row sums and to the case in which more than one automaton in $\bigcup_{i=s}^{S} v_i^{SCC}$ satisfies condition (iii) is immediate. $\square$

In the next subsection, we show that it is not difficult to apply Theorem 5.4 to a continuous-time SAN model.

5.2.1 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 (see appendix 8.2.3).

The SAN model of the mass storage problem consists of 5 automata and 3 synchronizing events. All local transition rate matrices have equal row sums of 0. Hence, we omit them from further consideration, but remark that transitions in $Q_l^{(\eta_2)}$ depend on the state of $\mathcal{A}^{(\eta_1)}$. Furthermore, $\mathcal{A}^{(\eta_1)}$ is not involved in the first two events, i.e., $Q_{e_1}^{(\eta_1)} = Q_{e_2}^{(\eta_1)} = I_{n_1}$. $\mathcal{A}^{(\eta_2)}$ is a slave automaton of event 1 and is not involved in the other two events. Observe that $Q_{e_1}^{(\eta_2)}$ has constant row sums of 1. $\mathcal{A}^{(\eta_3)}$ is a slave automaton of event 1 and $Q_{e_1}^{(\eta_3)}$ has constant row sums of 1. $\mathcal{A}^{(\eta_3)}$ acts as the master of event 2, and does not participate in event 3. $\mathcal{A}^{(\tilde{C})}$ is not involved in event 1 (i.e., $Q_{e_1}^{(\tilde{C})} = I_C$); but it is a slave automaton of events 2 and 3 such that $Q_{e_2}^{(\tilde{C})}$ and $Q_{e_3}^{(\tilde{C})}$ has constant row sums

Table 5.1: Summary information for the mass storage problem

Event	Master	Slave(s)	Dependencies
1	$\mathcal{A}^{(erl)}$	$\mathcal{A}^{(\eta_2)}$, $\mathcal{A}^{(\eta_3)}$	
2	$\mathcal{A}^{(\eta_3)}$	$\mathcal{A}^{(\tilde{C})}$	$\mathcal{A}^{(\eta_2)}[\mathcal{A}^{(\eta_3)}]$
3	$\mathcal{A}^{(\eta_1)}$	$\mathcal{A}^{(\tilde{C})}$	$\mathcal{A}^{(\eta_2)}[\mathcal{A}^{(\eta_1)}]$

of 1. Finally, $\mathcal{A}^{(erl)}$ is the master automaton of event 1; but it is not involved in the other two events.

Now, let us check the lumpability conditions of Theorem 5.4 using the information in Table 5.1 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 5.1, the digraph G of the SAN has the two edges (v_{η_2}, v_{η_3}) and (v_{η_2}, v_{η_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 $\mathcal{A}^{(\eta_2)}$ is placed after $\mathcal{A}^{(\eta_1)}$ and $\mathcal{A}^{(\eta_3)}$ is a proper ordering. Consider, for instance, the proper ordering $(\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3)}, \mathcal{A}^{(4)}, \mathcal{A}^{(5)})$, where 1 denotes erl, 2 denotes η_3, 3 denotes η_1, 4 denotes $\tilde{C}$, and 5 denotes η_2. For any $s \in \{2, 3, 4, 5\}$, the partitioning of the generator specified by s is lumpable as we next explain.

We first remark that $\mathcal{A}^{(5)}$ and $\mathcal{A}^{(4)}$ satisfy condition (i) of Theorem 5.4 meaning neither $\mathcal{A}^{(\eta_2)}$ nor $\mathcal{A}^{(\tilde{C})}$ is the master of any synchronizing event. This implies lumpability when $s \in \{4, 5\}$. Second, $\mathcal{A}^{(3)}$ satisfies condition (iii) implying lumpability when $s = 3$. This is because $\mathcal{A}^{(\eta_1)}$ is the master of synchronizing event 3, $Q_{e_3}^{(\eta_1)}$ does not have equal row sums, and $\mathcal{A}^{(i)}$, $i = 1, 2$, are not involved in synchronizing event 3. Similar to $\mathcal{A}^{(3)}$, $\mathcal{A}^{(2)}$ also satisfies condition (iii) implying lumpability when $s = 2$. In synchronizing event 2, $\mathcal{A}^{(\eta_3)}$ acts as the master, $Q_{e_2}^{(\eta_3)}$ does not have equal row sums, and $\mathcal{A}^{(1)}$ is not involved in synchronizing event 2. Thus, for the chosen proper ordering of automata, there are 4 lumpable partitionings of the generator for $s \in \{2, 3, 4, 5\}$.

In the next section we introduce the efficient iterative aggregation-disaggregation algorithm for lumpable SANs and discuss details of its implementation.

5.3 IAD algorithm for lumpable SANs

Assuming that the MC underlying a SAN model is lumpable with respect to the partition in (5.2) and is irreducible, we propose the following modified form of Koury-McAllister-Stewart's IAD algorithm [61] to compute its stationary probability vector, π. The proposed algorithm is different from the IAD algorithm of Koury-McAllister-Stewart in that the aggregation step is invoked only once and each subsequent iteration consists of the disaggregation step only.

IAD 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 $l_{ij} = \max(Q_{ij}u)$.

   (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

   $$z^{(it)} = (\tau_1 \frac{\pi_1^{(it-1)}}{\|\pi_1^{(it-1)}\|_1}, \tau_2 \frac{\pi_2^{(it-1)}}{\|\pi_2^{(it-1)}\|_1}, \ldots, \tau_K \frac{\pi_K^{(it-1)}}{\|\pi_K^{(it-1)}\|_1}).$$

   (b) Solve the K nonsingular systems of which the ith is given by

   $$\pi_i^{(it)} Q_{ii} = b_i^{(it)}$$

   for $\pi_i^{(it)}$, $i = 1, 2, \ldots, K$, where

   $$b_i^{(it)} = -\left(\sum_{j>i} z_j^{(it)} Q_{ji} + \sum_{j<i} \pi_j^{(it)} Q_{ji} \right).$$

4. Test $\pi^{(it)}$ for convergence. If the desired accuracy is attained, then stop and take $\pi^{(it)}$ as the stationary probability vector of Q. Else set $it = it + 1$ and go to step 3.

Here, we provide the algorithm for a continuous-time SAN. The algorithm for a discrete-time SAN given by equation (3.1) or (3.2) is straightforward and appears in appendix 8.6.

It is known that IAD exhibits fast convergence if the degree of coupling $\|F\|_\infty$ (see section 4.1) is small compared to 1. The study of local and global convergence of IAD (with the possibility of using iterative solution methods for the aggregation step) appear in [42] and [41].

The motivation behind proposing IAD rather than BGS [58, p.102] for lumpable SANs is that the lumped partitioning we consider is a balanced one with blocks of equal order, the aggregate matrix in IAD needs to be formed only once due to lumpability, and faster convergence will be achieved when $\|F\|_\infty$ is small. In [41], the convergence of a framework of IAD methods is studied. The IAD algorithm considered therein is different in that there is no requirement of a small degree of coupling in the partitioning. Furthermore, a number of relaxations (i.e., smoothings) of the power method kind is performed at the fine level. The authors prove that the errors at the fine and coarse levels are intimately related, and for a strictly positive initial approximation, the IAD approximation converges rapidly to the stationary probability vector as long as one is very precise in computing at the coarse level and a sufficiently high number of smoothings is performed at the fine level. Numerical results on randomly generated stochastic matrices with varying degrees of coupling and blocks of equal order, which are all tridiagonal, show that convergence is practically independent of the degree of coupling. See [20] for recent results on the computation of the stationary probability vector of a Markov chain.

Assuming that for the given ordering of automata there exists a lumpable partitioning and the indices of automata are ascending, the lumped matrix L is of order $K = \prod_{k=1}^{m-1} n_k$, where m is the smallest index among the automata that form the blocks of the partitioning (see equations (5.2) and (5.3)). In the IAD algorithm, L is computed at the outset and solved once for its stationary probability vector τ. As for the disaggregation phase (i.e., a BGS iteration), we need to look into how the right-hand sides $b_i^{(it)}$ at iteration it are computed. First, observe that the computation of $b_i^{(it)}$ involves only the off-diagonal blocks Q_{ij}, $i \neq j$. Hence, $\bar{Q}_e$ is omitted from the computation of $b_i^{(it)}$. Second, assuming that $(Q_e)_{ij}$ is the ijth block in the partitioning of Q_e as in equation (5.2), we have $(Q_e)_{ij} = \sum_{t=1}^{E} \xi_{ij}^{(t)} T_i^{(t)}$, where $\xi_{ij}^{(t)} = \prod_{k=1}^{m-1} Q_{e_t}^{(k)}(rQ_{e_t}^{(k)}, cQ_{e_t}^{(k)})$, $i \leftrightarrow (rQ_{e_t}^{(1)}, rQ_{e_t}^{(2)}, \ldots, rQ_{e_t}^{(m-1)})$, $j \leftrightarrow (cQ_{e_t}^{(1)}, cQ_{e_t}^{(2)}, \ldots, cQ_{e_t}^{(m-1)})$, and $T_i^{(t)} = \bigotimes_{k=m}^{N} Q_{e_t}^{(k)}$ (cf. equation (5.3)). Third, we have

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

for $i = 1, 2, \ldots, K$. Since $T_j^{(t)}$ is composed of $(N-m)$ tensor products, the vector-matrix multiplications $z_j^{(it)} T_j^{(t)}$ and $\pi_j^{(it)} T_j^{(t)}$ 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 $z_j^{(it)} T_j^{(t)}$ and $\pi_j^{(it)} T_j^{(t)}$ in the two summations appear in the computation of multiple $b_i^{(it)}$. 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 t, and then they can be scaled by $\xi_{ji}^{(t)}$ whenever necessary. Thus, when solving for $\pi^{(it)}$ in step 3(b) of the IAD, we first compute $\xi_{ji}^{(t)}$, check if it is nonzero, and only then multiply $z_j^{(it)}$ or $\pi_j^{(it)}$ with $T_j^{(t)}$ if this product was not computed before. With such an implementation, no more than one multiplication of $z_j^{(it)}$ or $\pi_j^{(it)}$ with $T_j^{(t)}$ 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 2 vectors of length n are used to store the previous and current approximations of the solution.

5.4 Conclusion

In this chapter, we derived lumpability conditions for MCs modeled as SANs. First, we identified the properties of a matrix that is a sum of tensor products of smaller matrices all of the same order. We showed that each block in the partitioning induced by the tensor representation has equal row sums if each of the matrices involved in the tensor products has equal row sums. We emphasized that when the matrices have functional elements, the ordering of the matrices in the tensor products is important. In the presence of functional elements, the matrices must be reverse topologically ordered with respect to the dependency graph G associated with the matrices. We also discussed the case of cyclic functional dependencies and showed that each block in the partitioning will have equal row sums if G has more than one SCC and all the matrices have equal row sums.

We extended results of section 5.1 to the descriptor of a continuous-time SAN and derived lumpability conditions for the MC underlying the SAN. For lumpable SANs, we introduced an efficient IAD algorithm and discussed its implementation. The introduced algorithm is a modified form of Koury-McAllister-Stewart's IAD algorithm and consists of two phases. In the aggregation phase, the lumped matrix is computed. The disaggregation phase is a BGS iteration on the transition matrix underlying the SAN. Since the MC is lumpable, the lumped matrix is solved for its stationary probability vector once. Hence, each iteration of IAD consists only of the disaggregation phase.

In the next chapter, we present results of experiments with IAD on various SAN models. We use the IAD algorithm to compute measures of interest for the discrete-time SAN model introduced in Chapter 3. We also compare performance of IAD with BGS on continuous-time SAN models and discuss the difficulties associated with solving lumpable SANs having unfavorable partitionings.

Chapter 6

Experiments with lumpable SANs

The goal of this chapter is to demonstrate application of the lumpability result discussed in the previous chapter and to compare performance of the introduced IAD algorithm with the other existing solvers.

In the next section, we present analysis of the discrete-time SAN model of the wireless ATM system designed in chapter 3. We discuss difficulties associated with the analysis of the SAN model using conventional techniques, show that the underlying MC is lumpable and employ the IAD algorithm for lumpable SAN to compute performance measures of interest. In section 6.2, we compare IAD with BGS on two continuos-time SAN models and in section 6.3 we show how lumpable SANs having unfavorable lumpable partiotionings can be analyzed efficiently.

6.1 Analysis of the discrete-time SAN model of the wireless ATM system

The discrete-time SAN model of the combined system (see section 3.4) has $(3+V)$ automata and 9 synchronizing events. The first 3 automata respectively have 2, $(C+1)$, $(B+1)$ states and the last V automata each have 4 states giving us a global state space size of $n = 2(C+1)(B+1)4^V$. The automaton $\mathcal{A}^{(3)}$ that models the data buffer depends on all the other automata, and each automaton $\mathcal{A}^{(k)}$, $k \in \{5, 6, \ldots, V+3\}$, that models VBR traffic depends on the automata $\mathcal{A}^{(4)}$,$\mathcal{A}^{(5)}$,$\ldots$,$\mathcal{A}^{(k-1)}$. The probability matrices associated with the automata are all relatively dense except the ones that correspond to the data buffer when $s\mathcal{A}^{(1)} = 0$.

Now, we are in a position to consider an example and discuss its implications. We set $\lambda = 0.1$, $S_C = 1$, $(p_{d0}, p_{d1}, p_{d2}, p_{d3}) = (0.05, 0.1, 0.25, 0.6)$ (amounting to an average of $\rho = 2.5$ packet arrivals during a TDMA frame), $(p_n, p_h, p_s) = C(5 \times 10^{-6}, 10^{-5}, 5 \times 10^{-6})$, $\lambda_v = 0.5$, $S_{Cv} = 10$, $(p_{empty}, p_{busy}) = (0.9, 0.1)$, $(p_{vn}, p_{vh}, p_{vs}) = V(5 \times 10^{-6}, 10^{-5}, 5 \times 10^{-6})$. For the problem $(C, V, B) = (8, 2, 15)$ with at most $2 (= M)$ CBR departures during a TDMA frame, we have $n = 4,608$ and $nz = 1,618,620$ (number of nonzeros larger than 10^{-16} is 1,174,657). Here nz denotes the number of nonzeros in the underlying DTMC. In the problem $(C, V, B) = (12, 3, 15)$ with $M = 3$, we have $n = 26,624$ and $nz = 39,042,922$ (number of nonzeros larger than 10^{-16} is 19,979,730). For the larger problem $(C, V, B) = (16, 4, 15)$ with $M = 4$, we have $n = 139,264$, but are not able to determine its number of nonzeros in a reasonable amount of time. Hence, in this problem, we not only have state space explosion, but we also have a relatively dense global DTMC hindering performance analysis by conventional techniques. However, the situation is not hopeless.

Observe that each transition probability matrix of each automaton of the SAN model have equal row sums (see description of the SAN model in sections 3.2 and 3.3). Note also that the automata of the SAN model can be reordered and then renumbered from 1 to N so that $\mathcal{A}^{(k)}[\mathcal{A}^{(l)}]$ for $k \in \{2, 3, \ldots, N\}$ imply $l \in \{1, 2, \ldots, k-1\}$. For our problem, such an ordering implies that the automaton corresponding to the data buffer be placed in the last position and the automata corresponding to VBR traffic be placed in any position other than the last as long as they are ordered according to increasing index among themselves. For instance, one possibility is $\mathcal{A}^{(2)}, \mathcal{A}^{(4)}, \mathcal{A}^{(5)}, \ldots, \mathcal{A}^{(V+3)}$, $\mathcal{A}^{(1)}, \mathcal{A}^{(3)}$. Hence, from equation (3.1) and Corollary 5.1, the SAN model of the ATM system is lumpable and there are $(N-1)$ lumpable partitionings for the given ordering of automata. In contrast to continuous-time SANs, the requirements of Corollary 5.1 are likely to be satisfied for a larger number of discrete-time SAN models. More importantly, the performance analyst has some flexibility in reordering the automata and the luxury of choosing m, which determines the partitioning.

6.1.1 Results of experiments

The IAD algorithm for lumpable discrete-time SANs (see appendix 8.6) that we name as IAD is implemented in C++ [66] as part of the software package PEPS [54]. We timed the solver on a Pentium III with 128 MBytes of RAM under Linux although the experimental framework in most problems could fit into 64 Mbytes. We order the automata as $\mathcal{A}^{(4)}, \mathcal{A}^{(5)}, \ldots, \mathcal{A}^{(V+3)}, \mathcal{A}^{(2)}, \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ and choose $m = N - 1$ for the partitioning in equation (5.2). Hence, each of the K nonsingular systems to be solved in step 3(b) of IAD is of order $2(B+1)$ and the lumped matrix to be solved in step 2 of the algorithm is of order $K = 4^V(C+1)$.

In each experiment, we use a tolerance of 10^{-8} on the approximate error $\|\pi^{(it)} - \pi^{(it-1)}\|_2$ in step 4 of the IAD algorithm. We remark that the approximate residual $\|\pi^{(it)} - \pi^{(it)}P\|_2$ turns out to be less than the approximate error upon termination in all our experiments. Furthermore, for all combinations of the integer parameters we considered, there is sufficient space to factorize in sparse format (that is, to apply sparse Gaussian elimination to) the K diagonal blocks in step 3(b) of IAD at the outset. Hence, we use sparse forward and back substitutions to solve the K nonsingular systems at each iteration of the algorithm. If this had not been the case, we would suggest using point Gauss-Seidel as discussed in [67] with a maximum number of 100 iterations and a tolerance of 10^{-3} on the approximate error for the K nonsingular systems at each iteration of the IAD algorithm (see [20]).

Regarding the solver for the lumped matrix formed in step 2 of the algorithm, we use Grassmann-Taksar-Heyman (GTH) method (see [19] and the references therein) when K is on the order of hundreds. When the lumped matrix is of considerable size and density with a number of nonzeros on the order of millions, we solve it using sparse IAD as discussed in [19] with a tolerance of 10^{-12} on its approximate error. In doing this, when the lumped matrix to be solved is NCD with a small degree of coupling [20], hence ill-conditioned, we employ an NCD partitioning. Otherwise we take advantage of the fact that the lumped matrix is also lumpable (see section 5.3) and try to use a balanced partitioning by separating the first $(N-2)$ automata in the chosen ordering into two subsets.

As for the performance measures of interest, the dropping probability of handover CBR calls is given by $P_{c_{drop}} = \|\pi_{s\mathcal{A}^{(2)}=C}\|_1$, whereas the blocking probability of new CBR calls is given by $P_{c_{block}} = \|\pi_{s\mathcal{A}^{(2)}\geq C-1}\|_1$. By the notation $\|\pi_{condition}\|_1$, we mean the sum of the stationary probabilities of all states that satisfy *condition*. Similarly, the dropping probability of handover VBR calls is given by $P_{v_{drop}} = \|\pi_{s\mathcal{A}^{(k)}\neq 0, \forall k\in\{4,5,\ldots,V+3\}}\|_1$, and the blocking probability of new VBR calls is given by $P_{v_{block}} = \|\pi_{s\mathcal{A}^{(k)}=0 \text{ for only one } k\in\{4,5,\ldots,V+3\}}\|_1$. Finally, the blocking probability of data packets is given by

$$P_a = P[0 \text{ empty slots}] + \frac{(p_{d2}+2p_{d3})P[1 \text{ empty slot}] + p_{d3}P[2 \text{ empty slots}]}{\rho},$$

where

$$\begin{aligned}
P[0 \text{ empty slots}] &= \|\pi_{\varphi(0)\,\wedge\, s\mathcal{A}^{(3)}=B}\|_1, \\
P[1 \text{ empty slot}] &= \|\pi_{(\varphi(1)\,\wedge\, s\mathcal{A}^{(3)}=B)\,\vee\,(\varphi(0)\,\wedge\, s\mathcal{A}^{(3)}=B-1)}\|_1, \\
P[2 \text{ empty slots}] &= \|\pi_{(\varphi(2)\,\wedge\, s\mathcal{A}^{(3)}=B)\,\vee\,(\varphi(1)\,\wedge\, s\mathcal{A}^{(3)}=B-1)\,\vee\,(\varphi(0)\,\wedge\, s\mathcal{A}^{(3)}=B-2)}\|_1,
\end{aligned}$$

and $\varphi(i)$ is true if $FS(0,0)=i$, otherwise, it is false.

We remark that $P_{c_{drop}}$ and $P_{c_{block}}$ are independent of ABR and VBR traffic. Similarly, $P_{v_{drop}}$ and $P_{v_{block}}$ are independent of ABR and CBR traffic. Hence, when $C=V=M$ and the real valued parameters for CBR and VBR traffic are the same, we have $P_{c_{drop}} = P_{v_{drop}}$ and $P_{c_{block}} = P_{v_{block}}$. In Figure 6.1, we set $(p_n, p_h, p_s) = C(5\times 10^{-6}, 10^{-5}, 5\times 10^{-6})$, $M=C$ and plot $P_{c_{block}}$ and $P_{c_{drop}}$ versus C. We verify that $P_{c_{block}}$ is larger than $P_{c_{drop}}$, and that larger C implies smaller $P_{c_{block}}$ and $P_{c_{drop}}$.

Next we consider the three problems $(C,V,B)\in\{(8,2,15),(12,3,15),$ $(16,4,15)\}$ that are respectively named *small*, *medium*, and *large* (see beginning of section 6.1). We set $(p_{d0},p_{d1},p_{d2},p_{d3}) = (0.05, 0.1, 0.25, 0.6)$, $(p_n,p_h,p_s) =$

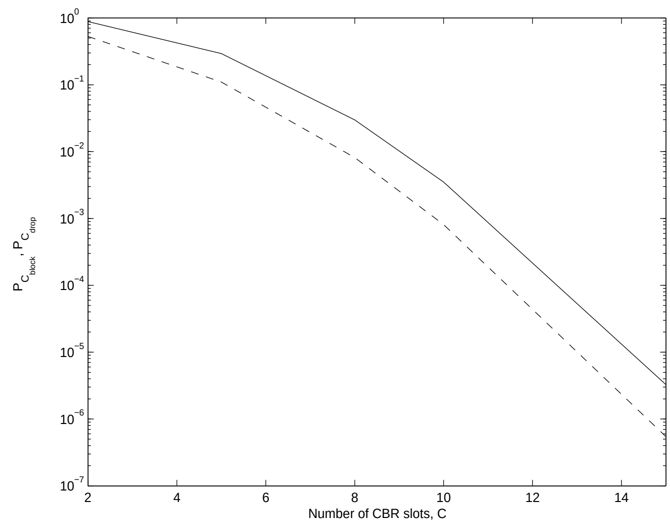
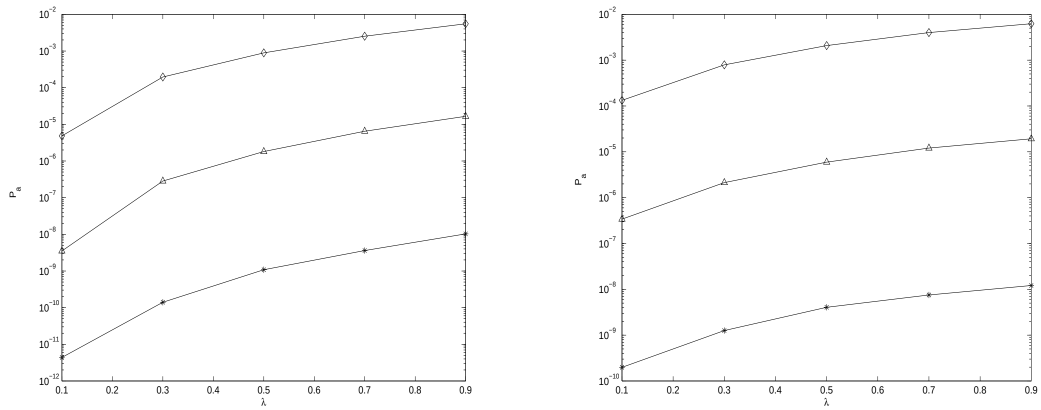


Figure 6.1: $P_{c_{block}}$ (solid) and $P_{c_{drop}}$ (dashed) vs C.

$C(5\times10^{-6}, 10^{-5}, 5\times10^{-6})$, $M = V$, and $(p_{vn}, p_{vh}, p_{vs}) = V(5\times10^{-6}, 10^{-5}, 5\times10^{-6})$. We choose (α, β) so as to satisfy $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and $S_C \in \{1, 10\}$ (see section 3.1). As for the VBR arrival processes, we set (α_v, β_v) so as to have $\lambda_v = 0.5$ and $S_{C_v} = 10$ (see section 3.3). Finally, we set $(p_{empty}, p_{busy}) = (0.9, 0.1)$ so that the rate of each VBR arrival process in the low intensity state is 10% that of its high intensity state. In Figure 6.2, we plot P_a versus λ for the problems *small*, *medium*, and *large* when (a) $S_C = 1$, (b) $S_C = 10$. We observe that P_a increases with λ and S_C though the increase with S_C happens slowly.

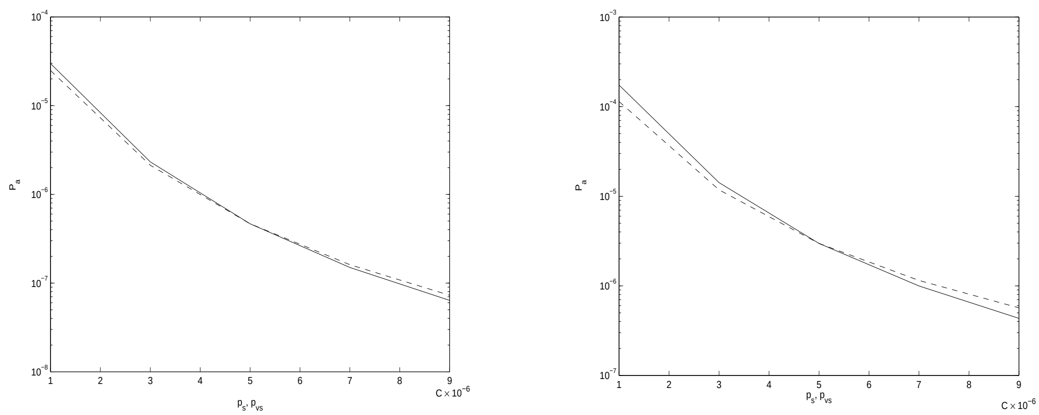
Note that the lumped matrices of the *small* problems are all the same. The same argument follows for the lumped matrices of the *medium* and *large* problems. This is simply because the parameters that we alter in the experiments of Figure 6.2 are only those of $s\mathcal{A}^{(1)}$, which happens to be among the last two automata in the chosen order of automata. Even though α and β change, the row sums of the blocks P_{ij} in equation (5.2) are the same because $P^{(1)}_{s_{ab}}u = u$ for all $a, b \in \{0, 1, 2\}$.

Since data packets can use VBR slots when a VBR connection is active but in the low intensity state, we expect P_a to depend weaker on p_{vs} than on p_s. To that effect, we consider the problem $(C, V, B) = (4, 4, 15)$. We set $(p_{d0}, p_{d1}, p_{d2}, p_{d3}) = (0.05, 0.1, 0.25, 0.6)$, $\lambda = 0.5$, and choose $S_C \in \{1, 10\}$.



(a) (b)

Figure 6.2: P_a vs λ for $(\diamond, \triangle, \star) = (small, medium, large)$ when (a) $S_C = 1$,(b) $S_C = 10$.



(a) (b)

Figure 6.3: P_a vs p_{vs} ($p_s = 5C \times 10^{-6}$, dashed), P_a vs p_s ($p_{vs} = 5V \times 10^{-6}$, solid) for $(C, V, B) = (4, 4, 15)$ and $\lambda = 0.5$ when (a) $S_C = 1$, (b) $S_C = 10$.

For the VBR arrival processes, we set (α_v, β_v) so as to have $\lambda_v = 0.5$ and $S_{Cv} = 10$, and we set $(p_{empty}, p_{busy}) = (0.9, 0.1)$. Furthermore, we set $(p_n, p_h) = (p_{vn}, p_{vh}) = C(5 \times 10^{-6}, \times 10^{-5})$, and $M = 4$. In Figure 6.3, we plot P_a versus $p_{vs} = iV \times 10^{-6}, i \in \{1, 3, 5, 7, 9\}$, for fixed $p_s = 5C \times 10^{-6}$ using a dashed curve, and we plot P_a versus $p_s = iC \times 10^{-6}, i \in \{1, 3, 5, 7, 9\}$, for fixed $p_{vs} = 5V \times 10^{-6}$ using a solid curve on the same graph when (a) $S_C = 1$, (b) $S_C = 10$.

In the last set of experiments, we again consider the problem $(C, V, B) = (4, 4, 15)$. We set $(p_{d0}, p_{d1}, p_{d2}, p_{d3}) = (0.05, 0.1, 0.25, 0.6)$, $\lambda = 0.5$, and choose $S_C \in \{1, 10\}$. We set $(p_n, p_h, p_s) = (p_{vn}, p_{vh}, p_{vs}) = C(10^{-6}, 5 \times 10^{-5}, 10^{-6})$,

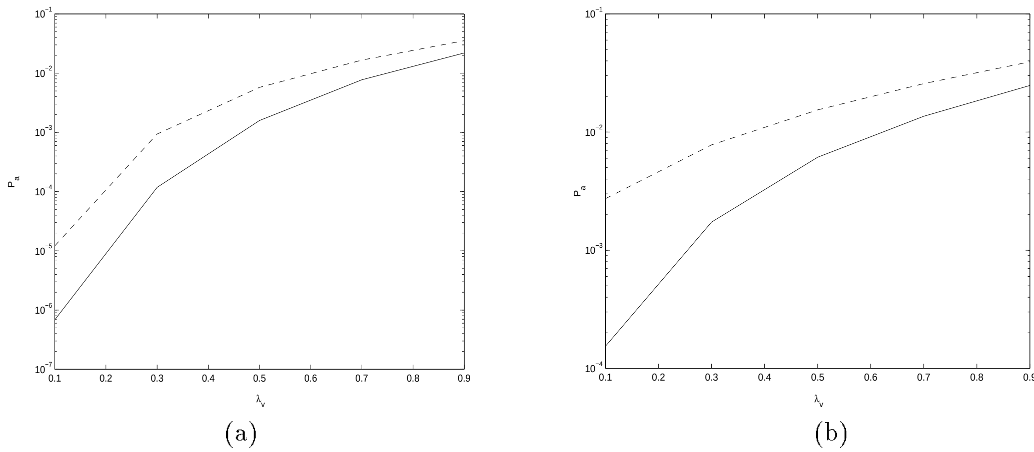


Figure 6.4: P_a vs λ_v when $\lambda = 0.5$, $S_C = 1$ (solid), $S_C = 10$ (dashed) for $(C, V, B) = (4, 4, 15)$ and $S_{C_v} = 10$ when (a) $(p_{empty}, p_{busy}) = (0.9, 0.1)$, (b) $(p_{empty}, p_{busy}) = (0.5, 0.5)$.

$M = 4$, and $S_{C_v} = 10$. In Figure 6.4, we plot P_a versus $\lambda_v \in \{0.1, 0.3, \ldots, 0.9\}$ when (a) $(p_{empty}, p_{busy}) = (0.9, 0.1)$, (b) $(p_{empty}, p_{busy}) = (0.5, 0.5)$. We observe that when the VBR arrival process behaves more like the CBR arrival process as in part (b), P_a is larger. Furthermore, the increase in P_a with respect to λ_v is smoother for larger S_C.

The underlying DTMCs of the SAN models in Figures 6.1, 6.3, and 6.4 are irreducible. Those of Figure 6.2 are reducible with a single subset of essential states. When a reducible discrete-time SAN has a single subset of essential states and each subset of the partition in equation (5.2) includes at least one essential state (which is the case in the problems of Figure 6.2), the lumped matrix computed in step 2 of the IAD algorithm is still irreducible. With such a partitioning, if one starts in step 1 with an initial approximation having zero elements corresponding to transient states, successive approximations computed by IAD will have zero elements corresponding to transient states as well. This simply follows from the fact that in step 3(a) the nonzero structure of z is the same as that of the previous approximation. Consequently, in step 3(b) each computed b_i has zero elements corresponding to transient states. Hence, the solutions of the K nonsingular systems at step 3(b) have zero elements corresponding to transient states.

In the problems of Figure 6.2, we start with a positive initial approximation just like the other problems and observe that all elements that correspond to transient states become zero at the second iteration in the *small* and *medium* problems and at the third iteration in the *large* problem. Once the elements that correspond to transient states in an approximate solution become zero, they remain zero by the argument in the previous paragraph. This is also confirmed experimentally. Hence, there is no need to run the time consuming SC algorithm (see appendix 8.3) for each of the 30 experiments in Figure 6.2, since the matrices in each of the three problems have the same nonzero structure.

Regarding the solution times of numerical experiments, we provide a representative group of results in Table 6.1 which are for the problems of Figure 6.2. We remark that among *small*, *medium*, and *large*, the DTMC of only the first can be stored on the target architecture (see parameters of the problems in section 6.1). The degree of coupling, $\|F\|_\infty$, associated with the partitioning in equation (5.2) for the three problems is respectively 0.9909, 0.9991, 0.9999 in four decimal digits of precision. Hence, none of the lumpable partitionings we consider in IAD is NCD. However, the smallest degree of coupling we find for each of the 10 *small* problems using the algorithm in [17] is on the order of 10^{-5}. The same value for each of the 10 *medium* problems is also on the order of 10^{-5}. We are not able to determine the value for the *large* problem due its order and density, but it is very likely that again we will have a value on the order of 10^{-5}. This all means that although the lumpable partitionings we consider for the problems in Figure 6.2 are not NCD partitionings, there exist highly NCD partitionings for each one, and therefore they are all very ill-conditioned. Nevertheless, we are fortunate that the IAD algorithm does not require NCD partitionings for convergence [41] as discussed in section 5.3.

We solve the lumped matrix ($n_L = 144$ and $nz_L = 7,644$) of the *small* problem using GTH in nearly 0 seconds. We solve the lumped matrix ($n_L = 832$ $nz_L = 188,094$) of the *medium* problem in 0.5 seconds and 16 iterations using sparse IAD with an NCD partitioning of 4 blocks (with orders varying between 117 and 351) and a degree of coupling on the order of 10^{-5}. We solve the lumped matrix ($n_L = 4,352$ and $nz_L = 3,980,512$) of the *large* problem in 48.7 seconds and 22 iterations using sparse IAD with an NCD partitioning of

16 blocks (with orders varying between 17 and 1,377) and a degree of coupling on the order of 10^{-4}. Hence, much larger problems than can fit explicitly into a given architecture may be solved by the proposed approach.

The reported times in Table 6.1 correspond to the iterative part of IAD and they exclude the time spent for solving L. Regarding the number of iterations taken by the IAD algorithm to convergence, the highest values are encountered when $\lambda \in \{0.7, 0.9\}$ and $S_C = 10$. They are iteration numbers greater than or equal to 35, and are for the cases in which α and β are highly unbalanced. As a result, the corresponding solution times are considerably larger than other combinations of λ and S_C. If we exclude these six cases, the *small* problem can be solved within 12 seconds, the *medium* problem within 169 seconds, and the *large* problem within 4,430 seconds. On the other hand, the smallest time to obtain a solution for the *small*, *medium*, and *large* problems is respectively 8, 77, and 851 seconds. In general, the solution times are very satisfactory if we keep in mind the number of nonzeros of the underlying DTMC.

Table 6.1: Timing results in seconds and # of IAD iterations for Figure 5.2

Problem	S_C	$\lambda = 0.1$		$\lambda = 0.3$		$\lambda = 0.5$		$\lambda = 0.7$		$\lambda = 0.9$	
		Time	#*it*	Time	#*it*	Time	#*it*	Time	#*it*	Time	#*it*
small	1	8	18	9	21	8	17	9	20	11	27
	10	8	18	8	19	8	19	15	35	38	89
medium	1	77	9	85	10	85	10	86	11	138	18
	10	146	19	168	22	94	12	309	41	824	110
large	1	889	4	990	5	803	4	1,742	9	3,622	19
	10	3,813	20	4,381	23	1,005	5	8,531	45	23,119	122

6.2 Results of experiments with lumpable continuous-time SANs

Similar to the IAD algorithm for discrete-time SANs, we implemented IAD algorithm for continuos-time SANs in C++ [66] as part of the software package PEPS [54]. We timed the solver on a Pentium III with 128 MBytes 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 IAD with BGS. We use the recursive implementation of BGS for SANs as discussed in [67]. In order to provide a fair comparison, IAD 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 (see appendix 8.2.3). We use its four instances in [67] that we number from 1 to 4. The integer parameters of these four problems are given in Table 6.2. Parameter C denotes the number of states in $\mathcal{A}^{(\tilde{C})}$, n_i denotes the number of states in $\mathcal{A}^{(\eta_i)}$, $i = 1, 2, 3$, and $\mathcal{A}^{(erl)}$ has 5 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.

Table 6.2: Integer parameters of the mass storage and the modified three queues SAN models

	mass storage				three queues (modified)			
Prob	C	N_i	n	nz	C_i	n	n_{ess}	nz_{ess}
1	26	6	6,480	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

In the first set of experiments, the automata are ordered as $\mathcal{A}^{(erl)}$, $\mathcal{A}^{(\eta_3)}$, $\mathcal{A}^{(\eta_1)}$, $\mathcal{A}^{(\tilde{C})}$, $\mathcal{A}^{(\eta_2)}$. As we indicated in subsection 5.2.1, there are 4 lumpable partitionings for this proper ordering of automata. We partition the automata

Table 6.3: Results of experiments with the mass storage problem, first ordering

Prob	*Solver*	*it#*	*time*	*dbgen*	*Lgen*	*Lsolve*	*perit*
1	BGS	102	2.59	0.04			0.03
	IAD	34	1.00	0.04	0.03	0.00	0.03
2	BGS	106	44.79	0.68			0.42
	IAD	40	13.03	0.68	0.02	0.00	0.31
3	BGS	201	417.98	8.53			2.03
	IAD	47	75.01	8.53	0.06	0.12	1.41
4	BGS	323	1,932.39	42.25			5.85
	IAD	58	303.16	42.25	0.14	0.39	4.49

as $\mathcal{A}^{(erl)}$, $\mathcal{A}^{(\eta_3)}$, $\mathcal{A}^{(\eta_1)} \mid \mathcal{A}^{(\tilde{C})}$, $\mathcal{A}^{(\eta_2)}$ 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 only once. Then the computed LU factors are used at each iteration to solve the K nonsingular systems in step 3(b) of the IAD algorithm.

The results of the first set of experiments are given in Table 6.3. Column *it#* gives the number of iterations performed till convergence, *time* gives the total time to solve the problem, *dbgen* gives the time to generate and factorize diagonal blocks at the outset, *Lgen* gives the time to generate the lumped matrix L, *Lsolve* 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 $(time - (Lgen + Lsolve + dbgen))/(it\#)$. Note that for BGS, columns *Lgen* and *Lsolve* are naturally zero.

In the first problem, L is stored as a two-dimensional matrix and solved using the GTH method (see [19]). In the last three problems, L is of order 605, 1,280 and 2,205, respectively. Hence, it is more feasable to store L in sparse format and solve it using IAD with a balanced partitioning (if possible) having a small degree of coupling (see [19, 20]). 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 5 blocks of equal order.

Even though step 3(a) of IAD does not exist in BGS, the experiments with the particular ordering and partitioning of automata show that time per iteration in IAD is smaller than that in BGS due to the gain obtained from computing the products $z_j^{(it)}Q_{ji}$ and $\pi_j^{(it)}Q_{ji}$ once at the expense of some storage space as discussed in subsection 5.3. Furthermore, IAD 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 IAD is considerably smaller than that with BGS although there is extra work associated with forming and solving the aggregated system.

Table 6.4: Results of experiments with the mass storage problem, second ordering

Prob	*Solver*	*it#*	*time*	*dbgen*	*Lgen*	*Lsolve*	*perit*
1	BGS	33	1.28	0.04			0.04
	IAD	8	0.46	0.04	0.03	0.03	0.05
2	BGS	25	14.88	0.35			0.58
	IAD	7	6.94	0.35	1.14	0.76	0.67
3	BGS	23	70.63	1.49			3.01
	IAD	7	42.74	1.49	10.34	5.85	3.58
4	BGS	30	293.39	4.48			9.63
	IAD	7	236.25	4.48	129.14	27.10	10.79

In the second set of experiments with the mass storage problem, the automata are ordered as $\mathcal{A}^{(erl)}$, $\mathcal{A}^{(\tilde{C})}$, $\mathcal{A}^{(\eta_1)}$, $\mathcal{A}^{(\eta_3)}$, $\mathcal{A}^{(\eta_2)}$. Observe that for this ordering, there are only 2 lumpable partitionings of the generator which are given by $\mathcal{A}^{(erl)}, \mathcal{A}^{(\tilde{C})}, \mathcal{A}^{(\eta_1)}, \mathcal{A}^{(\eta_3)} \mid \mathcal{A}^{(\eta_2)}$ and $\mathcal{A}^{(erl)} \mid \mathcal{A}^{(\tilde{C})}, \mathcal{A}^{(\eta_1)}, \mathcal{A}^{(\eta_3)}, \mathcal{A}^{(\eta_2)}$. Furthermore, the latter partitioning has blocks of 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 6.4 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 1,080, 6,655, 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 the IAD algorithm, we use the optimized recursive BGS implementation discussed in [67] rather than the implementation described in section 5.3, 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 IAD. Together with the fact that there is overhead associated with step 3(a) of the IAD algorithm, this implies slightly larger time per iteration in IAD than in BGS.

In the experiments of Table 6.4, both solvers converge in a smaller number

Table 6.5: Results of experiments with the modified version of the three queues problem

C_i	*Solver*	*it#*	*time*	*SC*	*dbgen*	*Lgen*	*Lsolve*	*perit*
20	BGS	341	189.82	1.31	6.59			0.53
	IAD	180	75.08	1.31	6.59	0.02	0.10	0.37
25	BGS	404	585.20	3.22	24.89			1.39
	IAD	201	209.43	3.22	24.89	0.03	0.25	0.90
30	BGS	456	1,312.05	6.78	72.70			2.70
	IAD	221	483.73	6.78	72.70	0.06	0.65	1.85
35	BGS	502	2,864.87	12.70	185.75			5.31
	IAD	241	1,017.49	12.70	185.75	0.09	1.21	3.39

of iterations when compared with the results of the first set of experiments. Nevertheless, it is not surprising to see that IAD 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 IAD is the three queues problem that appears in appendix 8.2.2. In that model, 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 the three queues problem in which customers of both types are lost when queue 3 is full, i.e., $Q_{e_1}^{(3_1)}$ has the same functional nonzero structure as $Q_{e_2}^{(3_2)}$ (see the description of the SAN model in appendix 8.2.2). In our experiments, we use the real valued parameters in [67].

We use four instances of the modified version of the three queues problem and number them from 1 to 4. The integer parameters are given in Table 6.2. 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 SC algorithm (see appendix 8.3) to classify the states into essential and transient subsets. Columns n_{ess} and nz_{ess} respectively give the number of essential 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 essential states and 0 for transient states, thereby enabling the identification of the subset of essential states in advance. See [22] for example SAN models and their reachability functions. In any case, once the essential 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 IAD.

The SAN model of the modified version of the three queues problem is in its explicit form. There are functional transitions in synchronizing transition probability matrices of $\mathcal{A}^{(3_1)}$ and $\mathcal{A}^{(3_2)}$. Functional transitions of $\mathcal{A}^{(3_1)}$ depend on the state of $\mathcal{A}^{(3_2)}$ and those in $\mathcal{A}^{(3_2)}$ depend on the state of $\mathcal{A}^{(3_1)}$ implying a cyclic dependency. Hence, a proper ordering of the automata in this SAN does not exist. We consider the quasi-proper ordering $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$, $\mathcal{A}^{(3_1)}$, $\mathcal{A}^{(3_2)}$, which has two lumpable partitionings given by $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)} \mid \mathcal{A}^{(3_1)}$, $\mathcal{A}^{(3_2)}$ and $\mathcal{A}^{(1)} \mid \mathcal{A}^{(2)}$, $\mathcal{A}^{(3_1)}$, $\mathcal{A}^{(3_2)}$. 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 $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)} \mid \mathcal{A}^{(3_1)}$, $\mathcal{A}^{(3_2)}$, 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 1,225, 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 6.5. Time spent for state classification is negligible (see column SC). The values in column *time* include those in SC. Numerical results show that IAD converges in a smaller number of iterations than BGS. Furthermore, time per iteration in IAD is smaller than that in BGS again due to the balanced nature of the partitioning. Finally, solution time with IAD is less than half of that with BGS in all experiments.

In the next section, we discuss difficulties associated with the analysis of SANs having relatively large blocks in lumpable partitionings, and we experiment with two SAN models having such unfavorable partitionings.

6.3 Lumpable SANs with unfavorable partitionings

The IAD algorithm introduced for SANs works only with lumpable partitionings. Obviously, the number of lumpable partitionings of a particular SAN model is limited. Moreover, the existing lumpable partitionings of a SAN may have relatively large blocks which is unfavorable for IAD. At each iteration in step 3(b) of IAD, K systems of linear equations each of order equal to the order of blocks in the given partitioning are solved. When a lumpable partitioning of a SAN has blocks of relatively small size, it is more reasonable to generate all diagonal blocks once, compute the LU factors of each block and store them in core memory. On the other hand, generation of large blocks is unfavorable due to two reasons. First, the generation of large diagonal blocks is almost equivalent to the generation of the underlying MC and hence, may take significant amount of time. Second, the size of core memory may be insufficient to store all the diagonal blocks.

As a remedy for unfavorable lumpable partitionings, BGS may be employed to solve the K nonsingular systems in step 3(b) of the IAD algorithm. Similar to block A_{ij} defined in equation (5.2), each diagonal block of the given partitioning of the MC underlying a SAN can be expressed as a sum of tensor products. Hence, if m in equation (5.2) is less than N, each diagonal block of the partitioning can be partitioned into subblocks as in equation (5.2) as well. In other words, the recursive implementation of BGS discussed in [67] can be used to solve the diagonal blocks of the partitioning. Thus, the diagonal blocks can be partitioned into subblocks so that all diagonal subblocks of the diagonal blocks can be generated and factorized once, and their LU factors can be stored in core memory.

In the next subsection, we present the results of numerical experiments [32] with IAD and BGS on two continuous-time SAN models having unfavorable lumpable partitionings.

Table 6.6: Integer parameters of the three queues and pushout SAN models

three queues				pushout			
C_i	n	n_{ess}	nz_{ess}	$B+1$	n	n_{ess}	nz_{ess}
25	390,625	203,125	1,170,625	250	125,000	62,750	374,500
30	810,000	418,500	2,482,200	500	500,000	250,500	1,499,000
35	1,500,625	771,750	4,499,425	750	1,125,000	563,250	3,373,500

6.3.1 Results of experiments

The framework of the experiments presented in this subsection is the same as that in section 6.2. In particular, the solver is timed on a Pentium III with 128 MBytes of RAM under Linux; the stopping tolerance on the norm of the difference between consecutive approximations is 10^{-8}; the running time of IAD is compared with the recursive implementation of BGS for SANs as discussed in [67]; IAD and BGS both use the same ordering of automata and the same routines to generate and solve the diagonal blocks of the partitioning. The timing results presented in Tables 6.7 and 6.8 are in seconds.

The first problem we use to test IAD is the three queues problem (see appendix 8.2.2). The SAN model of the three queues problem has functional transitions in synchronizing transition probability matrices of $\mathcal{A}^{(3)}$ and $\mathcal{A}^{(4)}$. Functional transitions of $\mathcal{A}^{(3)}$ depend on the state of $\mathcal{A}^{(4)}$ and those in $\mathcal{A}^{(4)}$ depend on the state of $\mathcal{A}^{(3)}$ implying a cyclic dependency. Hence, a proper ordering of the automata in this SAN does not exist. We consider the quasi-proper ordering $\mathcal{A}^{(2)}$, $\mathcal{A}^{(1)}$, $\mathcal{A}^{(3)}$, $\mathcal{A}^{(4)}$, which has a single lumpable partitioning given by $\mathcal{A}^{(2)} \mid \mathcal{A}^{(1)}, \mathcal{A}^{(3)}, \mathcal{A}^{(4)}$.

In our experiments, we use the real valued parameters in [67]. The integer parameters are given in Table 6.6. In the first set of experiments with the three queues problem 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 SC algorithm to classify the states into essential and transient subsets. Columns n_{ess} and nz_{ess} respectively give the number of essential states and the number of nonzero elements in the corresponding submatrix of the generator. Once the essential 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 IAD.

The diagonal blocks in the lumpable partitioning of the three queues problem are of order $C_1C_3^2$ and the lumped matrix is of order C_2. At each iteration of IAD, we use BGS to solve the large diagonal blocks with a stopping tolerance of 10^{-5} on the norm of the difference between consecutive approximations. We experiment with two partitionings of the diagonal blocks: $\mathcal{A}^{(2)} \mid \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ $[\mathcal{A}^{(4)}]$ and $\mathcal{A}^{(2)} \mid \mathcal{A}^{(1)}$ $[\mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$. Here, the square brackets separate the automata that form the subblocks. In the experiments with BGS, we use the two partitionings $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ $[\mathcal{A}^{(4)}]$ and $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}$ $[\mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$, where the square brackets separate the automata that form the blocks. We do not experiment with the partitioning $\mathcal{A}^{(2)}$ $[\mathcal{A}^{(1)}, \mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$. First, this partitioning is the same as the lumpable partitioning used for IAD. Hence, an iteration of IAD would be different than an iteration of BGS only in the disaggregation step 3(a) which takes negligible amount of time. Second, the lumped matrix of this partitioning is small and can be generated and solved (see step 2 of the IAD algorithm) almost in no time. Third, for the same partitioning, IAD always converges in a smaller number of iterations than BGS. In both of the partitionings for BGS as well as in the two partitionings for IAD, the blocks input to the BGS solvers are generated and factorized once in the outset and the LU factors are stored in core memory.

The results of experiments for the three queues problem with different partitionings of Q for BGS and different partitionings of the diagonal blocks for IAD are presented in Table 6.7. IAD with the partitioning $\mathcal{A}^{(2)} \mid \mathcal{A}^{(1)}$ $[\mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$ converges in a smaller number of iterations and a smaller amount of time than BGS with the partitioning $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ $[\mathcal{A}^{(4)}]$ or $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}$ $[\mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$. Observe also that IAD with the partitioning $\mathcal{A}^{(2)} \mid \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ $[\mathcal{A}^{(4)}]$ performs better than BGS with the partitioning $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ $[\mathcal{A}^{(4)}]$ but worse than BGS with the partitioning $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}$ $[\mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$.

The goal of the second set of experiments, which we do not tabulate, is to observe the speed of convergence of IAD with different orders of the lumped matrix. In other words, we vary C_2 while keeping C_1 and C_3 constant. When $C_2 = 10$, the number of iterations it takes to converge for IAD is the same as the corresponding values in Table 6.7. When $C_2 = 5$, the increase in the

Table 6.7: Results of experiments with the three queues problem (unfavorable partitionings)

	$C_i = 25$		$C_i = 30$		$C_i = 35$	
	it #	time	it #	time	it #	time
IAD, $\mathcal{A}^{(2)} \mid \mathcal{A}^{(1)}$ $[\mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$	389	437	447	1,098	497	2,368
BGS, $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}$ $[\mathcal{A}^{(3)}, \mathcal{A}^{(4)}]$	496	654	573	1,508	640	3,323
IAD, $\mathcal{A}^{(2)} \mid \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ $[\mathcal{A}^{(4)}]$	540	693	622	1,741	691	3,872
BGS, $\mathcal{A}^{(2)}, \mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ $[\mathcal{A}^{(4)}]$	744	1,071	862	2,447	965	5,295

number of iterations is less than 3%, and when $C_2 = 2$, the increase in the number of iterations is less than 36%, 48%, 58% for the instances in which $C_1 = C_3 = 25, 30, 35$, respectively.

The second example of a lumpable SAN we use in our experiments is the pushout SAN model described in appendix 8.2.4. There is a single irreducible subset of states in this model. Hence, as with the three queues problem, we first run the SC algorithm to identify transient states. We set the $\lambda_l = 0.2$, $\lambda_h = 0.5$, and $\tilde{\lambda}_h = 0.1$. The integer parameters are given in Table 6.6. We experiment with the quasi-proper ordering $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$, $\mathcal{A}^{(3)}$. There is a single lumpable partitioning in which blocks are formed by the last two automata and the lumped matrix is of order 2.

The results of experiments with the pushout model for different values of buffer size B, load L, and square coefficient of variation C^2 of the two state automaton that models the arrival process of high priority packets are presented in Table 6.8. With BGS we use the partitioning $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$ $[\mathcal{A}^{(3)}]$, and with IAD we use the partitioning $\mathcal{A}^{(1)} \mid \mathcal{A}^{(2)}$ $[\mathcal{A}^{(3)}]$ in which the diagonal blocks of order $(B+1)^2$ are solved using BGS. As in the experiments with the three queues problem, all the blocks of order $(B+1)$ that are input to the BGS solvers are generated and factorized once in the outset and the LU factors are stored in core memory.

When the diagonal blocks of the lumpable partitioning are solved with a stopping tolerance of 10^{-5}, IAD converges in a smaller number of iterations but in a larger amount of time than BGS in almost all experiments. With a stopping tolerance of 10^{-3} for the diagonal blocks, IAD converges in a smaller amount

Table 6.8: Results of experiments with the pushout problem (unfavorable partitionings)

			BGS		IAD, 10^{-5}		IAD, 10^{-3}	
$B+1$	C^2	L	it #	time	it #	time	it #	time
250	1	0.9	693	149	282	139	618	126
250	10	0.1	990	212	397	246	982	188
250	10	0.9	1,050	224	726	272	1,021	201
250	1	0.1	1,140	242	506	263	1,137	216
500	1	0.9	1,225	1,093	429	877	1,204	913
500	10	0.9	1,594	1,413	1,011	1,549	1,594	1,190
500	10	0.1	1,613	1,374	556	1,429	1,607	1,243
500	1	0.1	1,888	1,603	767	1,608	1,886	1,410
750	1	0.9	1,746	3,478	565	2,760	1,739	3,176
750	10	0.9	2,110	4,198	1,255	4,651	2,107	4,002
750	10	0.1	2,230	4,383	692	4,305	2,199	3,841
750	1	0.1	2,600	6,489	1,081	4,817	2,597	4,529

of time and number of iterations than BGS except the problem $(B+1) = 500$, $C^2 = 10$, $L = 0.9$ for which the number of iterations for BGS and IAD are the same.

6.4 Conclusion

In this chapter, we presented results of numerical experiments with various lumpable SANs. The first model we considered is the discrete-time SAN of the wireless ATM system in chapter 3. We indicated that the transition probability matrix of the underlying MC is a dense matrix. Moreover, transition probability matrices of automata of the SAN have large number of functional transitions. All this hints at difficulties in analysis of the SAN using conventional methods. At the same time, using the result of Corollary 5.1, we showed that the MC underlying the SAN model is lumpable with respect to the partitioning in equation (5.2). We analyzed the SAN model with the IAD algorithm for lumpable SANs and demonstrated its performance on selected instances of experiments.

In section 6.2, we concentrated on analysis of lumpable continuous-time SANs. We used two lumpable SAN models to compare performance of the

IAD algorithm with the highly competitive BGS. We observed that in all experiments IAD converged in smaller number of iteration and a smaller amount of time than BGS.

In section 6.3, we studied performance of IAD on SANs having unfavorable lumpable partitionings with relatively large blocks. To overcome difficulties in solving large diagonal blocks at each iteration of IAD, we employed recursive implementation of BGS for SANs discussed in [67]. The experiments with two SAN models having unfavorable partitionings showed that it is possible to tune IAD such that it outperforms BGS.

Chapter 7

Concluding remarks

In this thesis we considered methods for the efficient analysis of large, finite MCs modeled as SANs. First, we implemented a state classification algorithm for SANs that classifies each state in the global state space as essential or transient. The output of the state classification algorithm is used by the NCD partitioning algorithm for SANs and the IAD algorithm for lumpable SANs that we developed.

Second, we extended the NCD concept to SANs. The definitions, propositions, and remarks presented in section 2.2 and chapter 4 enabled us to devise an efficient algorithm that computes NCD partitionings of the MC underlying a SAN. The approach is based on determining the NCD connected components of a SAN from the description of individual automata without generating the global transition matrix. The time and space complexities of the NCD partitioning algorithm depend on the number of automata, the number of synchronizing events, the number of functions, the number of essential states of interest, the sparsity of automata matrices, the dependency sets, and the ordering of automata.

Third, we derived easy to check conditions for the lumpability of the MC underlying a SAN model with functional dependencies. For lumpable SANs, an efficient two-level iterative algorithm that uses exact aggregation with a BGS disaggregation step is introduced. We gave a discrete-time SAN model of a current application in mobile communications and pointed out the difficulties associated with the analysis of discrete-time SANs using conventional techniques. By applying the theorems in chapter 5, we showed that the MC underlying the particular discrete-time SAN model is lumpable and hence can be analyzed with the help of the devised IAD algorithm. Experiments with two lumpable continuous-time SAN models with functional dependencies show that the proposed IAD algorithm 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.

We compared the performance of IAD for lumpable continuous-time SANs with the highly competitive BGS solver on two SAN models having unfavorable lumpable partitionings in which blocks are of relatively large order. To overcome the difficulties associated with solving large diagonal blocks at each iteration of IAD, we used a recursive implementation of BGS. Numerical experiments with the two SAN models show that it is possible to tune the IAD solver so that it performs better than BGS. The time and the number of iterations it takes IAD to converge depends on the accuracy with which the relatively large diagonal blocks are solved and the order of the relatively small lumped matrix. When solving the diagonal blocks of a lumpable partitioning using BGS within the IAD framework, it is important to choose a balanced partitioning of diagonal blocks that results in faster convergence.

Bibliography

[1] M. Ajmone-Marsan, G. Babio, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[2] M. Ajmone-Marsan, G. Babio, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with generalized stochastic Petri nets.* John Wiley & Sons, Chichester, 1995.

[3] F. Basket, K. Chandy, R. Muntz, and F. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

[4] G. Bolch, S. Greiner, H. Meer, and K.S. Trivedi. *Queueing networks and Markov chains.* Interscience. John Wiley & Sons, New York, 1994.

[5] J.W. Brewer. Kronecker products and matrix calculus in systems theory. *IEEE Transactions on Circuits and Systems*, 25:772–781, 1978.

[6] P. Buchholz. An aggregation\disaggregation algorithm for stochastic automata networks. *Probability in the Engineering and Informational Sciences*, 11:229–253, 1997.

[7] P. Buchholz. An adaptive aggregation/disaggregation algorithm for hierarchical Markovian models. *European Journal of Operational Research*, 116:545–564, 1999.

[8] P. Buchholz. Projection methods for the analysis of stochastic automata networks. In B. Plateau, W. J. Stewart, and M. Silva, editors, *Proceedings of the 3rd International Workshop on the Numerical Solution of Markov Chains*, Spain, 1999. Prensas Universitarias de Zaragoza.

[9] P. Buchholz. Multilevel solutions for structured Markov chains. *SIAM Journal on Matrix Analysis and Applications*, 22:342–357, 2000.

[10] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solutions of Markov models. *INFORMS Journal on Computing*, 12(3):203–222, 2000.

[11] P. Buchholz, M. Fischer, and P. Kemper. Distributed steady state analysis using Kronecker algebra. In B. Plateau, W. J. Stewart, and M. Silvia, editors, *Proceedings of the 3rd International Workshop on the Numerical Solution of Markov Chains*, pages 76–95, Spain, 1999. Prensas Universitarias de Zaragoza.

[12] E. Çinlar. *Introduction to stochastic processes.* Prentice-Hall, Englewood Cliffs, 1975.

[13] R. Chan and W. Ching. Circulant preconditioners for stochastic automata networks. Technical Report 98-05 (143), Department of Mathematics, The Chinese University of Hong Kong, April 1998.

[14] G. Ciardo and K. S. Trivedi. Solution of large GSPN models. In W. J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 565–596, New York, 1991. Marcel Dekker Inc.

[15] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms.* The MIT Press, Cambridge, Massachusetts, 1990.

[16] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, C-30:116–125, 1981.

[17] T. Dayar. Permuting Markov chains to nearly completely decomposable form. Technical Report BU-CEIS-9808, Department of Computer Engineering, Bilkent University, Ankara, Turkey, August 1998. Available online at http://www.cs.bilkent.edu.tr/tech-reports/1998/BU-CEIS-9808.ps.z.

[18] T. Dayar, O. I. Pentakalos, and A. B. Stephens. Analytical modeling of robotic tape libraries using stochastic automata. Technical Report TR-97-189, CESDIS, NASA/GSFC, Greenbelt, Maryland, January 1997.

[19] T. Dayar and W. J. Stewart. On the effects of using the Grassmann-Taksar-Heyman method in iterative aggregation-disaggregation. *SIAM Journal on Scientific Computing*, 17:187–303, 1996.

[20] T. Dayar and W. J. Stewart. Comparison of partitioning techniques for two-level iterative solvers on large, sparse Markov chains. *SIAM Journal on Scientific Computing*, 21:1691–1705, 2000.

[21] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

[22] P. Fernandes, R.J. Hessel, B.Plateau, and W.J. Stewart. *PEPS-2000 user manual*, June 2000. Available online at http://www-apache.imag.fr/software/peps/PEPSman2000.ps.gz.

[23] P. Fernandes and B. Plateau. Stochastic automata networks (SAN): modeling and evaluation. In U. Herzog and M. Rettelbach, editors, *Proceedings of the second process algebras and performance modelling workshop*, pages 127–136. Universität Erlangen, 1994.

[24] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplications in stochastic automata networks. *Journal of the ACM*, 45:381–414, 1998.

[25] P. Fernandes, B. Plateau, and W. J. Stewart. Optimizing tensor product computations in stochastic automata networks. *RAIRO, Operations Research*, 32(3):325–351, 1998.

[26] J.-M. Fourneau. Stochastic automata networks: using structural properties to reduce the state space. In B. Plateau, W. J. Stewart, and M. Silva, editors, *Proceedings of the 3rd International Workshop on the Numerical Solution of Markov Chains*, pages 332–334, Spain, 1999. Prensas Universitarias de Zaragoza.

[27] J.-M. Fourneau, H. Maisonniaux, N. Pekergin, and V. Véque. Performance evaluation of a buffer policy with stochastic automata networks. In *IFIP Workshop on Modelling and Performance Evaluation of ATM Technology*,

volume C-15 of *IFIP Transactions North-Holland*, pages 433–451, Amsterdam, 1993. La Martinique.

[28] J.-M. Fourneau and F. Quessette. Graphs and stochastic automata networks. In W. J. Stewart, editor, *Computations with Markov Chains: Proceedings of the 2nd International Workshop on the Numerical Solution of Markov Chains*, pages 217–235, Boston, Massachusetts, 1995. Kluwer.

[29] E. Gelenbe and G. Pujolle. *Introduction to queueing networks.* Wiley, New York, 1987.

[30] D. Ginsburg. *ATM: solutions for enterprise internetworking.* Addison-Wesley, Great Britain, second edition, 1999.

[31] W. Gordon and G. Newell. Closed queuing systems with exponential servers. *Operations Research*, 15(2):254–265, 1967.

[32] O. Gusak and T. Dayar. Solving lumpable continuous-time stochastic automata networks with unfavorable partitionings. In M. Salamah, I. Aybay, and A. Acan, editors, *Proceedings of the Sixth Symposium on Computer Networks*, pages 31–40, TRNC, 2001. Eastern Mediterranean University.

[33] O. Gusak, T. Dayar, and J.-M. Fourneau. Stochastic automata networks and near complete decomposability. To appear in *SIAM Journal on Matrix Analysis and Applications*.

[34] O. Gusak, T. Dayar, and 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.

[35] B.R. Haverkort. *Performance of computer communication systems: a model-based approach.* Wiley, New York, 1998.

[36] R. Haverkort and K. Trivedi. Specification and generation of Markov reward models. *Discrete-Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.

[37] J. Jackson. Jobshop-like queuing systems. *Management Science*, 10(1):131–142, 1963.

[38] J.R. Kemeny and J.L. Snell. *Finite Markov chains.* Van Nostrand, New York, 1960.

[39] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Transactions on Computers*, C-22(9):615–628, 1996.

[40] L. Kleinrock. *Queueing systems*, volume 1: Theory. Wiley, New York, 1975.

[41] I. Marek and P. Mayer. Convergence analysis of an iterative aggregation-disaggregation method for computing stationary probability vectors of stochastic matrices. *Numerical Linear Algebra with Applications*, 5:253–274, 1998.

[42] I. Marek and D.B. Szyld. Local convergence of the (exact and inexact) iterative aggregation method for linear systems and Markov operators. *Numerische Mathematik*, 69:61–82, 1994.

[43] C. D. Meyer. Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems. *SIAM Review*, 31:240–272, 1989.

[44] M.K. Molloy. *Fundamentals of performance modeling.* Macmillan, New York, 1989.

[45] M.F. Neuts. *Matrix geometric solutions in stochastic models: an algorithmic approach.* John Hopkins University Press, 1981.

[46] M.F. Neuts. *Structured stochastic matrices of M/G/1 type and their applications.* Marcel Dekker, Inc., 1989.

[47] N. Pekergin. Stochastic performance bounds by state space reduction. *Performance Evaluation*, 36-37:1–17, 1999.

[48] J.L. Peterson. *Petri net theory and the modelling of systems.* Prentice-Hall, Englewood Cliffs, 1981.

[49] C.A. Petri. *Kommunikation mit automaten.* Dissertation, University of Bonn, Germany, 1962.

[50] B. Plateau. *De l'évaluation du parallélisme et de la synchronisation.* Thèse d'état, Université Paris Sud Orsay, France, 1984.

[51] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 147–154, Texas, 1985.

[52] B. Plateau and K. Atif. Stochastic automata network for modeling parallel systems. *IEEE Transactions on Software Engineering*, 17:1093–1108, 1991.

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

[54] B. Plateau, J.-M. Fourneau, and K.-H. Lee. PEPS: a package for solving complex Markov models of parallel systems. In R. Puigjaner and D. Ptie, editors, *Modeling Techniques and Tools for Computer Performance Evaluation*, pages 291–305, Spain, 1988.

[55] B. Plateau and W.J. Stewart. Stochastic automata networks: product forms and iterative solutions. Research Report 2939, INRIA, France, 1996. Available online at ftp://ftp.inria.fr/INRIA/Publication/RR/RR-2939.ps.gz.

[56] M. Reisig. *Petri nets: an introduction.* Springer-Verlag, Berlin, 1985.

[57] S.M. Ross. *Stochastic processes.* Wiley, New York, 1983.

[58] Y. Saad. *Iterative methods for sparse linear systems.* PWS Publishing Company, Boston, 1995.

[59] P. Schweitzer. A survey of aggregation-disaggregation in large Markov chains. In W. J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 63–88, New York, 1991. Marcel Dekker Inc.

[60] M. Shaked and G. Shantikumar. *Stochastic orders and their applications.* Academic Press, Orlando, 1994.

[61] W. Stewart, W. J. Stewart, and D. F. McAllister. A two-stage iteration for solving nearly completely decomposable Markov chains. In G. H. Golub, A. Greenbaum, and M. Luskin, editors, *Recent Advances in Iterative Methods, IMA Vol. Math. Appl. 60*, pages 201–216, New York, 1994. Springer-Verlag.

[62] W. J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, Princeton, New Jersey, 1994.

[63] W. J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. *European Journal of Operational Research*, 86:503–525, 1995.

[64] W. J. Stewart and W. Wu. Numerical experiments with iteration and aggregation for Markov chains. *ORSA Journal on Computing*, 4:336–350, 1992.

[65] D. Stoyan. *Comparison methods for queues and other stochastic models.* Wiley, New York, 1976.

[66] B. Stroustrup. *The C++ programming language.* Reading. Addison-Wesley, Massachusetts, 1991.

[67] E. Uysal and T. Dayar. Iterative methods based on splittings for stochastic automata networks. *European Journal of Operational Research*, 110:166–186, 1998.

[68] V. Vèque and J. Ben-Othman. MRAP: a multiservices resource allocation policy for wireless ATM network. *Computer Networks and ISDN Systems*, 29:2187–2200, 1998.

Chapter 8

Appendix

8.1 Tensor algebra

In this section, we give definitions of tensor product and tensor sum and state their basic properties. In the following, we use $A_{m\times n}$ for a matrix of dimension $m \times n$ and I_m for the identity matrix of size m.

Definition 8.1 *Let $A_{m\times n}$ and $B_{k\times l}$ be two matrices as*

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \; B = \begin{pmatrix} b_{11} & \cdots & b_{1l} \\ \vdots & \ddots & \vdots \\ b_{k1} & \cdots & b_{kl} \end{pmatrix}.$$

Then, the tensor product of A and B, $C_{mk\times nl} = A \otimes B$ is given by

$$C = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}.$$

Definition 8.2 *The tensor sum of two square matrices $A_{n\times n}$ and $B_{m\times m}$, $C_{nm\times nm} = A \oplus B$ is defined as*

$$C = A \otimes I_m + I_n \otimes B.$$

Below we list properties of the tensor operators as they appear in [24]. Note that the matrices A, B, C, and D are square.

- Associativity:

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \text{ and } A \oplus (B \oplus C) = (A \oplus B) \oplus C.$$

- Distributivity over ordinary matrix addition:

$$(A + B) \otimes (C + D) = A \otimes C + B \otimes C + A \otimes D + B \otimes D.$$

- Compatibility with ordinary matrix multiplication (case I):

$$(A \cdot B) \otimes (C \cdot D) = (A \otimes C) \cdot (B \otimes D).$$

- Compatibility with ordinary matrix multiplication (case II):

$$\bigotimes_{i=1}^{N} A^{(i)} = \prod_{i=1}^{N} I_{1:i-1} \otimes A^{(i)} \otimes I_{i+1:N},$$

  where $A^{(i)}$ is a square matrix of order n_i, $i = 1, 2, \ldots, N$, $I_{i:j}$ is the identity matrix of order $\prod_{k=i}^{j} n_k$.

- Compatibility with ordinary matrix inversion:

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}.$$

- Pseudo commutativity:

$$\bigotimes_{i=1}^{N} A^{(i)} = P_\sigma \bigotimes_{i=1}^{N} A^{\sigma(i)} P_\sigma^T,$$

  where σ is a permutation of set of integers $\{1, 2, \ldots, N\}$ and P_σ is a permutation square matrix of order $\prod_{k=1}^{N} n_k$.

Further information related to properties of tensor algebra can be found in [5, 16] and information related to definitions and properties of generalized tensor algebra can be found in [24].

8.2 Examples of continuous-time SANs

This section provides the description of SAN models used as test problems in the numerical experiments of chapters 4 and 6. The original SAN models of appendices 8.2.1 and 8.2.2 appear in [24], the SAN model of appendix 8.2.3 is introduced in [18], and the SAN model of appendix 8.2.4 appears in [22].

8.2.1 SAN model of the resource sharing problem

This SAN models a system that consists of U processes of which at most S can be simultaneously accessing the resource. Each process alternates between sleeping and resource using states. The transition rates between these two states for process i are characterized respectively by λ_i and μ_i, $i = 1, 2, \ldots, U$. We consider the case $S < U$; otherwise, all processes are independent from each other and the problem can be modeled as a trivial multidimensional MC. In the SAN model [24], each process is represented by a two-state automaton in which state 0 corresponds to the sleeping state and state 1 to the resource using state. There are U such automata implying a state space size of $n = 2^U$. This SAN does not have synchronizing events, i.e., $E = 0$. There is a single subset of essential states whose size depends on S with respect to U. Since $S < U$, process i cannot acquire the resource if it is already being used by S processes. Hence, λ_i is a functional transition rate, and the value of the function (i.e., whether the transition is enabled or disabled) depends on the state of all other automata. The local transition rate matrix of automaton i is given by

$$Q_l^{(i)} = \begin{pmatrix} -\lambda_i f & \lambda_i f \\ \mu_i & -\mu_i \end{pmatrix},$$

where

$$f = \delta\left(\sum_{i=1}^{U} \delta(s\mathcal{A}^{(i)} = 1) < S\right),$$

and δ is the Kronecker delta.

8.2.2 SAN model of the three queues problem

The three queues problem is an open queueing network that consists of three queues with capacities (C_1-1), (C_2-1), and (C_3-1). Customers from queues 1 and 2, called respectively as type 1 and type 2 customers, try to join queue 3. When type 1 customers try to join queue 3 and find it full, they are blocked, whereas type 2 customers in the same situation are lost. Queue i has arrival rate λ_i and service rate μ_i, $i=1,2$. The service rate of queue 3 depends on the type of customer and is either μ_{3_1} or μ_{3_2}; moreover, type 1 customers have priority over type 2 customers in service. The system is modeled by 2 synchronizing events and 4 automata $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3_1)}, \mathcal{A}^{(3_2)}$ with respectively C_1, C_2, C_3, C_3 states. The states of the automata are numbered starting from 0 and correspond to the number of customers in a particular queue. The first synchronizing event corresponds to the departure of a type 1 customer from queue 1 and its arrival to queue 3. The second synchronizing event models the departure of a type 2 customer from queue 2 and its arrival to queue 3 (if there is an empty space in it). The state space size is given by $n=C_1C_2C_3^2$ and there is a single subset of $C_1C_2C_3(C_3+1)/2$ essential states. Functional transition rates appear in local transition rate and synchronizing event matrices.

$\mathcal{A}^{(1)}$ models the number of customers in queue 1 and we have

$$Q_l^{(1)} = \begin{pmatrix} -\lambda_1 & \lambda_1 & 0 & \cdots & 0 \\ 0 & -\lambda_1 & \lambda_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\lambda_1 & \lambda_1 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \quad Q_{e_1}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_1 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \mu_1 & 0 & 0 \\ 0 & \cdots & 0 & \mu_1 & 0 \end{pmatrix},$$

$$\bar{Q}_{e_1}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -\mu_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mu_1 & 0 \\ 0 & \cdots & 0 & 0 & \mu_1 \end{pmatrix}, \quad Q_{e_2}^{(1)} = I_{C_1} = \bar{Q}_{e_2}^{(1)}.$$

For $\mathcal{A}^{(2)}$ that models the number of customers in queue 2, we have

$$Q_l^{(2)} = \begin{pmatrix} -\lambda_2 & \lambda_2 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\lambda_2 & \lambda_2 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \ Q_{e_2}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_2 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \mu_2 & 0 & 0 \\ 0 & \cdots & 0 & \mu_2 & 0 \end{pmatrix},$$

$$\bar{Q}_{e_2}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -\mu_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mu_2 & 0 \\ 0 & \cdots & 0 & 0 & \mu_2 \end{pmatrix}, \ Q_{e_1}^{(2)} = I_{C_2} = \bar{Q}_{e_1}^{(2)}.$$

$\mathcal{A}^{(3_1)}$ models the number of type 1 customers in queue 3 and its transition matrices are given by

$$Q_l^{(3_1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_{3_1} & -\mu_{3_1} & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \mu_{3_1} & -\mu_{3_1} & 0 \\ 0 & \cdots & 0 & \mu_{3_1} & -\mu_{3_1} \end{pmatrix}, \ Q_{e_1}^{(3_1)} = \begin{pmatrix} 0 & f & 0 & \cdots & 0 \\ 0 & 0 & f & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 & f \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix},$$

$$\bar{Q}_{e_1}^{(3_1)} = \begin{pmatrix} f & 0 & 0 & \cdots & 0 \\ 0 & f & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & f & 0 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \ Q_{e_2}^{(3_1)} = I_{C_3} = \bar{Q}_{e_2}^{(3_1)},$$

where

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

Finally, $\mathcal{A}^{(3_2)}$ models the number of type 2 customers in queue 3 and we have

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

$$Q_{e_2}^{(3_2)} = \begin{pmatrix} 1-f & f & 0 & \cdots & 0 \\ 0 & 1-f & f & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1-f & f \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix}, \; \bar{Q}_{e_2}^{(3_2)} = I_{C_3} = Q_{e_1}^{(3_2)} = \bar{Q}_{e_1}^{(3_2)},$$

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

8.2.3 SAN model of the mass storage problem

The SAN model of the mass storage system consists of two layers of storage. The first layer provides fast access based on magnetic disks. The second layer (i.e., nearline storage) utilizes a robotic tape library. The system which describes the interaction between read/write requests and these two layers is modeled by a SAN of 3 synchronizing events and 5 automata with functional transitions. The automata are denoted by $\mathcal{A}^{(\tilde{C})}, \mathcal{A}^{(\eta_1)}, \mathcal{A}^{(\eta_2)}, \mathcal{A}^{(\eta_3)}\mathcal{A}^{(erl)}$, and they respectively have $n_C = \lceil (H-L)(C-1) \rceil + 1$, n_1, n_2, n_3, R states, where $H(L)$ is the high (low) water-mark for the disk cache. The state space size is $n = n_C n_1 n_2 n_3 R$, and the corresponding MC is irreducible. We use the same notation for the parameters of the SAN model as in [18] where detailed description of the underlying physical model, its parameters, and the design decisions can be found.

For $\mathcal{A}^{(\eta_1)}$, we have

$$Q_l^{(\eta_1)} = \begin{pmatrix} -\lambda_g(1-h_g) & \lambda_g(1-h_g) & 0 & \cdots & 0 \\ 0 & -\lambda_g(1-h_g) & \lambda_g(1-h_g) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\lambda_g(1-h_g) & \lambda_g(1-h_g) \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix},$$

$$Q_{e_3}^{(\eta_1)} = \begin{pmatrix} 0 & 0 & \cdots & \cdots & 0 \\ \min\{1,t_i\}\mu & 0 & \ddots & \ddots & \vdots \\ 0 & \min\{2,t_i\}\mu & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \min\{n_1-1,t_i\}\mu & 0 \end{pmatrix},$$

$$\bar{Q}_{e_3}^{(\eta_1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -\min\{1,t_i\}\mu & 0 & \ddots & \vdots \\ 0 & 0 & -\min\{2,t_i\}\mu & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & -\min\{n_1-1,t_i\}\mu \end{pmatrix},$$

and $Q_{e_1}^{(\eta_1)} = \bar{Q}_{e_1}^{(\eta_1)} = Q_{e_2}^{(\eta_1)} = \bar{Q}_{e_2}^{(\eta_1)} = I_{n_1}$.

For $\mathcal{A}^{(\eta_2)}$ we have:

$$Q_l^{(\eta_2)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ g_1\mu & -g_1\mu & 0 & \ddots & \vdots \\ 0 & g_2\mu & -g_2\mu & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & g_{n_2-1}\mu & -g_{n_2-1}\mu \end{pmatrix},$$

where $g_i = \min\{i, t_m\}$, and t_m is a function of $s\mathcal{A}^{(\eta_1)}$. We also have

$$Q_{e_1}^{(\eta_2)} = \begin{pmatrix} f_0 & f_1 & \cdots & f_{n_3-1} & 0 & \cdots & 0 \\ 0 & f_0 & f_1 & \cdots & f_{n_3-1} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & f_0 & \ddots & \ddots & f_{n_3-1} \\ \vdots & \ddots & \ddots & 0 & f_0 & \ddots & f_{n_3-2} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & f_0 \end{pmatrix},$$

where $f_0, f_1, \ldots, f_{n_3-1}$ are functions whose values depend on $s\mathcal{A}^{(\eta_2)}$ and $s\mathcal{A}^{(\eta_3)}$. The rest of the synchronizing transition probability matrices of $\mathcal{A}^{(\eta_2)}$ are identity of order n_2.

The local transition rate matrix $Q_l^{(\eta_3)}$ is a zero matrix, and the synchronizing transition matrices of $\mathcal{A}^{(\eta_3)}$ are given by

$$Q_{e_1}^{(\eta_3)} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}, \; \bar{Q}_{e_1}^{(\eta_3)} = Q_3^{(\eta_3)} = \bar{Q}_3^{(\eta_3)} = I_{n_3},$$

$$Q_{e_2}^{(\eta_3)} = \begin{pmatrix} 0 & \lambda & 0 & \cdots & 0 \\ 0 & 0 & \lambda & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 & \lambda \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix}, \; \bar{Q}_{e_2}^{(\eta_3)} = \begin{pmatrix} -\lambda & 0 & 0 & \cdots & 0 \\ 0 & -\lambda & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -\lambda & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix}.$$

For $\mathcal{A}^{(erl)}$, we have:

$$Q_l^{(erl)} = \begin{pmatrix} -R\gamma & R\gamma & 0 & \cdots & 0 \\ 0 & -R\gamma & R\gamma & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -R\gamma & R\gamma \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix},$$

$$Q_{e_1}^{(erl)} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ R\gamma & 0 & \cdots & 0 \end{pmatrix}, \; \bar{Q}_{e_1}^{(erl)} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & -R\gamma \end{pmatrix},$$

and the rest of the synchronizing transition probability matrices are identity of order R.

For $\mathcal{A}^{(\tilde{C})}$, the local transition rate matrix $Q_l^{(\tilde{C})}$ is a zero matrix,

$$Q_{e_2}^{(\tilde{C})} = \begin{pmatrix} h_p & \bar{h}_p & 0 & \cdots & 0 \\ 0 & h_p & \bar{h}_p & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & h_p & \bar{h}_p \\ \bar{h}_p & 0 & \cdots & 0 & h_p \end{pmatrix}, \; Q_{e_3}^{(\tilde{C})} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix},$$

where $\bar{h}_p = 1 - h_p$. The rest of the synchronizing transition probability matrices of $\mathcal{A}^{(\tilde{C})}$ are identity of order n_C.

8.2.4 SAN model of the pushout problem

This SAN example models a priority queue that receives high and low priority packets. Low priority packets can receive service only if there are no high priority packets in the buffer. The buffer has a limited capacity. Hence, an arrival is lost when the buffer is full. To avoid the loss of high priority packets, signaling cells are used that precede the arrival of a burst of high priority packets. Upon the arrival of a signaling cell, all low priority packets that are present in the buffer are lost.

The SAN model of the pushout probelm consists of 3 automata numbered consecutively from 1 to 3. The first automaton having two states, 0 and 1, models the arrival process of high and low priority packets. The last two automata model a buffer of size B that accommodates high and low priority packets served by a single server with rate μ. Upon transition of automaton 1 from state 0 to state 1, all low priority packets are removed from the buffer. This behavior is modeled by the only synchronizing event. States of $\mathcal{A}^{(2)}$ and $\mathcal{A}^{(3)}$ are numbered starting from 0 and correspond to the number of high and low priority packets in the buffer, respectively. The SAN model contains functions in the matrices of the last two automata which cyclically depend on each other. The values of the functions in the matrices of the second automaton, which models the number of high priority packets in the buffer, also depend on the state of the first automaton.

For $\mathcal{A}^{(1)}$, we have

$$Q_l^{(1)} = \begin{pmatrix} 0 & 0 \\ \beta & -\beta \end{pmatrix}, \; Q_{e_1}^{(1)} = \begin{pmatrix} 0 & \alpha \\ 0 & 0 \end{pmatrix}, \; \bar{Q}_{e_1}^{(1)} = \begin{pmatrix} -\alpha & 0 \\ 0 & 0 \end{pmatrix},$$

where α and β are transition rates from state 0 to 1 and from state 1 to 0, respectively.

For $\mathcal{A}^{(2)}$, we have

$$Q_l^{(2)} = \begin{pmatrix} -f_h & f_h & 0 & \cdots & 0 \\ \mu & -(\mu+f_h) & f_h & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \mu & -(\mu+f_h) & f_h \\ 0 & \cdots & 0 & \mu & -\mu \end{pmatrix}, \; Q_{e_1}^{(2)} = \bar{Q}_{e_1}^{(2)} = I_{(B+1)},$$

where $f_h = \left(\delta(s\mathcal{A}^{(1)} = 1)\lambda_h + \delta(s\mathcal{A}^{(1)} = 0)\tilde{\lambda}_h\right)\delta(s\mathcal{A}^{(2)} + s\mathcal{A}^{(3)} \leq B)$, λ_h and $\tilde{\lambda}_h$ are the arrival rates of high priority packets when the data source is in state 1 and state 0, respectively.

For $\mathcal{A}^{(3)}$, we have

$$Q_l^{(3)} = \begin{pmatrix} -f_l & f_l & 0 & \cdots & 0 \\ \mu & -(\mu+f_l) & f_l & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \mu & -(\mu+f_l) & f_l \\ 0 & \cdots & 0 & \mu & -\mu \end{pmatrix}, \; Q_{e_1}^{(3)} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix},$$

and $\bar{Q}_{e_1}^{(3)} = I_{(B+1)}$, where $f_l = \lambda_l \delta(s\mathcal{A}^{(2)} + s\mathcal{A}^{(3)} \leq B)$ and λ_l is the arrival rate of low priority packets.

8.3 State classification algorithm for SANs

The SC algorithm is based on an algorithm that finds SCCs of a digraph using depth first search (DFS) algorithm. When the digraph corresponds to the MC underlying a SAN, for each global state s of the SAN, we have to find all states $\tilde{s}(\neq s)$ such that $Q(s, \tilde{s}) \neq 0$. Recall that the global state s can be represented as a vector $(s_1, s_2, \ldots, s_N)$, where $s_k = s\mathcal{A}(k)$ (see section 2.1). According to Remarks 4.1 and 4.2, a nonzero element in Q can originate either from a local transition or from one or more synchronizing transitions. If $Q(s, \tilde{s})$ originates from the local transition $Q_l^{(k)}(s_k, \tilde{s}_k)$, then it must be that $\tilde{s}$ corresponds to the global state $(s_1, \ldots, s_{k-1}, \tilde{s}_k, s_{k+1}, \ldots, s_N)$. If $Q(s, \tilde{s})$ is formed of transitions of synchronizing event j, then $Q_{e_j}^{(k)}(s_k, \tilde{s}_k) \neq 0$ for $k = 1, 2, \ldots, N$ (see Remark 4.3). The implementation of a DFS algorithm for SANs is straightforward once these observations are made. The output of the SC algorithm is an integer array of length n with states corresponding to the essential subset of interest marked. Assuming that the matrices of the SAN description are stored in sparse format, the number of times global states are visited by the SC algorithm is $\sum_{i=1}^{N} nz_l^{(i)} \prod_{k=1,k\neq i}^{N} n_k + \sum_{j=1}^{E} \prod_{i=1}^{N} nz_{e_j}^{(i)}$, where $nz_l^{(i)}$ is the number of off-diagonal nonzero elements in $Q_l^{(i)}$ and $nz_{e_j}^{(i)}$ is the number of nonzero elements in $Q_{e_j}^{(i)}$.

8.4 Algorithm 1. NCD partitioning algorithm for SANs

This section contains the description of the three step NCD partitioning algorithm introduced in chapter 4. The description of the algorithm also appears in [34].

8.4.1 Algorithm 1.1. $Q \rightarrow P$ transformation

Computing α.

Input:

- $Q_l^{(i)}$, $\bar{Q}_{e_j}^{(i)}$, $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, E$
- $\mathcal{D}_k$ is kth dependency set, $k = 1, 2, \ldots, N_{\mathcal{D}}$, as in Definition 4.1
- integer array of length n with essential subset of interest marked by SC algorithm

Output:

- α as discussed in subsection 4.2.1

Notation:

- S_k: state that corresponds to max_$\mathcal{D}_k$
- $\mathcal{M}_{\mathcal{D}_k}$: set of synchronizing events whose masters are in $\mathcal{D}_k$
- s: global state that corresponds to $S_1, S_2, \ldots, S_{N_\mathcal{D}}$
- $Diag_k$: double precision array of length $\prod_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} n_i$

for $k = 1, 2, \ldots, N_D$,
    $Diag_k = \left|\bigoplus_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} \mathrm{diag}(Q_l^{(i)}) + \sum_{j,e_j\in\mathcal{M}_{\mathcal{D}_k}} \bigotimes_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} \mathrm{diag}(\bar{Q}_{e_j}^{(i)})\right|$
    sort elements of $Diag_k$ in non-increasing order
    find max_$\mathcal{D}_k = \max_i Diag_k(i)$, and corresponding state S_k
    set element corresponding to max_$\mathcal{D}_k$ to 0 in $Diag_k$
while global state s does not map into essential subset of interest,
    for $k = 1, 2, \ldots, N_\mathcal{D}$, next_max_$\mathcal{D}_k = \max_i Diag_k(i)$
    find t such that next_max_$\mathcal{D}_t \geq$ next_max_$\mathcal{D}_k$ for $k = 1, 2, \ldots, N_D$
    let $\tilde{S}_t$ be state corresponding to next_max_$\mathcal{D}_t$
    max_$\mathcal{D}_t$ =next_max_$\mathcal{D}_t$
    set element corresponding to max_$\mathcal{D}_t$ to 0 in $Diag_t$, and $S_t = \tilde{S}_t$
$\alpha = \sum_{i=1}^{N_\mathcal{D}}$ max_$\mathcal{D}_k$

8.4.2 Algorithm 1.2. Finding potential sets

Finding potential sets.

Input:

- $Q_{e_j}^{(i)}$, $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, E$
- ϵ

- α computed by Algorithm 1.1

Output:

- $\mathcal{P}_r$ is rth potential set, $r = 1, 2, \ldots, N_{\mathcal{P}}$, as discussed in subsection 4.2.2

Notation:

- $\|\mathcal{P}_r\|$: sum of transition rates of synchronizing events in $\mathcal{P}_r$[1]
- $Q^{(k)}_{e_i} \equiv Q^{(k)}_{e_j}$ holds if $Q^{(k)}_{e_i}$ and $Q^{(k)}_{e_j}$ have at least one nonzero element with same indices (see second if-statement in innermost for-loop)

Remark:

- $r = N_{\mathcal{P}}$ upon termination

for $i = 1, 2, \ldots, E$, set synchronizing event e_i to unmarked
$r = 0$
for $i = 1, 2, \ldots, E$,
    if e_i is unmarked then
        $r = r + 1$
        $\mathcal{P}_r = \{e_i\}$
        if $\|\mathcal{P}_r\| < \alpha\epsilon$ then
            for $j = i + 1, i + 2, \ldots, E$,
                if e_j is unmarked and $\|\{e_j\}\| < \alpha\epsilon$ then
                    if $Q^{(k)}_{e_i} \equiv Q^{(k)}_{e_j}$ for $k = 1, 2, \ldots, N$ then $\mathcal{P}_r = \mathcal{P}_r \cup \{e_j\}$
            if $\|\mathcal{P}_r\| < \alpha\epsilon$ then
                $\mathcal{P}_r = \emptyset$
                $r = r - 1$
            else mark each $e_p \in \mathcal{P}_r$
        else mark e_i

[1] Note that $\mathcal{P}_r$ is a set and $\|.\|$ should not be mixed up with the norm operator

8.4.3 Algorithm 1.3. Constructing NCD connected components

Constructing NCD connected components.

Input:

- $Q_l^{(i)}$, $Q_{e_j}^{(i)}$, $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, E$
  with $\mathcal{A}^{(i)}$ reordered/renumbered as discussed in subsection 4.2.3
- ϵ
- α computed by Algorithm 1.1
- $\mathcal{P}_r$ is rth potential set, $r = 1, 2, \ldots, N_{\mathcal{P}}$, computed by Algorithm 1.2

Output:

- $\mathcal{C}$: NCD CCs of MC underlying SAN as discussed in subsection 4.2.3

Notation:

- $\mathcal{C}^{(i)}$: set of NCD CCs of $Q^{(i)}$
- σ_r: smallest index among automata involved in synchronizing events in $\mathcal{P}_r$
- $\odot$: binary operator as defined in subsection 4.2.3

. for $i = 1, 2, \ldots, N$,
.   using $\alpha\epsilon$, find $\mathcal{C}^{(i)}$ by treating all functional transition rates as 0
. let K be the number of $\mathcal{C}^{(i)}$s that are singletons
. reorder/renumber $\mathcal{A}^{(i)}$ such that $\mathcal{C}^{(i)}$, $i = N-K+1, N-K, \ldots, N$, are singletons
. if $K > 0$ then $\mathcal{C} = \{\{1, 2, \ldots, n_{N-K+1}\} \times \{1, 2, \ldots, n_{N-K}\} \times \ldots \times \{1, 2, \ldots, n_N\}\}$
. else
. $\mathcal{C} = \mathcal{C}^{(N)}$
. $K = 1$
. for $k = N-K, N-K-1, \ldots, 1$,
. $\mathcal{C} = \mathcal{C}^{(k)} \odot \mathcal{C}$

. for each functional rate $f = Q_l^{(k)}(s_k, \tilde{s}_k)$
. for each pair $\mathsf{c}, \tilde{\mathsf{c}} \in \mathcal{C}$ such that
. $(s_k, s_{k+1}, \ldots, s_N) \in \mathsf{c}$ and $(\tilde{s}_k, s_{k+1}, \ldots, s_N) \in \tilde{\mathsf{c}}$ and
$f(s_k, s_{k+1}, \ldots, s_N) \geq \alpha\epsilon$
. join c with $\tilde{\mathsf{c}}$
. if there exists r such that $\sigma_r = k$ then
. for each pair $\mathsf{c}, \tilde{\mathsf{c}} \in \mathcal{C}$ such that
. $(s_k, s_{k+1}, \ldots, s_N) \in \mathsf{c}$ and $(\tilde{s}_k, \tilde{s}_{k+1}, \ldots, \tilde{s}_N) \in \tilde{\mathsf{c}}$ and
$\sum_{p \in \mathcal{P}_r} \prod_{i=k}^{N} Q_{e_p}^{(i)}(s_i, \tilde{s}_i) \geq \alpha\epsilon$
. join c and $\tilde{\mathsf{c}}$

8.5 Complexity analysis of Algorithm 1

In this section we present the detailed complexity analysis of each step of the Algorithm 1. The detailed complexity analysis also appears in [34].

The time complexity analysis of Algorithm 1 is complicated in that it requires one to make assumptions regarding the number of nonzero elements in automata matrices, the dependency sets, the ordering of automata, the number of functions, and the number of states in the essential subset of interest. Nevertheless, in the following three subsections, we provide some results concerning time and space complexity. In order to simplify the analysis of the algorithm, we assume that the MC underlying the given SAN description is irreducible. We remark that the algorithm and automata matrices are coded in sparse format. Most diagonal corrector matrices in applications turn out to be identity. To reduce storage requirements further and to improve the complexity of operations with identity matrices, we do not store identity matrices.

8.5.1 Algorithm 1.1

According to our simplifying assumption, the MC underlying the given SAN description is irreducible. For an irreducible SAN, the diagonal element with maximum magnitude is given by equation (4.6), which is the sum of the maximums of dependency sets. The operation of finding the maximum of a dependency set requires $\prod_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} n_i$ floating-point comparisons. Since this operation is performed N_D times, the total number of floating-point comparisons is given by $\sum_{k=1}^{N_D}\prod_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} n_i$. For the best case in which each dependency set is a singleton, the total number of floating-point comparisons reduces to $\sum_{i=1}^{N} n_i$. On the other hand, if all automata form a single dependency set, we have the upper bound $\prod_{i=1}^{N} n_i = n$.

When the MC underlying the SAN has uninteresting states, the vectors $\left|\bigoplus_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} \mathrm{diag}(Q_l^{(i)}) + \sum_{j,e_j\in\mathcal{M}_{\mathcal{D}_k}} \bigotimes_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} \mathrm{diag}(\bar{Q}_{e_j}^{(i)})\right|$, $k = 1,2,\ldots,N_{\mathcal{D}}$, are stored as sorted. For a dependency set $\mathcal{D}_k$, sorting of an array of length $n_{\mathcal{D}_k} = \prod_{i,\mathcal{A}^{(i)}\in\mathcal{D}_k} n_i$ requires $O(n_{\mathcal{D}_k}\log n_{\mathcal{D}_k})$ floating-point comparisons. Thus, the total number of floating-point comparisons performed before the while-loop in Algorithm 1.1 is $\sum_{k=1}^{N_{\mathcal{D}}} O(n_{\mathcal{D}_k}\log n_{\mathcal{D}_k})$. The number of iterations of the while-loop depends on the number of essential states of interest and their global state indices. If the number of essential states is n_{ess}, the number of iterations performed of the while-loop is less than or equal to $(n - n_{ess})$. The memory requirement of Algorithm 1.1 consists of a double precision array of length $\sum_{k=1}^{N_{\mathcal{D}}} n_{\mathcal{D}_k}$ $(\leq n)$.

8.5.2 Algorithm 1.2

Each potential set $\mathcal{P}_r$ is described by its member synchronizing events, the sum of scaled transition rates $||\mathcal{P}_r||$, and σ_r, which is the smallest index among automata involved in the synchronized events in $\mathcal{P}_r$. Since we can have at most E potential sets, Algorithm 1.2 requires at most two integer arrays of length E and one double precision array of length E.

Algorithm 1.2 does not involve any floating-point arithmetic. Therefore, its time complexity depends on the relation between $\alpha\epsilon$ and the transition rate of each simple synchronizing event in the SAN description. See the comparison between $||\mathcal{P}_r||$ and $\alpha\epsilon$ in the second if-statement of Algorithm 1.2. Hence, we give lower and upper bounds on the total number of floating-point comparisons.

The lower bound corresponds to the case where the transition rate of each simple synchronizing event is greater than or equal to $\alpha\epsilon$. In this case, the SAN under consideration has E potential sets and each rate is compared against $\alpha\epsilon$ only once since the algorithm never enters the innermost for-loop. Thus, the smallest total number of floating-point comparisons is E.

The upper bound is achieved when the transition rate of each simple synchronizing event is less than $\alpha\epsilon$ and the transition rates of synchronizing events do not sum up in Q. See the innermost if-statement in Algorithm 1.2. In this case, the outermost for-loop is executed E times, and for each value of i, the innermost for-loop is executed $(E - i)$ times. Hence, the total number of floating-point comparisons with $\alpha\epsilon$ is $\frac{1}{2}E(E+1)$.

8.5.3 Algorithm 1.3

It is clear that Algorithm 1.3 requires the largest amount of storage since it works with the global state space of the SAN. The computed NCD partitioning is kept in the structure $\mathcal{C}$ which requires an integer array of length $O(n)$. The exact amount of storage for $\mathcal{C}$ depends on the particular implementation which must be sophisticated enough so that operations of the form "join c and $\tilde{\mathsf{c}}$" can be executed rapidly. This data structure must also grant fast access to a specific element in a particular NCD CC. On the other hand, the data structure $\mathcal{C}^{(i)}$ requires an integer array of length at most $(2n_i + 1)$. The first n_i elements of this array contain a permutation of the state indices of $\mathcal{A}^{(i)}$, its $(n_i + 1)$st element contains the number of NCD CCs, and its last n_i elements contain the number of states in each NCD CC since we may have n_i NCD CCs. Thus for N automata, we need a maximum of $N + 2\sum_{i=1}^{N} n_i$ integer locations for $\mathcal{C}^{(i)}$. The total amount of storage is then upper bounded by $O(n)$ integer locations.

For the time complexity of Algorithm 1.3, we consider floating-point comparisons and also count floating-point multiplications performed to compute transition rates. Floating-point comparisons take place when $\mathcal{C}^{(i)}$ are determined. For $\mathcal{A}^{(i)}$, this means $nz_l^{(i)}$ comparisons are performed, where $nz_l^{(i)}$ is the number of off-diagonal nonzeros in $Q_l^{(i)}$. Hence, we have $\sum_{i=1}^{N} nz_l^{(i)}$ floating-point comparisons for N automata. Floating-point comparisons also take place when evaluating functions. The number depends on the ordering of automata and the number of functions to evaluate in each automaton. The number of floating-point comparisons performed due to one functional transition in $Q_l^{(i)}$ is $\prod_{j=i+1}^{N} n_j$. The total number of such floating-point comparisons is $\sum_{i=1}^{N-1} nf_i \prod_{j=i+1}^{N} n_j$, where nf_i is the number of functional transitions in $Q_l^{(i)}$. Based on our assumption regarding dependency among automata and their ordering, the Nth automaton cannot contain functional transitions and is excluded from the summation. To estimate the number of floating-point comparisons due to synchronizing events, we assume that each synchronizing event in the SAN is classified as a potential set $\mathcal{P}_r$, where r corresponds to the index of the event in $\mathcal{P}_r$, implying E potential sets. Let m_r be the master automaton of event r. Then, for synchronizing event r, we have $\prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$ floating-point comparisons, where $nz_{e_r}^{(j)}$ is the number of nonzeros in $Q_{e_r}^{(j)}$. The total number of such comparisons is $\sum_{r=1}^{E} \prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$. Hence, the total number of floating-point comparisons in Algorithm 1.3 is given by $\sum_{i=1}^{N} nz_l^{(i)} + \sum_{i=1}^{N-1} nf_i \prod_{j=i+1}^{N} n_j + \sum_{r=1}^{E} \prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$. This expression depends strongly on the number of functional transitions and synchronizing events in a SAN as well as the automata ordering. Thus, an optimal ordering that minimizes the total number of floating-point comparisons from the perspective of synchronizing events is one in which there is no automaton with index greater than σ_r uninvolved in event r.

For floating-point arithmetic operations incurred when handling synchronizing events, we make the following observations. Computation of a single nonzero transition originating from synchronizing event r requires $(N - \sigma_r)$ floating-point multiplications. For synchronizing event r, we compute $\prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}$ elements. Hence, the maximum number of floating-point multiplications in Algorithm 1.3 is $\sum_{r=1}^{E} [(N - \sigma_r) \prod_{j=\sigma_r, j\neq m_r}^{N} nz_{e_r}^{(j)}]$.

8.6 IAD algorithm for lumpable discrete-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 $l_{ij} = \max(P_{ij}u)$.

   (b) Solve the singular system $\tau(I - L) = 0$ subject to $\|\tau\|_1 = 1$ for $\tau = (\tau_1, \tau_2, \ldots, \tau_K)$.

3. Disaggregation:

   (a) Compute the row vector

$$z^{(it)} = (\tau_1 \frac{\pi_1^{(it-1)}}{\|\pi_1^{(it-1)}\|_1}, \tau_2 \frac{\pi_2^{(it-1)}}{\|\pi_2^{(it-1)}\|_1}, \cdots, \tau_K \frac{\pi_K^{(it-1)}}{\|\pi_K^{(it-1)}\|_1}).$$

   (b) Solve the K nonsingular systems of which the ith is given by

$$\pi_i^{(it)}(I - P_{ii}) = b_i^{(it)}$$

   for $\pi_i^{(it)}$, $i = 1, 2, \ldots, K$, where

$$b_i^{(it)} = \sum_{j>i} z_j^{(it)} P_{ji} + \sum_{j<i} \pi_j^{(it)} P_{ji}.$$

4. Test $\pi^{(it)}$ for convergence. If the desired accuracy is attained, then stop and take $\pi^{(it)}$ as the stationary probability vector of P. Else set $it = it+1$ and go to step 3.