STOCHASTIC COMPARISON ON NEARLY COMPLETELY DECOMPOSABLE MARKOV CHAINS

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
MASTER OF SCIENCE

By
Denizhan N. Alparslan
July, 2000

STOCHASTIC COMPARISON ON NEARLY COMPLETELY DECOMPOSABLE MARKOV CHAINS

A THESIS SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Denizhan N. Alparslan
July, 2000

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 Master of Science.

Assist. Prof. Dr. Tuğrul Dayar (Advisor)

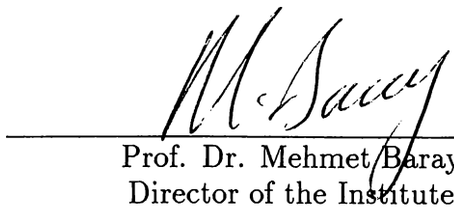
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Varol Akman

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 Master of Science.

Assist. Prof. Dr. Murat Fadıloğlu

Approved for the Institute of Engineering and Science:



Prof. Dr. Mehmet Baray
Director of the Institute

ABSTRACT

STOCHASTIC COMPARISON ON NEARLY COMPLETELY DECOMPOSABLE MARKOV CHAINS

Denizhan N. Alparslan
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Tuğrul Dayar
July, 2000

This thesis presents an improved version of a componentwise bounding algorithm for the steady state probability vector of nearly completely decomposable Markov chains. The given two-level algorithm uses aggregation and stochastic comparison with the strong stochastic (st) order. In order to improve accuracy, it employs reordering of states and a better componentwise probability bounding algorithm given st upper- and lower-bounding probability vectors. A thorough analysis of the algorithm from the point of view of irreducibility is provided. The bounding algorithm is implemented in sparse storage and its implementation details are given. Numerical results on an application of wireless Asynchronous Transfer Mode network show that there are cases in which the given algorithm proves to be useful in computing bounds on the performance measures of the system. An improvement in the algorithm that must be considered to obtain better bounds on performance measures is also presented at the end.

Keywords: Markov chains, near complete decomposability, stochastic comparison, st-order, reorderings, aggregation.

ÖZET

NEREDEYSE TAMAMEN BÖLÜNEBİLİR MARKOV ZİNCİRLERİ ÜZERİNDE RASSAL KARŞILAŞTIRMA

Denizhan N. Alparslan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Tuğrul Dayar
Temmuz, 2000

Bu tezde neredeyse tamamen bölünebilir Markov zincirlerinin değişmez durum olasılık dağılımları icin tek tek sınırlar veren bir sınırlandırma algoritmasının gelişmiş biçimi anlatılmaktadır. Sunulan bu iki seviyeli algoritma, birleştirmeye ve güçlü rassal (st) sıralama ile rassal karşılaştırmaya dayalıdır. Sonucun kesinliğinin arttırabilmesi için durumların yeniden sıralanması ve st bağıntısına göre üstten- ve alttan-sınırlayan olasılık dağılımlarından tek tek sınırların elde edilmesini sağlayan daha iyi bir algoritma ortaya konmuştur. Sınırlandırma algoritmasının indirgeme açısından eksiksiz bir analizi yapılmıstır. Bu algoritma seyrek saklama düzeninde programlanmış ve bu programlamanın ayrıntıları verilmiştir. Farklı zamanlı aktarma biçimi üzerine kurulmuş olan kablosuz bir ağ sisteminden elde edilen sayısal sonuçlar bu algoritmanın bazı durumlarda verilen sistemin başarım değerleri üzerinde sınırlar bulmada yararlı olabileceğini göstermektedir. Başarım değerleri üzerinde verilen sınırların daha iyi olabilmesi için algoritmada yapılması gereken iyileştirme en sonda belirtilmiştir.

Anahtar sözcükler: Markov zincirleri, neredeyse tamamen bölünebilirlik, rassal karşılaştırma, güçlü rassal sıralama, sıralama, birleştirme.

Acknowledgements

I would like to express my deep gratitude to my supervisor Assist. Prof. Dr. Tuğrul Dayar for his guidance, suggestions, invaluable encouragement, and patience throughout my thesis work. I also would like to thank Assoc. Prof. Dr. Nihal Pekergin from Université de Versailles-St.Quentin for her comments and help during this work. Finally, I would like to thank my committee members Prof. Dr. Varol Akman and Assist. Prof. Dr. Murat Fadıloğlu for reading the thesis and their comments.

I am grateful to my family for their infinite moral support, patience, and help.

To my parents and brothers

Contents

List of Figures

Chapter 1

Introduction

Most physical systems in the areas of engineering, science, and economics can be modeled by uniquely identifying all the states the system occupies. The transitions that are defined on the time axis among these states determine the future behavior of the system. Markov chains (MCs) is an effective tool in modeling and analyzing systems arising in areas such as queueing network analysis, computer systems performance evaluation, and large-scale economic modeling. With the help of MCs, performance measures such as blocking probabilities of finite buffers, average number of customers can be computed.

A set of states corresponds to each Markov chain. The number of states can be large enough to cause problems in Markovian modeling. This phenomenon is known as the state space explosion problem. In Markovian modeling, the system being modeled can occupy one state at a specific time instant and the future behavior of the system is determined by the transition probabilities or rates among states. The fundamental property in Markovian modeling is that the next state depends only on the current state and not on the past history. This is known as the *Markovian property* [21, p. 3].

A *stochastic process* $\{X(t),\ t \in \mathcal{T}\}$ is a collection of random variables. The index set $\mathcal{T}$ can be interpreted as the time axis of the process. When $\mathcal{T}$ is countable, the process is said to be a *discrete-time* process. If $\mathcal{T}$ is an interval

on the real line and can take any value in that interval, the process is referred to as a *continuous-time process*.

Let X_k represent the state of the system at time instant k and let the sequence of random variables $X_0, X_1, X_2, \ldots$ form a discrete-time stochastic process. This process forms a MC if it satisfies the Markovian property. Since we are observing the process at discrete time instants, it is referred to as a discrete-time Markov chain (DTMC). The conditional probabilities $p_{ij}(k) = Prob\{X_{k+1} = j \mid X_k = i\}$ are known as *one-step transition probabilities* of the DTMC. If these transition probabilities are independent of k, then we have a *time-homogeneous* DTMC. In this case, $Prob\{X_{k+1} = j \mid X_k = i\} = p_{ij}$ for any k.

On the other hand, suppose we have a continuous-time stochastic process $\{X(t),\ t \geq 0\}$ taking values in $[0, +\infty)$. This stochastic process is a MC if the distribution of the future $X(t+s)$, given the present $X(s)$ and the past, depends only on the present and the length of s, and is independent of the past (i.e., $X(t)$ possesses the Markovian property). Under this condition the process $X(t)$ is referred to as a continuous-time Markov chain (CTMC). If, in addition, $Prob\{X(t+s) = j \mid X(t) = i\}$ is independent of t but depends only on s, the CTMC is time-homogeneous. This thesis considers time-homogeneous MCs.

A DTMC can be represented by the matrix of transition probabilities, P, which has p_{ij} in row i and column j. All the entries of P are greater than or equal to zero, and its row sums are one. In other words, P is a stochastic matrix.

The situation is different for the continuous-time case. A CTMC is represented by the matrix of *transition rates*, Q. By discretizing the time axis, the probability of transition from one state to another in the interval of observation can be approximated. In this way, a CTMC can be transformed to a matrix of transition probabilities, which is dependent on the size of the observation interval. The transformation is known as uniformization [21, p. 19]. To speak more formally, the corresponding DTMC is obtained by considering transitions that take place at intervals of Δt. The interval Δt must be chosen sufficiently small to make probability of more than one transition in Δt negligible. The transition

probability matrix of the DTMC is given by the equation

$$P = \Delta t Q + I.$$

If $0 \leq \Delta t \leq (\max_i \mid q_{ii} \mid)^{-1}$, the matrix P is stochastic. To test the proposed algorithm, we consider examples that are modeled as CTMCs, and the corresponding discrete-time MCs are generated through uniformization.

Now, let us state some definitions concerning MCs. If the stochastic process, which is represented by the MC, can reach state j from state i, then j is said to be accessible from i. If, in addition to this, i is accessible from j, then i and j are called communicating states. Two states that communicate are said to be in the same class. The concept of communication may partition the state space of MC into a number of subsets. The MC is said to be *irreducible* if there is only one communicating class; that is, each state of the MC can be reached from every other state. Let f_j be the probability of returning to state j. In particular, if f_j=1, then state j is said to be recurrent; on the other hand, if f_j <1, then state j is said to be transient. Furthermore, if after leaving state j a return is possible only in a number of transitions that is a multiple of integer γ >1, then the state j is said to be periodic with period γ. If γ=1, then state j is said to be *aperiodic.*

By using symmetric permutations a DTMC can be transformed to the following normal form [21, p. 26]:

$$P = \begin{pmatrix} P_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & P_{22} & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & P_{kk} & 0 & 0 \\ P_{k+1,1} & P_{k+1,2} & & P_{k+1,k} & P_{k+1,k+1} & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ P_{m1} & P_{m2} & \cdots & P_{mk} & P_{m,k+1} & P_{mm} \end{pmatrix} \tag{1.1}$$

For $i \in \{1, \ldots, k\}$, the submatrices P_{ii} are stochastic and irreducible. Once the process enters one of the states corresponding to P_{ii}, $i \in \{1, \ldots, k\}$, it will remain there. Each of the P_{ii} corresponds to an essential subset of states. On the other hand, when $i \in \{k+1, \ldots, m\}$, the P_{ii} are substochastic and each one corresponds to a transient subset of states. If the process is in one of the transient subset

of states, it may leave that subset of states with a positive probability and not return back to the same subset. The applications we consider consist of a single essential subset of states (i.e., $k = 1$) and possibly many transient subset of states (i.e., $m \geq 0$).

Now let us denote by $\pi_j(k)$ the probability of finding the system in state j at step k for a DTMC and by $\pi_j(t)$ the probability of finding the system in state j at time t for a CTMC. For a finite, irreducible, discrete or continuous, time-homogeneous MC of n states, whose states are all aperiodic, the limiting probabilities of being in any state in the long run exists [12, p. 29]. Whenever this steady state probability distribution exists, it is a stationary probability distribution and is denoted by π_j for state j. The (row) vector $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ is known as the stationary probability vector and it satisfies $\pi P = \pi$ (or $\pi Q = 0$), where $\sum_{j=1}^{n} \pi_j = 1$.

In Markovian modeling, it is frequently the case that the state space of the model can be partitioned into disjoint subsets, with strong interactions among the states of a subset but weak interactions among the subsets themselves. Such problems are referred to as being nearly completely decomposable (NCD). NCD Markov chains [4],[15],[21] are irreducible stochastic matrices that can be symmetrically permuted to the block form

$$P_{n\times n} = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{pmatrix} \begin{matrix} n_1 \\ n_2 \\ \vdots \\ n_N \end{matrix} \tag{1.2}$$

in which nonzero elements of the off-diagonal blocks are small compared with those of the diagonal blocks [21, p. 286]. To permute the matrix into the almost block-diagonal form in equation (1.2), a pre-processing effort is needed. The larger the elements in the off-diagonal blocks, the less NCD the chain becomes. To summarize this more formally, let $P = \text{diag}(P_{11}, P_{22}, \ldots, P_{NN}) + F$. The diagonal blocks P_{ii} are square, of order n_i, with $n = \sum_{i=1}^{N} n_i$. 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 [9, p. 290]. Hence, NCD Markov chains are said to be ill-conditioned, and the smaller $\|F\|_\infty$ is, the more ill-conditioned P becomes [15, p. 258]. On the other hand, if P were reducible, it must be decomposed into its essential and transient subsets of states as in equation (1.1) and the analysis should continue on the essential subsets.

To compute performance measures of interest, either the long-run distribution of state probabilities (i.e., steady state analysis) or the probability distribution at a specific time instant (i.e., transient analysis) needs to be known. In this work, we focus on steady state analysis which requires the solution of a homogeneous system of linear equations with a singular coefficient matrix under a normalization constraint, (i.e., $\pi(I - P) = 0$ or $\pi Q = 0, \|\pi\|_1 = 1$) but the scope of this work can be extended to include transient analysis.

To each NCD MC corresponds an irreducible stochastic matrix, C, known as the coupling matrix [15]. Its (ij)th element is given by

$$c_{ij} = \frac{\pi_i}{\|\pi_i\|_1} P_{ij} e \qquad \forall i, j \in \{1, 2, \ldots, N\}. \tag{1.3}$$

Here e represents a column vector of all ones, and π_i, of size n_i, is obtained by partitioning π conformally with P in equation (1.2). The coupling matrix models the transitions of the system among NCD partitions and it cannot be computed without the knowledge of π. Note that it is the irreducibility of the NCD MC in the definition which guarantees the irreducibility of C.

For the partitioning in equation (1.2), the stochastic complement [15] of $P_{i,i}$ for $i \in \{1, 2, \ldots, N\}$ is given by

$$\overline{P}_{i,i} = P_{i,i} + P_{i,:}(I - P_i)^{-1} P_{:,i},$$

where $P_{i,:}$ is the $n_i \times (n - n_i)$ matrix composed of the ith row of blocks of P with $P_{i,i}$ removed, $P_{:,i}$ is the $(n-n_i) \times n_i$ matrix composed of the ith column of blocks of P with $P_{i,i}$ removed, and P_i is the $(n-n_i) \times (n-n_i)$ principal submatrix of P with ith row and ith column of blocks removed. The ith stochastic complement is the stochastic transition probability matrix of an irreducible MC of order n_i obtained

by observing the original process in the ith NCD partition. The conditional steady state probability vector of the ith NCD partition is $\pi_i/\|\pi_i\|_1$, and it may be computed by solving for the steady state vector of $\overline{P}_{i,i}$ (see [15] for details). However, each stochastic complement has an embedded matrix inversion which may require excessive computation.

The transient and steady state performance measures of a MC can be computed exactly in floating-point arithmetic. However the time it takes to obtain them can be very long. Stochastic comparison is a technique by which both performance measures of a MC may be bounded without having to compute them exactly. The applications of this technique exist in different areas of applied probability [20] and in practical problems of engineering [16], [17]. The stochastic comparison of MCs is discussed in detail in [13], [22], [14]. The comparison of two MCs requires comparing their transient probability vectors at each time instant according to a predefined order relation. Obviously, if steady states exist, stochastic comparison between their steady state probability vectors is also possible.

Sufficient conditions for the existence of stochastic comparison due to an order relation of two time-homogeneous MCs are given by the stochastic monotonicity and bounding properties of their one step transition probability matrices [13], [14]. In [23], this idea is used to devise an algorithm that constructs an optimal st-monotone (i.e., monotonicity due to the strong stochastic order relation) upper-bounding MC. Later, this algorithm is used to compute stochastic bounds on performance measures that are defined on a totally ordered and reduced state space [1]. However, the given algorithm may provide loose bounds when the dynamics of the underlying system is not considered.

The bounded aggregation method discussed in [5] and [19] uses polyhedra theory to compute the best possible componentwise upper and lower bounds on the steady state probability vector of a given NCD MC. In [24], a different componentwise bounding algorithm which trades accuracy to solution time is given. It is a two-level algorithm using aggregation and stochastic comparison with the strong stochastic (st) order. However, it has not been implemented

and tested on any applications; moreover, its theoretical analysis lacks essential components.

This thesis is an extension of the work in [24]. An improved, coherent, and readily understandable form of the algorithm is given. We remedy the situation regarding theoretical analysis. The improvements include the possibility of re-ordering the states in each NCD block and the introduction of a new st-monotone lower-bounding matrix construction algorithm. In addition to these, a better componentwise probability bounding algorithm is given. Finally, the proposed algorithm is implemented in sparse storage, meaning zero entries are not stored.

In chapter 2, we provide the background on MCs, stochastic comparison, irreducibility of matrices and direct methods for solving linear systems. In chapter 3, we introduce the improved algorithm. The irreducibility analysis is provided in chapter 4. The details of the sparse implementation are given in chapter 5. Numerical results on a current application in mobile communications is provided in chapter 6. In chapter 7, we conclude.

Chapter 2

Theoretical Background

This chapter provides the background on stochastic comparison of MCs and direct methods for solving them.

2.1 Stochastic Comparison

In this work, we are interested in obtaining bounds on the steady state performance measures of problems without having to compute them exactly. In doing this, we use stochastic comparison. The objective is to trade accuracy with solution time.

For the stochastic comparison of random variables, an ordering relation is needed. The relation must be reflexive and transitive, but not necessarily antisymmetric. There are different stochastic ordering relations which satisfies these properties and the most well known is the strong stochastic ordering (i.e., $\leq_{st}$). Intuitively speaking, two random variables X and Y (which take values on a totally ordered space) being comparable in the strong stochastic sense (i.e., $X \leq_{st} Y$) means that it is less probable for X to take larger values than Y (see [20], [22]).

In this thesis, we use strong stochastic ordering whose definition is given below. For further information on stochastic comparison, we refer the reader to [22].

DEFINITION 2.1 *Let X and Y be random variables taking values on a totally ordered space. Then X is said to be less than Y in the strong stochastic sense, that is, $X \leq_{st} Y$ iff*

$$E[f(X)] \leq E[f(Y)]$$

for all nondecreasing functions f whenever the expectations exist.

DEFINITION 2.2 *Let X and Y be random variables taking values on the finite state space $\{1, 2, \ldots, n\}$. Let p and q be probability vectors such that*

$$p_i = Prob(X = i) \quad and \quad q_i = Prob(Y = i) \quad for \quad i \in \{1, 2, \ldots, n\}.$$

Then X is said to be less than Y in the strong stochastic sense, that is, $X \leq_{st} Y$ iff

$$\sum_{i=j}^{n} p_i \leq \sum_{i=j}^{n} q_i \quad for \quad j = n, n-1, \ldots, 1.$$

COROLLARY 2.1 *If X and Y are random variables taking values on the finite state space $\{1, 2, \ldots, n\}$ with probability vectors p and q respectively, and $X \leq_{st} Y$, then*

$$p_n \leq q_n \quad and \quad p_1 \geq q_1.$$

The comparison of MCs has been largely studied in [13], [22], [14]. We use the following definition (Definition 4.1.2 of [22, p. 59]) to compare MCs.

DEFINITION 2.3 *Let $\{X(t),\ t \in \mathcal{T}\}$ and $\{Y(t),\ t \in \mathcal{T}\}$ be two time-homogeneous MCs. Then $\{X(t),\ t \in \mathcal{T}\}$ is said to be less than $\{Y(t),\ t \in \mathcal{T}\}$ in the strong stochastic sense, that is, $\{X(t)\} \leq_{st} \{Y(t)\}$ iff*

$$X(t) \leq_{st} Y(t) \qquad \forall t \in \mathcal{T}.$$

Moreover, it is shown in Theorem 3.4 of [14, p. 355] that monotonicity and comparability of the probability transition matrices of time-homogeneous MCs yield sufficient conditions for their stochastic comparison, which is summarized in:

THEOREM 2.1 *Let P and $\tilde{P}$ be stochastic matrices respectively characterizing time-homogeneous MCs $X(t)$ and $Y(t)$. Then $\{X(t),\ t \in \mathcal{T}\} \leq_{st} \{Y(t),\ t \in \mathcal{T}\}$ if*

- $X(0) \leq_{st} Y(0)$,
- *st-monotonicity of at least one of the probability transition matrices holds, that is,*

$$\textit{either} \quad P_{i,*} \leq_{st} P_{j,*} \quad \textit{or} \quad \tilde{P}_{i,*} \leq_{st} \tilde{P}_{j,*} \quad \forall i,j \quad \textit{such that} \quad i \leq j,$$

- *st-comparability of the transition matrices holds, that is,*

$$P_{i,*} \leq_{st} \tilde{P}_{i,*} \qquad \forall i.$$

Here $P_{i,*}$ refers to row i of P.

Next, we discuss two intimately related direct numerical solution methods for the computation of the stationary distribution of a MC.

2.2 Direct Methods

The method used in this thesis to find componentwise upper and lower bounds on the steady state probabilities of a given MC requires the solution of a homogeneous singular system of linear equations. In other words, we are concerned with solving

$$\pi(I - P) = 0$$

for a nonnegative π with unit 1-norm, where P is the transition probability matrix of the MC of interest. Hence, we are seeking the nontrivial solution whose existence is guaranteed when P is irreducible.

The nontrivial solution can be obtained by direct or iterative methods. Direct methods compute the solution in a fixed number of floating-point operations. On the other hand, iterative methods begin from some approximation and (hopefully) converge to the solution after an unknown number of iterations. There are many types of iterative methods and they are most commonly used in large-scale Markovian analysis. In this thesis, we use direct methods because the systems to be solved are of moderate order. We use two types of direct methods: Gaussian elimination (GE) and the method of Grassmann-Taksar-Heyman (GTH), which is a more stable variant of GE.

2.2.1 Gaussian Elimination

Recall that we are seeking the nontrivial solution of $\pi P = \pi$ or equivalently $(I - P^T)\pi^T = 0$, where π^T is the transpose of the row vector π. Let $A = (I - P^T)$ and $x = \pi^T$. It is known that A is a singular M-matrix [3, p. 147]. In this way, the linear system is transformed to the following system of homogeneous equations:

$$Ax = 0.$$

Since we are seeking the stationary vector of an irreducible MC, the coefficient matrix A must be singular (i.e., its determinant is zero), otherwise the only solution to this system is the zero vector.

GE may be viewed as transforming the system $Ax = 0$ to an equivalent system $Ux = 0$ in which the matrix U is upper triangular. Obtaining U from A is called the reduction phase and requires $n - 1$ steps, where n is the order of A. In the ith elimination step, all nonzero elements below the ith diagonal element in the reduced matrix are eliminated by adding a multiple of row i to all the rows below row i. More formally, let $A^{(i)}$ be the reduced matrix obtained at the end of the

ith step of elimination with $A^{(0)} = A$. Then the elements $a_{kl}^{(i)}$ are given by

$$a_{kl}^{(i)} = \begin{cases} a_{kl}^{(i-1)} & \text{for } k \leq i \text{ and } l = 1, 2, \ldots, n \\ a_{kl}^{(i-1)} - m_{ki} a_{il}^{(i-1)} & \text{for } k > i \text{ and } l = 1, 2, \ldots, n \end{cases},$$

where the multipliers are given by $m_{ki} = a_{ki}^{(i-1)} / a_{ii}^{(i-1)}$. Obviously, for $k > i$, $a_{ki}^{(i)} = 0$. The elements $a_{ii}^{(i)}$ are called pivots and they must be nonzero if the algorithm is to carry on satisfactorily. The irreducibility of P ensures that the pivots are nonzero in exact arithmetic. At the end of the elimination, $A^{(n-1)} = U$ is computed. Inherently, U is obtained by pre-multiplying the coefficient matrix A by a nonsingular unit lower-triangular matrix. That is,

$$L^{-1}A = U,$$

where L is the unit lower triangular matrix (i.e., $l_{ij} = 0$, for $i < j$ and $l_{ii} = 1$). The singularity of A and the nonsingularity of L imply that U must be singular. Furthermore, it can be shown that the last row of U must be zero. Hence, we have

$$(LU)x = 0$$

Since L is nonsingular, the only solution is available through $Ux = 0$. If we proceed to solve $Ux = 0$, (the back-substitution phase), we can assign any nonzero value, say η, to x_n because the last row of U is zero. We can determine the remaining elements of the vector x in terms of η and compute the solution after normalizing x according to the constraint $\|x\|_1$. When A is dense, GE requires $O(n^3)$ floating point operations (flops) to reach this solution, and the space requirement is $O(n^2)$. Clearly, the time complexity of GE increases rapidly with the size of the problem.

Notice that to obtain the homogeneous linear system $Ax = 0$, we transform $\pi P = \pi$ to $(I - P^T)\pi^T = 0$. This transformation has an important consequence: we do not have to keep the entries of L at all. We could have also tried to solve $\pi(I - P) = 0$. However, this requires one to save both the L and U factors when row reductions are carried out. The main drawback of working with the non-transposed version of the system is this, and therefore, in this thesis we work with stochastic matrices in transposed form.

2.2.2 The method of Grassmann-Taksar-Heyman

In computing the stationary probability vector of irreducible MCs, we consider one more direct method. The GTH method [11] is used because of the difficult nature of some input matrices. For certain types of problems, small differences in the input data may result in large differences in the results. Such problems are called *ill-conditioned.* When small differences in the data always lead to small differences in the results, the problem is said to be *well-conditioned.* Ill-conditioning and well-conditioning are properties of the problem rather than the algorithm used to solve the problem. On the other hand, an algorithm is a computer based implementation of basic arithmetic operations and usually generates errors. Because of this reason, algorithms are said to compute an approximation to the exact solution. The accuracy of this approximate solution is of significant importance. A stable algorithm is one that yields a solution that is almost exact for a well-conditioned problem. It should not be expected to give an accurate solution for an ill-conditioned problem. However, it should not introduce unacceptable errors which originate from the nature of the algorithm either. For unstable algorithms we can not guarantee the accuracy of the solution.

In the GTH method, pivot elements are computed by summing the off-diagonal elements below the pivot and negating this sum. This approach works, because the column sums of the bottom rightmost submatrix of order $(n-i)$ in $A^{(i)}$ are zero in each step of the elimination phase. It is known that subtractions can lead to loss of significance in the representation of real numbers on the computer. The GTH method involves no subtractions, and therefore yields a more stable algorithm than GE [9]. If π_j is the exact value and π_j^{GTH} is the approximate value computed by GTH for a MC of order n, the entrywise relative error $| \pi_j - \pi_j^{GTH} | / \pi_j = O(n^3 u)$, where u is the unit roundoff. It is clear that GTH requires slightly more floating point operations than GE due to the summations to compute the pivots. In this thesis, accuracy of the solution is of importance since we will be computing the stationary vectors of NCD MCs. We refer to [21, p. 84] for more information about the GTH method. The implementation details of this method in sparse storage are given in subsequent chapters.

Chapter 3

Componentwise Bounding Algorithm

This thesis is focused on finding componentwise bounds for the steady state vector of NCD MCs without solving them exactly. Our componentwise bounding algorithm (see Algorithm 1) is based on the two-level algorithm in [24] that uses aggregation and stochastic comparison with the st-order. Aggregation is the process of forming the coupling matrix given by equation (1.3).

In Step 0 of our algorithm, the given Markov chain P is permuted to an NCD block form as in equation (1.2). This form is of significant importance and is determined by the algorithm in [6]. The algorithm in [6] first constructs an undirected graph whose vertices are states of the MC by introducing edges between vertices i and j if $p_{ij} \geq \epsilon$ or $p_{ji} \geq \epsilon$ for a particular value of the decomposability parameter ϵ. Then, it determines the CCs of this undirected graph. Each CC is a subset of the NCD partitioning. The partitioning of states returned by this algorithm is based on the (user specified) decomposability parameter ϵ. By a balanced partitioning, we mean one in which the order of the diagonal blocks in equation (1.2) do not differ significantly from each other. In this thesis, we consider balanced NCD partitionings of the state space.

After obtaining the NCD partition, we apply the first level of the algorithm.

At this level (see Step 1), componentwise upper and lower bounds on the conditional steady state probability vector of each NCD subset of states are computed for the partition of P in Step 0. This is achieved by computing st-monotone upper- and lower-bounding matrices for each NCD subset of states (see Algorithms 2–6). The stochastic matrices that are input to the bounding matrix construction algorithms are generated in Step 1.b using the P_{ii}. The st-monotone bounding matrices for each NCD block are obtained in Step 1.c. To construct the st-monotone upper-bounding matrix, we use the algorithm in [1] as in [24] (see Algorithm 5), but devise and use a new st-monotone lower-bounding matrix construction algorithm (see Algorithm 6) whose optimality is proved in the next section.

At the second level (see Step 2), st-monotone upper- and lower-bounding matrices for the coupling matrix, C, corresponding to the same partition of P are computed using the conditional steady state probability bounding vectors obtained at the first level again using Algorithms 2–6. From these two matrices, lower and upper unconditioning steady state probability bounds for the conditional steady state probability bounding vectors are computed and componentwise bounds for the steady state vector of P are given. Recall that, we cannot compute C exactly since we do not know the exact steady state vector of P. We compute st-bounding matrices for C.

Obviously, the order of states within each NCD partition affects the quality of the bounds that may be obtained by the stochastic comparison approach [7] due to the conditions of st-monotonicity and st-comparability in Theorem 2.1. To obtain tighter probability bounds, we permute one of the states within each NCD partition to be the last and order the remaining states in the same partition using the heuristic given in Algorithm 10 (see Step 1.a). The state to be permuted to the end of each NCD block is chosen as the state which has the largest self transition probability among the states in the same NCD partition followed by a simple tie-breaking rule if needed. We do not use ordering at the second level of the algorithm since the resulting matrices are highly diagonally dominant implying a small gain (if at all). For more discussion on this ordering heuristic we refer the reader to [7, p. 17].

Neither of the two bounding matrices computed for each NCD block may be irreducible [1]. However, as we prove in the next section, they both have one essential subset of states. This is also true for the st-monotone bounding matrices computed for C. In other words, if we try to permute the bounding matrices to the form given in equation (1.1), we obtain $k = 1$. After identifying and removing the transient states (see Steps 1.d and 2.d), the resulting irreducible matrices are solved for their steady state vectors. In extracting the essential subsets from the bounding matrices, we use a slightly different version of the strongly connected component (SCC) search algorithm in a graph. The details of this algorithm are given in Chapter 5. We remark that Steps 1.d and 2.d should omit the removal of transient states and replace the steady solution process with a transient solution procedure when performing transient analysis.

3.1 Algorithms

ALGORITHM 1. Componentwise bounding algorithm for the steady state vector of NCD MCs:

0. Find a (balanced) NCD partitioning of P and symmetrically permute it to the form in equation (1.2). Let $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_N\}$ be the resulting state space partition.

1. for $i = 1, 2, \ldots, N$,

   a. Choose a state from $\mathcal{S}_i$, say f_i, make it the last state and find the ordering of the remaining states in $\mathcal{S}_i$ with respect to f_i by Algorithm 10. Symmetrically permute $P_{i,i}$ according to the resulting ordering.

   b. Compute the two stochastic matrices $\overline{S}_i$ and $\underline{S}_i$ of order n_i corresponding to $P_{i,i}$ by Algorithms 2 and 3, respectively (see Remark 3.1).

   c. Compute the st-monotone upper-bounding matrix $\overline{Q}_i$ of order n_i corresponding to $\overline{S}_i$ by Algorithm 5 and the st-monotone lower-bounding matrix $\underline{Q}_i$ of order n_i corresponding to $\underline{S}_i$ by Algorithm 6.

d. Extract the irreducible submatrices of $\overline{Q}_i$ and $\underline{Q}_i$ and solve the corresponding systems of equations for their steady state vectors $\overline{\pi}_i^{st}$ and $\underline{\pi}_i^{st}$, respectively. Place zero steady state probabilities for transient states in each vector.

e. Compute the componentwise bounding vectors π_i^{sup} and π_i^{inf} on the conditional steady state probability vector corresponding to $\mathcal{S}_i$ from $\overline{\pi}_i^{st}$ and $\underline{\pi}_i^{st}$ by Algorithm 7.

2. a. Compute U and L of order N using π_i^{sup} and π_i^{inf}, $i \in \{1, 2, \ldots, N\}$ by Algorithms 8 and 9, respectively.

b. Compute the two stochastic matrices $\overline{S}$ and $\underline{S}$ of order N corresponding to L and U by Algorithms 2 and 3, respectively.

c. Compute the st-monotone upper-bounding matrix $\overline{Q}$ of order N corresponding to $\overline{S}$ by Algorithm 5 and the st-monotone lower-bounding matrix $\underline{Q}$ of order N corresponding to $\underline{S}$ by Algorithm 6.

d. Extract the irreducible submatrices of $\overline{Q}$ and $\underline{Q}$ and solve the corresponding systems of equations for their steady state vectors $\overline{\xi}^{st}$ and $\underline{\xi}^{st}$, respectively. Place zero steady state probabilities for transient states in each vector.

e. Compute the componentwise bounding vectors ξ^{sup} and ξ^{inf} on the steady state probability vector corresponding to C from $\overline{\xi}^{st}$ and $\underline{\xi}^{st}$ by Algorithm 7.

3. Compute the componentwise steady state probability upper- and lower-bounding vectors for $\mathcal{S}_i$ respectively as $\xi_i^{sup}\pi_i^{sup}$ and $\xi_i^{inf}\pi_i^{inf}$, $i \in \{1, 2, \ldots, N\}$.

REMARK 3.1 *When Algorithms 2 and 3 are invoked for the substochastic matrices $P_{i,i}$, $L = P_{i,i}$ and $U = L + \Delta$, where $d = e - Le$ and $\Delta = [d\ d\ \cdots\ d]$.*

ALGORITHM 2. Construction of stochastic matrix $\overline{S}$ corresponding to L and U of order m:

$\Delta = U - L;$
for $i = 1, 2, \ldots, m,$
  $\epsilon_i^{(0)} = 1 - \sum_{j=1}^{m} l_{i,j};$
for $i = 1, 2, \ldots, m,$
  for $j = m, m-1, \ldots, 1,$
    $\overline{s}_{i,j} = l_{i,j} + \min(\delta_{i,j}, (\epsilon_i^{(m-j)})^+);$
    $\epsilon_i^{(m-j+1)} = \epsilon_i^{(m-j)} - \delta_{i,j};$

ALGORITHM 3. Construction of stochastic matrix $\underline{S}$ corresponding to L and U of order m:

$\Delta = U - L;$
for $i = 1, 2, \ldots, m,$
  $\epsilon_i^{(0)} = 1 - \sum_{j=1}^{m} l_{i,j};$
for $i = 1, 2, \ldots, m,$
  for $j = 1, 2, \ldots, m,$
    $\underline{s}_{i,j} = l_{i,j} + \min(\delta_{i,j}, (\epsilon_i^{(j-1)})^+);$
    $\epsilon_i^{(j)} = \epsilon_i^{(j-1)} - \delta_{i,j};$

ALGORITHM 4. Construction of matrix B (to be used in Algorithms 5 and 6) corresponding to stochastic matrix S of order m:

for $i = 1, 2, \ldots, m,$
  $b_{i,m} = s_{i,m};$
  for $j = m-1, m-2, \ldots, 1,$
    $b_{i,j} = b_{i,j+1} + s_{i,j};$

ALGORITHM 5. Construction of st-monotone upper-bounding matrix $\overline{Q}$ corresponding to stochastic matrix $\overline{S}$ of order m:

Compute B by Algorithm 4 for $\overline{S}$ of order m.
$\overline{q}_{1,m} = b_{1,m}$;
for $i = 2, 3, \ldots, m$,
  $\overline{q}_{i,m} = \max(b_{i,m}, \overline{q}_{i-1,m})$;
for $l = m-1, m-2, \ldots, 1$,
  $\overline{q}_{1,l} = b_{1,l} - b_{1,l+1}$;
  for $i = 2, 3, \ldots, m$,
    $\overline{q}_{i,l} = \max(b_{i,l}, \sum_{j=l}^{m} \overline{q}_{i-1,j}) - \sum_{j=l+1}^{m} \overline{q}_{i,j}$;

ALGORITHM 6. Construction of st-monotone lower-bounding matrix $\underline{Q}$ corresponding to stochastic matrix $\underline{S}$ of order m:

Compute B by Algorithm 4 for $\underline{S}$ of order m.
for $l = 1, 2, \ldots, m-1$,
  $\underline{q}_{m,l} = b_{m,l} - b_{m,l+1}$;
  for $i = m-1, m-2, \ldots, 1$,
    $\underline{q}_{i,l} = \max(1 - b_{i,l+1}, \sum_{j=1}^{l} \underline{q}_{i+1,j}) - \sum_{j=1}^{l-1} \underline{q}_{i,j}$;
$\underline{q}_{m,m} = b_{m,m}$;
for $i = m-1, m-2, \ldots, 1$,
  $\underline{q}_{i,m} = 1 - \sum_{j=1}^{m-1} \underline{q}_{i,j}$;

ALGORITHM 7. Computation of componentwise probability bounding vectors v^{sup} and v^{inf} given st upper- and lower-bounding probability vectors $\overline{v}^{st}$ and $\underline{v}^{st}$ of length m:

$v_m^{sup} = \overline{v}_m^{st}$;
$v_m^{inf} = \underline{v}_m^{st}$;
for $j = m-1, m-2, \ldots, 1$,

$$v_j^{sup} = \sum_{k=j}^{m} \overline{v}_k^{st} - \sum_{k=j+1}^{m} \underline{v}_k^{st};$$
$$v_j^{inf} = (\sum_{k=j}^{m} \underline{v}_k^{st} - \sum_{k=j+1}^{m} \overline{v}_k^{st})^{+};$$

ALGORITHM 8. Computation of componentwise upper-bounding matrix U for C of order N using P and π_i^{sup}, $i \in \{1, 2, \ldots, N\}$:

for $i = 1, 2, \ldots, N$,
  for $j = 1, 2, \ldots, N$,
    $u_{i,j} = \min(\pi_i^{sup} P_{i,j} e, \max(P_{i,j} e))$;

ALGORITHM 9. Computation of componentwise lower-bounding matrix L for C of order N using P and π_i^{inf}, $i \in \{1, 2, \ldots, N\}$:

for $i = 1, 2, \ldots, N$,
  for $j = 1, 2, \ldots, N$,
    $l_{i,j} = \max(\pi_i^{inf} P_{i,j} e, \min(P_{i,j} e))$;

ALGORITHM 10. Determining the ordering of NCD block of size m which is permuted to make the selected state last. The ordering is kept in the vector *index*.

$is = m - 1$;
for $j = 1, 2, \ldots, is$,
  $state = j$;
  $\mathcal{I} = \mathcal{I} \cup \{state\}$;
end;
$index_m = m$;
while $is > 0$ do
  $\mathcal{I}_t = \{k \mid k \in \mathcal{I}, p_{km} = \max_{i \in \mathcal{I}} p_{im}\}$;
  if $\#(\mathcal{I}_t) > 1$ then

$j = 1$;
while $(m - j > is)$ and $(\#(\mathcal{I}_t) > 1)$ do
$\mathcal{I}_{tt} = \{k \mid k \in \mathcal{I}_t, p_{k,index_{m-j}} = \max_{i \in \mathcal{I}_t} p_{i,index_{m-j}}\}$;
$\mathcal{I}_t = \mathcal{I}_{tt}$;
if $\#(\mathcal{I}_t) > 1$ then $j = j + 1$
else let the one in $\mathcal{I}_t$ be k;
end
else let the one in $\mathcal{I}_t$ be k;
if $\#(\mathcal{I}_t) > 1$ then
if $p_{mm} < p_{km}, k \in \mathcal{I}_t$ then $\mathcal{I}_{tt} = \{k \mid k \in \mathcal{I}_t, p_{mk} = \max_{i \in \mathcal{I}_t} p_{mi}\}$
else $\mathcal{I}_{tt} = \{k \mid k \in \mathcal{I}_t, p_{mk} = \min_{i \in \mathcal{I}_t} p_{mi}\}$;
if $\#(\mathcal{I}_{tt}) > 1$ then
choose one from $\mathcal{I}_{tt}$ randomly; let it be k
else let the one in $\mathcal{I}_{tt}$ be k;
$index_{is} = k$;
$is = is - 1$;
$\mathcal{I} = \mathcal{I} - \{k\}$;
end;

3.2 Numerical Example

In this section, we give a numerical example due to Courtois [5] and apply Algorithm 1 to obtain componentwise bounds for its steady state vector. The Courtois matrix is given by

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \left(\begin{array}{cccccccc} 0.85 & 0 & 0.149 & 0.0009 & 0 & 0.00005 & 0 & 0.00005 \\ 0.1 & 0.65 & 0.249 & 0 & 0.0009 & 0.00005 & 0 & 0.00005 \\ 0.1 & 0.8 & 0.0996 & 0.0003 & 0 & 0 & 0.0001 & 0 \\ \hline 0 & 0.0004 & 0 & 0.7 & 0.2995 & 0 & 0.0001 & 0 \\ 0.0005 & 0 & 0.0004 & 0.399 & 0.6 & 0.0001 & 0 & 0 \\ \hline 0 & 0.00005 & 0 & 0 & 0.00005 & 0.6 & 0.2499 & 0.15 \\ 0.00003 & 0 & 0.00003 & 0.00004 & 0 & 0.1 & 0.8 & 0.0999 \\ 0 & 0.00005 & 0 & 0 & 0.00005 & 0.1999 & 0.25 & 0.55 \end{array}\right).$$

In Step 0, we choose a degree of decomposability of 0.001 and obtain the partitioning $\{S_1, S_2, S_3\}$, where $S_1 = \{1,2,3\}$, $S_2 = \{4,5\}$ and $S_3 = \{6,7,8\}$. This is an NCD partitioning with degree of coupling 0.001 (i.e., $\|F\|_\infty = 0.001$). The corresponding NCD blocks are

$$P_{1,1}=\begin{pmatrix} 0.85 & 0 & 0.149 \\ 0.1 & 0.65 & 0.249 \\ 0.1 & 0.8 & 0.0996 \end{pmatrix}, P_{2,2}=\begin{pmatrix} 0.7 & 0.2995 \\ 0.399 & 0.6 \end{pmatrix},$$

$$P_{3,3}=\begin{pmatrix} 0.6 & 0.2499 & 0.15 \\ 0.1 & 0.8 & 0.0999 \\ 0.1999 & 0.25 & 0.55 \end{pmatrix}.$$

After this initial step we apply Step 1.a. States 1,4, and 7 in P are chosen as the last states in the NCD blocks 1,2, and 3 respectively. Using these last states in Step 1.a, we apply Algorithm 10 to find the ordering within each NCD block. The algorithm returns the orderings (3,2,1), (2,1), and (1,3,2) for the NCD blocks 1,2, and 3 respectively. If we symmetrically permute the P_{ii} with respect to these orderings, the blocks become

$$P_{1,1} = \begin{matrix} 3 \\ 2 \\ 1 \end{matrix}\begin{pmatrix} 0.0996 & 0.8 & 0.1 \\ 0.249 & 0.65 & 0.1 \\ 0.149 & 0 & 0.85 \end{pmatrix}, \; P_{2,2} = \begin{matrix} 5 \\ 4 \end{matrix}\begin{pmatrix} 0.6 & 0.399 \\ 0.2995 & 0.7 \end{pmatrix},$$

$$P_{3,3} = \begin{matrix} 6 \\ 8 \\ 7 \end{matrix}\begin{pmatrix} 0.6 & 0.15 & 0.2499 \\ 0.1999 & 0.55 & 0.25 \\ 0.1 & 0.0999 & 0.8 \end{pmatrix}.$$

Obviously, the P_{ii} are substochastic. Step 1.b generates 2 stochastic matrices for each of the (permuted) NCD blocks, which are given by

$$\overline{S}_1 = \begin{pmatrix} 0.0996 & 0.8 & 0.1004 \\ 0.249 & 0.65 & 0.101 \\ 0.149 & 0 & 0.851 \end{pmatrix}, \; \underline{S}_1 = \begin{pmatrix} 0.1 & 0.8 & 0.1 \\ 0.25 & 0.65 & 0.1 \\ 0.15 & 0 & 0.85 \end{pmatrix},$$

$$\overline{S}_2 = \begin{pmatrix} 0.6 & 0.4 \\ 0.2995 & 0.7005 \end{pmatrix}, \; \underline{S}_2 = \begin{pmatrix} 0.601 & 0.399 \\ 0.3 & 0.7 \end{pmatrix},$$

$$\overline{S}_3 = \begin{pmatrix} 0.6 & 0.15 & 0.25 \\ 0.1999 & 0.55 & 0.2501 \\ 0.1 & 0.0999 & 0.8001 \end{pmatrix}, \underline{S}_3 = \begin{pmatrix} 0.6001 & 0.15 & 0.2499 \\ 0.2 & 0.55 & 0.25 \\ 0.1001 & 0.0999 & 0.8 \end{pmatrix}.$$

Using these stochastic matrices, in Step 1.c we compute st-monotone upper-bounding and lower bounding matrices for each of the NCD blocks. The bounding matrices are given by

$$\overline{Q}_1 = \begin{pmatrix} 0.0996 & 0.8 & 0.1004 \\ 0.0996 & 0.7994 & 0.101 \\ 0.0996 & 0.0494 & 0.851 \end{pmatrix}, \underline{Q}_1 = \begin{pmatrix} 0.25 & 0.65 & 0.1 \\ 0.25 & 0.65 & 0.1 \\ 0.15 & 0 & 0.85 \end{pmatrix},$$

$$\overline{Q}_2 = \begin{pmatrix} 0.6 & 0.4 \\ 0.2995 & 0.7005 \end{pmatrix}, \underline{Q}_2 = \begin{pmatrix} 0.601 & 0.399 \\ 0.3 & 0.7 \end{pmatrix},$$

$$\overline{Q}_3 = \begin{pmatrix} 0.6 & 0.15 & 0.25 \\ 0.1999 & 0.55 & 0.2501 \\ 0.1 & 0.0999 & 0.8001 \end{pmatrix}, \underline{Q}_3 = \begin{pmatrix} 0.6001 & 0.15 & 0.2499 \\ 0.2 & 0.55 & 0.25 \\ 0.1001 & 0.0999 & 0.8 \end{pmatrix}.$$

All of the upper and lower bounding st-monotone matrices for the NCD blocks of the Courtois example turn out to be irreducible. In other words, they do not have any transient states. In Step 1.d we solve the st-monotone bounding matrices directly for their steady state vectors using GTH. The $\overline{\pi}_i^{st}$ and $\underline{\pi}_i^{st}$ in 6 decimal digits of precision are

$$\overline{\pi}_1^{st} = [0.099600, 0.496639, 0.403761],\ \underline{\pi}_1^{st} = [0.210000, 0.390000, 0.400000],$$

$$\overline{\pi}_2^{st} = [0.428163, 0.571837],\ \underline{\pi}_2^{st} = [0.429185, 0.570815],$$

$$\overline{\pi}_3^{st} = [0.240679, 0.203597, 0.555724],\ \underline{\pi}_3^{st} = [0.240882, 0.203616, 0.555502].$$

Using these vectors, in Step 1.e. we compute componentwise bounding vectors on the conditional steady state probability vectors for the NCD blocks as

$$\pi_1^{sup} = [0.210000, 0.500400, 0.403761],\ \pi_1^{inf} = [0.099600, 0.386239, 0.400000],$$

$$\pi_2^{sup} = [0.429185, 0.571837],\ \pi_2^{inf} = [0.428163, 0.570815],$$

$$\pi_3^{sup} = [0.240882, 0.203819, 0.555724],\ \pi_3^{inf} = [0.240679, 0.203393, 0.555502].$$

Since Step 1 of Algorithm 1 is over, we start executing Step 2. In Step 2.a, we generate U and L of order 3 using π_i^{sup} and π_i^{inf}, $i \in \{1,2,3\}$, as

$$U = \begin{pmatrix} 0.999600 & 0.000877 & 0.000100 \\ 0.000615 & 0.999500 & 0.000100 \\ 0.000056 & 0.000044 & 0.999900 \end{pmatrix}, L = \begin{pmatrix} 0.999000 & 0.000737 & 0.000100 \\ 0.000614 & 0.999000 & 0.000100 \\ 0.000056 & 0.000044 & 0.999900 \end{pmatrix}.$$

The stochastic matrices $\overline{S}$ and $\underline{S}$ corresponding to L and U computed in Step 2.b are

$$\overline{S} = \begin{pmatrix} 0.999023 & 0.000877 & 0.000100 \\ 0.000614 & 0.999286 & 0.000100 \\ 0.000056 & 0.000044 & 0.999900 \end{pmatrix}, \underline{S} = \begin{pmatrix} 0.999163 & 0.000737 & 0.000100 \\ 0.000615 & 0.999285 & 0.000100 \\ 0.000056 & 0.000044 & 0.999900 \end{pmatrix}.$$

In Step 2.c we obtain the st-monotone upper and lower bounding matrices

$$\overline{Q} = \begin{pmatrix} 0.999023 & 0.000877 & 0.000100 \\ 0.000614 & 0.999286 & 0.000100 \\ 0.000056 & 0.000044 & 0.999900 \end{pmatrix}, \underline{Q} = \begin{pmatrix} 0.999163 & 0.000737 & 0.000100 \\ 0.000615 & 0.999285 & 0.000100 \\ 0.000056 & 0.000044 & 0.999900 \end{pmatrix}.$$

These two bounding matrices are also irreducible. In step 2.d, we solve them for their steady state vectors and obtain

$$\overline{\xi}^{st} = [0.210388, 0.289612, 0.500000],$$

$$\underline{\xi}^{st} = [0.230836, 0.269164, 0.500000].$$

In Step 2.e, the componentwise bounding vectors on the steady state probability vector corresponding to C using $\overline{\xi}^{st}$ and $\underline{\xi}^{st}$ are computed as

$$\xi^{sup} = [0.230836, 0.289612, 0.500000],$$

$$\xi^{inf} = [0.210388, 0.269164, 0.500000].$$

In Step 3 of Algorithm 1, we compute the componentwise steady state probability upper- and lower-bounding vectors for each NCD block as

$$\xi_1^{sup}\pi_1^{sup} = [0.048476, 0.115510, 0.093203],\ \xi_1^{inf}\pi_1^{inf} = [0.020955, 0.081260, 0.084155],$$

$$\xi_2^{sup}\pi_2^{sup} = [0.124297, 0.165611],\ \xi_2^{inf}\pi_2^{inf} = [0.115246, 0.153643],$$

$$\xi_3^{sup}\pi_3^{sup} = [0.120441, 0.101910, 0.277862],\ \xi_3^{inf}\pi_3^{inf} = [0.120339, 0.101697, 0.277751].$$

The exact steady state vector of the Courtois matrix in four digits of precision is given by

$$\pi = [0.0893, 0.0928, 0.0405, 0.1585, 0.1189, 0.1204, 0.2778, 0.1018].$$

We must consider the permutations that we performed on each NCD block to obtain the correctly ordered componentwise bounding vectors for π. Hence, we permute the componentwise bounding vectors back to their original ordering and obtain the following componentwise upper and lower bounding vectors on π

$$\pi^{sup} = [0.093203, 0.115510, 0.048476, 0.165611, 0.124297, 0.120441, 0.277862, 0.101910],$$

$$\pi^{inf} = [0.084155, 0.081260, 0.020955, 0.153643, 0.115246, 0.120339, 0.277751, 0.101697].$$

Compare the result of the improved algorithm with those of the following three cases:

(i) No reorderings used :

$$\pi^{sup} = [0.093817, 0.116272, 0.048795, 0.166606, 0.125044, 0.166694, 0.309165, 0.125083]$$

$$\pi^{inf} = [0.083459, 0.080588, 0.020781, 0.152774, 0.114594, 0.100000, 0.208222, 0.090835]$$

(ii) Algorithm 11 used instead of Algorithm 7:

$$\pi^{sup} = [0.093277, 0.115602, 0.049383, 0.165698, 0.124363, 0.120552, 0.277862, 0.101910]$$

$$\pi^{inf} = [0.084094, 0.081201, 0.020149, 0.153538, 0.115168, 0.120228, 0.277751, 0.101697]$$

(iii) Both improvements turned off (i.e., No reorderings used and Algorithm 11 used instead of Algorithm 7):

$$\pi^{sup} = [0.128242, 0.124810, 0.052378, 0.168326, 0.126335, 0.200943, 0.309165, 0.125083]$$

$$\pi^{inf} = [0.059553, 0.079426, 0.020482, 0.143034, 0.107289, 0.065751, 0.208222, 0.090835]$$

After assessing the quality of the bounds, we conclude that the performance of the Algorithm 1 on the Courtois example is extremely good, and it is superior to each of the three cases. However, the Courtois problem is small, and to have

a better understanding of Algorithm 1 we must apply it to larger examples. One of the following chapters is dedicated to such a problem.

The theoretical analysis of Algorithm 1 is given in the next chapter. The former work reported in [24] and in [1] lacks essential theoretical components. We remedy this situation by providing a comprehensive analysis.

Chapter 4

Analysis

In the componentwise bounding algorithm, we extract the submatrix corresponding to the irreducible subset of states from each bounding matrix and solve this subset for its steady state vector. Recall that, the steady state probability distribution of an st-monotone bounding matrix exists iff there exists only one irreducible subset (i.e., one essential subset) in the bounding matrix. Since it is possible to have transient states in each bounding matrix the existence of a single irreducible subset of states must be proved. This discussion, which is very important for the analysis of the algorithm, can not be found in [24]. A similar discussion exists in [1] but lacks important aspects. In this chapter, we give a proof to that effect by stating various definitions, lemmas, and theorems, and show why Algorithm 1 works. Moreover, we prove that our componentwise bounding algorithm that takes in st upper- and lower-bounding probability vectors (see Algorithm 7) is superior to its counterpart in [24]. In [24], the st lower-bounding vector on the steady state distribution of a MC is computed by reversing the order of its states and running Algorithm 5 on the permuted MC. See [24, p. 847] for details. Our new st-monotone lower-bounding matrix construction algorithm (see Algorithm 6) eliminates the need for a permutation vector to order the states of the input stochastic matrix in reverse. The optimality proof of this algorithm is also given in this chapter. Our discussion assumes matrices of order 2 or larger and is based on [18]. First, we introduce two types of stochastic matrices.

DEFINITION 4.1 *A stochastic matrix A of order m that satisfies:*

(i) $\exists j \in \{2, 3, \ldots, m\}$ *such that* $a_{1,j} > 0$,

(ii) $\exists i \in \{1, 2, \ldots, m-1\}$ *such that* $a_{i,m} > 0$,

(iii) $\forall i \in \{1, 2, \ldots, m-1\}$ $\exists k \leq i$ *and* $\exists j > i$ *such that* $a_{k,j} > 0$

is called a type-1 stochastic matrix.

DEFINITION 4.2 *A stochastic matrix A of order m that satisfies:*

(i) $\exists j \in \{1, 2, \ldots, m-1\}$ *such that* $a_{m,j} > 0$,

(ii) $\exists i \in \{2, 3, \ldots, m\}$ *such that* $a_{i,1} > 0$,

(iii) $\forall i \in \{2, 3, \ldots, m\}$ $\exists k \geq i$ *and* $\exists j < i$ *such that* $a_{k,j} > 0$

is called a type-2 stochastic matrix.

LEMMA 4.1 *Let $\overline{S}_i$ be the stochastic matrix computed by Algorithm 2 for the submatrix P_{ii} of order n_i in Algorithm 1. Then $\overline{S}_i$ is a type-1 stochastic matrix.*

LEMMA 4.2 *Let $\underline{S}_i$ be the stochastic matrix computed by Algorithm 3 for the submatrix P_{ii} of order n_i in Algorithm 1. Then $\underline{S}_i$ is a type-2 stochastic matrix.*

PROOF. Let us prove Lemma 4.1. The proof of Lemma 4.2 is similar. The proof consists of showing that parts (i), (ii), and (iii) of Definition 4.1 hold for $\overline{S}_i$. Note that $\overline{S}_i$ (alternatively, $\underline{S}_i$) is P_{ii} with its last (alternatively, first) column perturbed. See Remark 3.1 and consider its implications on Algorithms 2 and 3. For ease of understanding, let us denote P_{ii} by Y, $\overline{S}_i$ by A, and n_i by m.

(i) There are two cases. If $\sum_{j=1}^{m} y_{1,j} = 1$, then $Y_{1,*} = A_{1,*}$ implying $\exists j \in \{2, 3, \ldots, m\}$ such that $a_{1,j} > 0$, otherwise state 1 would be absorbing contradicting the fact that P is irreducible. If $\sum_{j=1}^{m} y_{1,j} < 1$, then by Algorithm 2 we have $\epsilon_1^{(0)} > 0$ and $\delta_{1,m} > 0$ implying $a_{1,m} > 0$. Hence, $\exists j \in \{2, 3, \ldots, m\}$ such that $a_{1,j} > 0$.

(ii) In Y, it is not possible to have $y_{i,m} = 0$ and $\sum_{j=1}^{m} y_{i,j} = 1$ $\forall i \in \{1, 2, \ldots, m-1\}$, otherwise P would be reducible. There are two cases. Suppose for a row $i \in \{1, 2, \ldots, m-1\}$, we have $y_{i,m} > 0$. Then $a_{i,m} > 0$. On the other hand, suppose for a row $i \in \{1, 2, \ldots, m-1\}$, we have $\sum_{j=1}^{m} y_{i,j} < 1$. Then by Algorithm 2, $\epsilon_i^{(0)} > 0$ and $\delta_{i,m} > 0$ implying $a_{i,m} > 0$. Hence, $\exists i \in \{1, 2, \ldots, m-1\}$ such that $a_{i,m} > 0$.

(iii) Let l be the smallest row index among $i \in \{1, 2, \ldots, m-1\}$ for which $a_{i,m} > 0$. From part (ii), there exists such an l. By considering the particular values $k = l$ and $j = m$, for each $i \in \{l, l+1, ..., m-1\}$ $\exists k \leq i$ and $\exists j > i$ such that $a_{k,j} > 0$. Since for each $i \in \{1, 2, ..., l-1\}$, $y_{i,m} = 0$ and $\sum_{j=1}^{m} y_{i,j} = 1$, the irreducibility of P implies that for each $i \in \{1, 2, ..., l-1\}$ $\exists k \leq i$ and $\exists j > i$ such that $a_{k,j} > 0$. □

LEMMA 4.3 *Let $\overline{S}$ be the stochastic matrix computed by Algorithm 2 for the componentwise upper- and lower-bounding coupling matrices U and L of order N in Algorithm 1. Then $\overline{S}$ is a type-1 stochastic matrix.*

LEMMA 4.4 *Let $\underline{S}$ be the stochastic matrix computed by Algorithm 3 for the componentwise upper- and lower-bounding coupling matrices U and L of order N in Algorithm 1. Then $\underline{S}$ is a type-2 stochastic matrix.*

PROOF. Let us prove Lemma 4.3. The proof of Lemma 4.4 is similar. The proof consists of showing that parts (*i*), (*ii*), and (*iii*) of Definition 4.1 hold for $\overline{S}$. Note that, if C is the coupling matrix given by equation (1.3), then $L \leq C \leq U$ by construction (see Algorithms 8 and 9).

(i) Since C is irreducible, there is at least one column $j \in \{2, 3, \ldots, N\}$ such that $c_{1,j} > 0$. Now, there are two cases. When $l_{1,j} = 0$, we have $\epsilon_1^{(0)} > 0$ and $\delta_{1,j} > 0$ (since $L \leq C$) implying $\exists k \geq j$ $\overline{s}_{1,k} > 0$. When $l_{1,j} > 0$, we have $\overline{s}_{1,j} > 0$. Hence, $\exists j \in \{2, 3, \ldots, N\}$ such that $\overline{s}_{1,j} > 0$.

(ii) Since C is irreducible, there is at least one row $i \in \{1, 2, \ldots, N-1\}$ such that $c_{i,N} > 0$. Now, there are two cases. When $l_{i,N} = 0$, we have $\epsilon_i^{(0)} > 0$ and $\delta_{i,N} > 0$ (since $L \leq C$) implying $\overline{s}_{i,N} > 0$. When $l_{i,N} > 0$, we have $\overline{s}_{i,N} > 0$. Hence, $\exists i \in \{1, 2, \ldots, N-1\}$ such that $\overline{s}_{i,N} > 0$.

(iii) Since C is irreducible, for each row $i \in \{1, 2, \ldots, N-1\}$, $\exists k \leq i$ and $\exists j > i$ such that $c_{k,j} > 0$. Again there are two cases. For row i, $l_{k,j} = 0$ implies $\epsilon_k^{(0)} > 0$ and $\delta_{k,j} > 0$ (since $L \leq C$). Then $\exists l \geq j$ $\overline{s}_{k,l} > 0$. For row i, $l_{k,j} > 0$ implies $\overline{s}_{k,j} > 0$. Hence, for each row $i \in \{1, 2, \ldots, N-1\}$, $\exists k \leq i$ and $\exists j > i$ such that $\overline{s}_{k,j} > 0$. □

LEMMA 4.5 *If the input matrix $\overline{S}$ to Algorithm 5 is a type-1 stochastic matrix of order m, then there is a path from each state $i \in \{1, 2, \ldots, m-1\}$ to state m in the output st-monotone upper-bounding matrix $\overline{Q}$.*

LEMMA 4.6 *If the input matrix $\underline{S}$ to Algorithm 6 is a type-2 stochastic matrix of order m, then there is a path from each state $i \in \{2, 3, \ldots, m\}$ to state 1 in the output st-monotone lower-bounding matrix $\underline{Q}$.*

PROOF. Let us prove Lemma 4.5. The proof of Lemma 4.6 is similar. Let l be the state with the smallest index in $\overline{S}$ such that $\overline{s}_{l,m} > 0$. Since $\overline{S}$ is a type-1 stochastic matrix, the existence of such an l is guaranteed by part (ii) of Definition 4.1. From Algorithm 5, $\overline{q}_{l,m} > 0$ as well. From the st-monotonicity of $\overline{Q}$, for each $i \in \{l, l+1, \ldots, m-1\}$ we have $\overline{q}_{i,m} > 0$ implying a path of length one from each state $e \in \{l, l+1, \ldots, m-1\}$ to state m in $\overline{Q}$. What remains to be done is to show that there is a path from each state $e \in \{1, 2, \ldots, l-1\}$ to state m in $\overline{Q}$.

Now, let l_1 be the state with the largest index such that $\overline{s}_{1,l_1} > 0$. The existence of such an l_1 is guaranteed by part (i) of Definition 4.1. From Algorithm

5, $\overline{q}_{1,l_1} > 0$ since the first rows of $\overline{S}$ and $\overline{Q}$ are identical. From the st-monotonicity of $\overline{Q}$, for each $i \in \{2,3,\ldots,m\}$ $\exists j \geq l_1$ it must be that $\overline{q}_{i,j} > 0$. Hence, there are two cases depending on the value of l_1.

When $l_1 \geq l$, for each state $e \in \{2,3,\ldots,l-1\}$ there is a direct transition to a state $j(\geq l_1)$ and a path from state j to state m in $\overline{Q}$. When $l_1 < l$, consider part (iii) of Definition 4.1 and notice that $\exists i \leq l_1$ and $\exists k > l_1$ such that $\overline{s}_{i,k} > 0$. Since $\overline{Q}$ is an st-monotone upper-bounding matrix, this implies $\exists l_2 \geq k > l_1$ such that $\overline{q}_{i,l_2} > 0$. Now, let i_1 be the state with the smallest index among $i(\leq l_1)$ corresponding to the state with the largest index l_2; then $\overline{q}_{i_1,l_2} > 0$. From the st-monotonicity of $\overline{Q}$, for each $i \in \{i_1, i_1+1,\ldots,m\}$ $\exists j \geq l_2$ it must be that $\overline{q}_{i,j} > 0$. Again, there are two cases.

When $e \geq i_1$, there is a direct transition from state e to a state $e'(\geq l_2 > l_1)$ in $\overline{Q}$. When $e < i_1$, there is a direct transition from state e to a state $j(\geq l_1)$. Since $i_1 \leq i \leq l_1$, we have $j \geq i_1$ implying the existence of direct transitions from state e to a state j and from state j to a state e'. Then we must observe the value of l_2. If $l_2 \geq l$, we are at the very first case. If $l_2 < l$, we must continue the recursive analysis as above until l_2 becomes larger than l. Note that $l_2 > l_1$ and l_2 will eventually exceed l. □

THEOREM 4.1 *If the input matrix $\overline{S}$ to Algorithm 5 is a type-1 stochastic matrix of order m, then there is a single irreducible (sub)set of states that includes state m in the given ordering of states in the output st-monotone upper-bounding matrix $\overline{Q}$.*

THEOREM 4.2 *If the input matrix $\underline{S}$ to Algorithm 6 is a type-2 stochastic matrix of order m, then there is a single irreducible (sub)set of states that includes state 1 in the given ordering of states in the output st-monotone lower-bounding matrix $\underline{Q}$.*

PROOF. Let us prove Theorem 4.1. The proof of Theorem 4.2 is similar. Since $\overline{Q}$ is not necessarily irreducible, it may have several classes of states (see [21, p. 26]). Let us denote these classes by C_j and the class which contains state m as

C_l. Now, if C_l is an irreducible class, then from each class C_j, $j \neq l$, there must be a path to C_l. This follows from Lemma 4.5 and the fact that the classes C_j form an exact partition of the state space. Furthermore, if C_l is an irreducible class, it is not possible to leave C_l. Suppose C_l is transient and there is path from C_l to C_j for some $j \neq l$. Then C_l and C_j are equivalent, which contradicts the fact that C_l and C_j are distinct classes. Hence, C_l must be irreducible. Note that C_j cannot be an irreducible class since there is a path to C_l. □

It is possible to have different st-monotone upper and lower-bounding matrices for a stochastic matrix. The problem is to obtain the optimal bounding matrix with respect to the st-order. The optimality of the st-monotone upper-bounding matrix computed by Algorithm 5 is proved in [1, pp. 12–14]. Here we give the proof for the st-monotone lower-bounding matrix computed by Algorithm 6.

THEOREM 4.3 *Let $\underline{Q}$ be the st-monotone lower-bounding matrix computed by Algorithm 6 for the stochastic matrix $\underline{S}$ of order m. Let $\underline{R}$ be another st-monotone lower-bounding matrix for $\underline{S}$. Then $\underline{Q}$ is optimal in the sense that $\underline{R} \leq_{st} \underline{Q}$.*

PROOF. The proof is by induction. By construction, row m in $\underline{Q}$ and $\underline{S}$ are identical (see Algorithm 6). Since $\underline{R}$ is another st-monotone lower-bounding matrix for $\underline{S}$, it must be that $\underline{R}_{m,*} \leq_{st} \underline{Q}_{m,*} = \underline{S}_{m,*}$. This forms the basis step. Now, let us assume that $\underline{R}_{i,*} \leq_{st} \underline{Q}_{i,*}$ for $i \in \{l, l+1, \ldots, m-1\}$. This forms the induction hypothesis. We must show that $\underline{R}_{l-1,*} \leq_{st} \underline{Q}_{l-1,*}$.

Since $\underline{R}$ is st-monotone, $\underline{R}_{l-1,*} \leq_{st} \underline{R}_{l,*}$. Since it is an st-monotone lower-bound for $\underline{S}$, we have $\underline{R}_{l,*} \leq_{st} \underline{S}_{l,*}$. Then from the induction hypothesis, $\underline{R}_{l-1,*} \leq_{st} \underline{R}_{l,*} \leq_{st} \underline{Q}_{l,*}$. Observe that the inequality $\underline{R}_{l-1,*} \leq_{st} \underline{Q}_{l-1,*}$ to be proven is equivalent to

$$\sum_{j=1}^{k} \underline{r}_{l-1,j} \geq \sum_{j=1}^{k} \underline{q}_{l-1,j} \qquad \forall k \in \{1, 2, \ldots, m\}.$$

Now, we analyze Algorithm 6 to see how the elements of $\underline{Q}$ are computed. In the algorithm,

$$\sum_{j=1}^{k} \underline{q}_{l-1,j} = \max(1 - b_{l-1,k+1}, \sum_{j=1}^{k} \underline{q}_{l,j}) \quad \forall k \in \{1, 2, \ldots, m-1\}$$

(when $k = m$, all row sums are 1). We remark that

$$1 - b_{l-1,k+1} = \sum_{j=1}^{k} \underline{s}_{l-1,j}$$

from Algorithm 4; therefore, the $(1 - b_{l-1,k+1})$ argument of max is due to comparison with matrix $\underline{S}$ for a lower-bound. The second argument of max is due to comparison with row l of $\underline{Q}$ for st-monotonicity. Hence, there are two cases for each $k \in \{1, 2, \ldots, m-1\}$. We either have

$$\sum_{j=1}^{k} \underline{q}_{l-1,j} = \sum_{j=1}^{k} \underline{s}_{l-1,j} \quad \text{implying} \quad \sum_{j=1}^{k} \underline{r}_{l-1,j} \geq \sum_{j=1}^{k} \underline{q}_{l-1,j}$$

since $R \leq_{st} S$, or we have

$$\sum_{j=1}^{k} \underline{q}_{l-1,j} = \sum_{j=1}^{k} \underline{q}_{l,j} \quad \text{again implying} \quad \sum_{j=1}^{k} \underline{r}_{l-1,j} \geq \sum_{j=1}^{k} \underline{q}_{l-1,j}$$

since $R_{l-1,*} \leq_{st} Q_{l,*}$. Combining the two cases, we obtain

$$\sum_{j=1}^{k} \underline{r}_{l-1,j} \geq \sum_{j=1}^{k} \underline{q}_{l-1,j} \quad \forall k \in \{1, 2, \ldots, m\}$$

implying $\underline{R}_{l-1,*} \leq_{st} \underline{Q}_{l-1,*}$. □

LEMMA 4.7 *Algorithm 7 computes better componentwise probability bounds than the following algorithm used in [24]:*

ALGORITHM 11. Computation of componentwise probability bounding vectors w^{sup} and w^{inf} as in [24] given st upper- and lower-bounding probability vectors $\overline{v}^{st}$ and $\underline{v}^{st}$ of length m:

$w_m^{sup} = \overline{v}_m^{st}$;
$w_m^{inf} = \underline{v}_m^{st}$;
for $j = m-1, m-2, \ldots, 1$,
  $w_j^{sup} = \min(1, (\sum_{k=j}^{m} \overline{v}_k^{st} - \sum_{k=j+1}^{m} w_k^{inf})^+)$;
  $w_j^{inf} = \min(1, (\sum_{k=j}^{m} \underline{v}_k^{st} - \sum_{k=j+1}^{m} w_k^{sup})^+)$;

PROOF. The proof is by induction. We must show that $v^{sup} \leq w^{sup}$ and $v^{inf} \geq w^{inf}$. By construction (see Algorithms 7 and 10), $v_m^{sup} = w_m^{sup} = \overline{v}_m^{st}$ and $v_m^{inf} = w_m^{inf} = \underline{v}_m^{st}$. This is the basis step. Now, let us assume that $v_j^{sup} \leq w_j^{sup}$ and $v_j^{inf} \geq w_j^{inf}$ for $j \in \{l, l+1, \ldots, m-1\}$. This forms the induction hypothesis. We must show that $v_{l-1}^{sup} \leq w_{l-1}^{sup}$ and $v_{l-1}^{inf} \geq w_{l-1}^{inf}$.

From Algorithms 7 and 10, we respectively have

$$v_{l-1}^{sup} = v_l^{sup} + \overline{v}_{l-1}^{st} - \underline{v}_l^{st} \quad \text{and} \quad w_{l-1}^{sup} = w_l^{sup} + \overline{v}_{l-1}^{st} - w_l^{inf}.$$

From the induction hypothesis, we have $v_l^{sup} \leq w_l^{sup}$ and $v_l^{inf} \geq w_l^{inf}$. Observing that $\underline{v}_l^{st} \geq v_l^{inf}$ from Algorithm 7, we obtain $v_{l-1}^{sup} \leq w_{l-1}^{sup}$.

Similarly, from Algorithms 7 and 10, we respectively have

$$v_{l-1}^{inf} = v_l^{inf} + \underline{v}_{l-1}^{st} - \overline{v}_l^{st} \quad \text{and} \quad w_{l-1}^{inf} = w_l^{inf} + \underline{v}_{l-1}^{st} - w_l^{sup}.$$

Observing that $\overline{v}_l^{st} \leq v_l^{sup}$ and using the induction hypothesis, we obtain $v_{l-1}^{inf} \geq w_{l-1}^{inf}$. □

Chapter 5

Implementation Details

The pioneering work about finding componentwise bounds for the steady state vector of NCD MCs in [24] and [1] does not include any applications. Such applications generally have thousands of states and possess some sparsity pattern. To asses the validity of our work, we apply the proposed algorithm to a current application.

Working on applications requires the implementation of Algorithm 1 efficiently in terms of time and space. Obviously, Algorithm 1 is complicated to implement. Our implementation is focused on obtaining results in acceptable time limits using memory available on a workstation. To achieve this, we use different types of data structures in implementing the algorithm.

5.1 Compact Sparse Row Format

There are matrices generated from Markov models that are too large or too dense to permit regular two-dimensional storage in computer memory. In addition to this, these matrices are sparse (i.e., a large percentage of their elements are zero). For these reasons, we store MCs in a different type of data structure which leads to considerable storage and computational savings at run-time. Our data structure

is a kind of compaction scheme whereby only the nonzero elements and their positions in the matrix are stored and is called the *Compact Sparse Row* (CSR) format. It requires for each coefficient matrix of order m one real and one integer array of size nz (i.e., number of nonzero elements in the coefficient matrix), and one integer array of size $(m+1)$. This scheme not only has the advantage of using less space but is also very useful in computation. Whenever we see the advantage of exploiting the zeros in any step of Algorithm 1, we use CSR format. The following is an example stored in CSR format:

$$A = \begin{pmatrix} -2.1 & 0.8 & 0.2 & 0.0 \\ 0.0 & -0.8 & 1.5 & 0.3 \\ 1.7 & 0.0 & -1.7 & 0.2 \\ 0.4 & 0.0 & 0.0 & -0.5 \end{pmatrix}$$

	1	2	3	4	5	6	7	8	9	10	11
aa:	-2.1	0.8	0.2	-0.8	1.5	0.3	1.7	-1.7	0.2	0.4	-0.5
ja:	1	2	3	2	3	4	1	3	4	1	4
ia:	1	4	7	10	12						

Here aa is the real and ja is the integer array of size nz and ia is the integer array of size $(m+1)$. Most of the steps of Algorithm 1 are implemented in CSR format.

5.2 The Details of Algorithm 1

The input matrix P of order n which has nz nonzero elements is kept in CSR format in our implementation. Assume that the nonzero elements of P are distributed uniformly across the matrix. Note that nz is considered to be $O(n)$ for sparse matrices. Hence, there will be roughly $k = nz/n$ nonzero elements per row/column of P and $k_i = nz \times n_i/n^2$ nonzero elements per row/column of P_{ii}. The uniform assumption is neither an optimistic nor a pessimistic one.

In [6], an efficient way of implementing the NCD partitioning algorithm in CSR format is given. We implement the algorithm in [6] in CSR format, which

requires extra $O(n+nz)$ integer and $O(nz)$ real space. We name these extra spaces as integer and real work arrays, allocate them at the beginning of the program, and use them in any step of Algorithm 1 when we need temporary space. In other words, we do not allocate extra space when needed. For the time complexity, we account for floating-point comparisons and floating-point arithmetic operations. Hence, the time complexity of the NCD partitioning algorithm is $O(n + nz)$ floating-point comparisons [6].

In Step 0 we symmetrically permute the matrix P to put it in the form of equation (1.2). The cost of permutation in CSR format is negligible since it does not involve any comparisons or arithmetic operations and is faster than two dimensional (2D) implementations because we do not deal with permuting the zero entries. On the other hand, the permutation operation is not in place. Therefore, we need to store the permuted matrix in temporary space. For this reason, we use nz real and $(nz + n + 1)$ integer temporary space to store the permuted matrix in CSR format.

5.2.1 The Orderings of NCD blocks

Recall that, in order to improve the quality of the bounds, we find an ordering for each of the NCD blocks using Algorithm 10 in Chapter 3 after determining the last state in each block. Considering the uniform distribution assumption regarding the nonzeros, the last state that has the largest self-transition probability is determined after nearly $n_i \times k_i/2 + n_i$ floating-point comparisons for block i in our CSR implementation if tie-breaking rules are not used. We store the selected state as the last entry of a permutation vector whose other elements will be the remaining states of the particular NCD block. This permutation vector is given as input to the implementation of Algorithm 10. In this way, symmetric permutation of the selected state to be the last in the NCD block becomes unnecessary.

Algorithm 10 is a complicated algorithm. Our implementation is done in CSR format and runs in a reasonable time as we will see later. The underlying

implementation of Algorithm 10 is quite different than the corresponding pseudo-code given in Chapter 3. There is a lot of repetition in the code. For example, there is no need to generate the set $\mathcal{I}_t$ from scratch in each step if it contains more than one element in any step of the algorithm. Removing one element from $\mathcal{I}_t$ in the next step is sufficient after generating it in the current step (see line 8).

We keep $\mathcal{I}$, $\mathcal{I}_t$, and $\mathcal{I}_{tt}$ as singly linked lists. The elements of $\mathcal{I}$ are kept sorted with respect to the field p_{im} (m is the order of the NCD block) with insertion sort before starting the outer while loop. The worst case cost of this sort is $O(m^2)$ floating-point comparisons. The elements which have the largest transition probabilities to the last state are placed at the head of $\mathcal{I}$. In this way, when generating $\mathcal{I}_t$, we just look at the head of the list. If $\mathcal{I}_t$ has more than one element, it is not generated in the following steps until all the elements of $\mathcal{I}_t$ are placed in the permutation vector. This kind of implementation minimizes the cost of generating the set $\mathcal{I}_t$.

In the case of more than one element in $\mathcal{I}_t$, we have two tie-breaking rules to select a state from $\mathcal{I}_t$. If the tie is not broken in the first rule, it is broken in the second rule. We implement the second tie-breaking rule as it appears in the algorithm, but consider a different implementation for the first rule to improve time. Before entering the first rule, we map the states of $\mathcal{I}_t$ to an ordinary array. This is done at the time of creation of $\mathcal{I}_t$ and this array is kept until the next creation of $\mathcal{I}_t$. Assume that the condition $(m - j > is)$ is satisfied and we enter the inner while loop. We perform a quick sort in the array with respect to the field $p_{i,index_{m-1}}$. Now, it is possible to have a sorted array as $\{S_1, S_2, S_3\}$, where S_i represents the subset of states that have the same value in the sort field and the set having the largest value is at the beginning of this array. In the best case, the set S_1 has just one element, the tie is broken, and the algorithm moves to the next step. However, we do not move to the next step in our implementation if the condition $(m - j > is)$ permits us to stay in the innermost while loop and at least one of the remaining subsets has more than one element. We do not leave because the algorithm enters the first rule to resolve the tie in the subsets having more than one element. Our implementation stays and performs another quick sort due to the field $p_{i,index_{m-2}}$ (i.e., next field) in the other subsets. We break from

the while loop if ($m - j \leq is$) or all the subsets have a single element. At that point, it is possible to have a sorted array as $\{S_1, S_{2_1}, S_{2_2}, S_{3_1}, S_{3_2}, S_{3_3}\}$, where all S_i represent subsets having one or more states. If one of these subsets have more than one state, in the next entrance to the first rule we order the remaining states starting from the field where we left in the previous sort. We never start from the beginning as in the pseudo-code. These improvements introduce substantial savings to the cost of Algorithm 10. The space complexity of implementing this algorithm is $O(m)$ integers. There are at most $O(m^2)$ floating-point comparisons in our implementation. After determining the ordering for each of the NCD blocks, we symmetrically permute each block according to the corresponding ordering.

5.2.2 Bounding Matrices

In Steps 1.b and 1.c we generate the st-monotone bounding matrices for each NCD block, and in Steps 2.b and 2.c we obtain the bounding matrices for the exact coupling matrix.

We implement Algorithms 2 and 3 in a slightly different manner when generating the stochastic matrices for each NCD block P_{ii}. When we apply Algorithm 2 (alternatively, Algorithm 3) to P_{ii} to generate $\overline{S}_i$ (alternatively, $\underline{S}_i$), we see that $\overline{S}_i$ is P_{ii} with its last column perturbed and similarly $\underline{S}_i$ is P_{ii} with first column perturbed. However this observation does not hold when we use the same algorithms in Step 2.b. Perturbations may occur in any column of L and U to generate the stochastic matrices $\overline{S}$ and $\underline{S}$. Hence, in our implementation the innermost loops of Algorithms 2 and 3 execute exactly once for each block in Step 1.b. For Step 2.b, these algorithms are implemented as they appear. The time complexity of Step 1.b is roughly $O(n_i \times k_i + n_i)$ floating-point arithmetic operations and $O(n_i)$ floating-point comparisons for block P_{ii}. On the other hand, Step 2.b requires $O(N^2)$ floating-point comparisons and arithmetic operations.

The ordered NCD blocks, which are input to Step 1.b, are kept in CSR format. This format gives us the advantage of not dealing with zero entries in finding the

row sums. However, we do not keep the output of Algorithms 2 and 3 in CSR format. It is possible to extract the zero entries with additional floating point comparisons, but this approach ends up being inefficient when we consider the details of Algorithms 2,3,5, and 6.

Algorithms 2 and 3 may change the nonzero structure of the matrices they have as input. When we employ these algorithms to NCD blocks in Step 1.b of Algorithm 1, the resulting stochastic matrices may have additional nonzero entries in the last column or in the first column. Since we are keeping the blocks in CSR format, adding nonzero entries to the matrix in this format requires an expansion and a compaction. It is clear that if a compaction is done in Step 1.b, an expansion must follow in the next step. These are all unwanted costs. In addition, the bounding matrix algorithms (i.e., Algorithms 5 and 6) may change the nonzero structure of these stochastic matrices altogether. They may insert a nonzero to or remove a nonzero from any location of the stochastic matrices they have as input. As a result, the number of nonzeros in the st-monotone bounding matrices cannot be known. If the number of nonzeros cannot be anticipated, storage advantages of the CSR format diminish. Therefore, we employ a 2D implementation and storage scheme in Algorithms 2,3,5, and 6 for NCD blocks.

We generate L and U in Step 2a row by row and construct the rows of $\overline{S}$ and $\underline{S}$ simultaneously. A compaction is not performed, because the number and location of nonzeros in the bounding matrices which will be generated by Algorithms 5 and 6 are completely unknown. For this reason, the stochastic matrices are kept in 2D and the implementation of Step 2.c is also done in this format.

The 2D implementation and storage scheme requires at least two $(\max n_i)^2$ real temporary space for Steps 1.b and 1.c. One of them is used to generate $\overline{S}_i$ and $\overline{Q}_i$; the other one is used for $\underline{S}_i$ and $\underline{Q}_i$. Similarly, the space complexity of Steps 2.b and 2.c is $(2 \times N^2)$ reals. The time complexity of st-monotone bounding matrix construction algorithms is $O(m^2)$ floating-point comparisons and arithmetic operations for a matrix of order m. Floating-point operations needed to implement Algorithms 5 and 6 may seem higher than this value at first, but we have an efficient implementation. We keep a temporary vector of

size m that holds the sum of newly computed entries in each row. In this way, the summation operations in the innermost loops become negligible.

5.2.3 Extracting the Essential Class

Remember that the st-monotone bounding matrices may be reducible. However in Chapter 4, we proved the existence of only one essential class in these stochastic matrices. In order to achieve good run times in our implementation, extracting the essential block from all bounding matrices must be negligible when compared with other steps.

Let $\overline{Q}$ be an st-monotone upper bounding stochastic matrix in Algorithm 1. Obtaining the essential block in $\overline{Q}$ is equivalent to making a modified strongly connected component (SCC) search in the directed graph G represented by $\overline{Q}$. The SCC search in G generates the component graph G^{SCC} whose nodes represent the SCC components of G, and the edges are the transitions between these components. The essential class of $\overline{Q}$ can be obtained from G^{SCC}. It is the node of G^{SCC} which has only incoming arcs since it is impossible to leave such a node.

In the section about direct methods for solving homogeneous linear systems, we mentioned the advantages of solving the transposed version of these systems. Therefore, we use the transposed version of the bounding matrices. In order to avoid making extra transpositions of $\overline{Q}$, we run the SCC search algorithm on $\overline{Q}^T$. Let G^T be the graph of $\overline{Q}^T$ and $G^{T^{SCC}}$ be its corresponding component graph. The SCC components of G and G^T are the same, but $G^{T^{SCC}}$ is the transpose of G^{SCC}. Consequently, the essential class of $\overline{Q}$ is the SCC component of G^T which has only outgoing arcs. Figures 5.1 and 5.2 summarize these results. Let the component graph of G take the form in Figure 5.1. It can be seen that SCC_1, SCC_2, and SCC_3 represent the transient blocks of Q. SCC_4 is the only essential block of $\overline{Q}$.

The essential class in $\overline{Q}^T$ can be obtained by making another search in $G^{T^{SCC}}$ after generating it. However this is an inefficient solution. We make a small

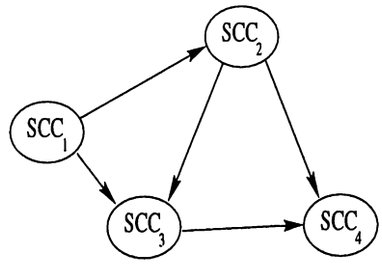


Figure 5.1: Component graph of G

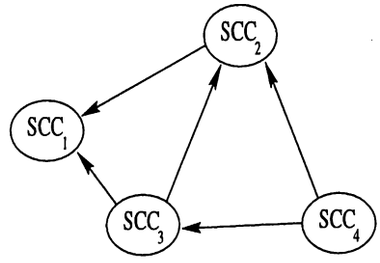


Figure 5.2: Component graph of G^T

modification in the SCC search algorithm given in [2, p. 196] to detect the essential class during the SCC search in $\overline{Q}^T$. Our modification introduces negligible change in the time complexity and space requirement of the algorithm in [2].

The SCC search algorithm in [2] keeps track of vertices to be placed in SCCs by pushing each vertex and its adjacent vertices into a stack. When a SCC component is detected, the vertices in that component are in the top positions of the current stack. They are popped, and therefore do not enter the stack again. To obtain the essential class, we follow the same implementation in [2], but apply it to $\overline{Q}^T$ by adding some checkpoints to detect the essential class. During the search of adjacent vertices of a vertex in $\overline{Q}^T$, if we encounter a vertex which has been discovered and placed in an SCC, we call its corresponding SCC block as transient. It means that, this SCC has an incoming arc from another SCC which has not been discovered yet in $G^{T^{SCC}}$. In addition to this, there is one more criteria to detect the essential class. The algorithm in [2] starts popping the vertices of an SCC from the stack when it discovers a new component. At the end of these popping operations, the stack may be nonempty. If the stack is not empty, this means that the newly discovered SCC has an incoming arc in

$G^{T^{SCC}}$, and this arc belongs to the vertex which still exists in the stack after the pop operations for the new SCC. Consequently this SCC is a transient class of $\overline{Q}$.

Using these two checks in the SCC algorithm, we detect the SCC which has only outgoing arcs in $\overline{Q}^T$ (i.e., essential class) at the end of the search. The implementation of this search is done in CSR format. We have a routine to transform the 2D bounding matrices to CSR format in transposed version. We call this routine before making the search. The CSR format introduces substantial savings to the run time of our implementation. The complexity of the search is $O(nz_m)$ floating-point comparisons and space used by the recursion stack is $O(m)$ integers for a matrix of order m having nz_m nonzero entries [2, p. 196].

5.2.4 Steady State Vectors

After extracting the essential class from the transposed version of the bounding matrix, we solve the transposed system for its steady state vector. A direct method is employed in solving the linear system. Due to the nature of the problem, the bounding matrices may be ill-conditioned. We prefer to use the GTH method for ill-conditioned problems and solve the others using GE.

Let B be the transpose of the submatrix corresponding to the essential class in the bounding matrix and assume that it is of order m. We seek the LU factorization of B. Only U needs to be kept; multipliers are discarded immediately after they are used. We keep B in CSR format in our implementation.

In the implementation of GE, keeping B in CSR format has several advantages over traditional implementations in two dimensional storage. These advantages originate from the sparsity pattern of B. In our implementation of GE, we expand the row to be processed and eliminate it using the rows above. No further updates are made on the current row. Speaking more formally, when row i of B is considered, rows 1 through $(i-1)$ have been reduced to upper triangular form. The first $(i-1)$ rows may then be used to eliminate all nonzero elements in row i

between column positions 1 through $(i-1)$. Once row i is treated in this manner, no more fill-in occurs. After this, row i may be compacted into CSR format and appended to the rows that have already been reduced. Here, the U factor must be kept in an another storage area, because the nonzero pattern of U can not be forecasted at the beginning. We store U in our real and integer temporary work arrays.

The implementation of GTH is quite different from GE. Difficulties arise in implementing GTH because the LU factorization of B is sought and CSR storage scheme is used. In the CSR format, we have easy access to the rows of B. However, since we are solving the transposed system we also need easy access to the columns of B to obtain the pivots by the GTH way. The CSR storage scheme does not provide convenient access to columns of B. To alleviate this problem, we use a different elimination procedure in GTH. We update all rows with indices larger than i in the ith step of elimination. Moreover, the running sum of elements that contribute to the pivot of the $(i+1)$st step is computed. In other words, the negated sum of the elements below row $(i+1)$ for the $(i+1)$st column is accumulated and taken as the pivot element in the next elimination step. This elimination procedure requires the expansion and recompaction of the unreduced submatrix continuously throughout the algorithm. Hence, the CSR implementation of GTH is inefficient. However, we need GTH for ill-conditioned problems that are sparse. To solve this dilemma, we consider a different compaction scheme for GTH. It is based on generating linked lists for each row of B to hold its nonzeros.

During elimination, it is possible for the number of nonzeros in a row to increase. As a result of this, the number of nodes necessary to store the row in the linked list may be insufficient. To solve this problem efficiently, for each row we generate a number of nodes equal to a multiple of the number of nonzeros in that row before the elimination. Obviously, it is possible to have fill-in that exceeds the preallocated number of nonzeros. Only in that case, we generate additional nodes in the linked list for that row. This implementation of GTH seems inefficient at first, but its comparison with the available CSR implementation of GTH reveals that the linked list implementation for sparse matrices works substantially faster.

5.2.5 Ordering for Small Bandwidth

In our experimental observations, we see that the bounding matrices are mostly sparse. To expedite the solution process of direct solvers in Algorithm 1, we consider an ordering approach for sparse matrices. The objective of the ordering is to minimize the number of operations in the elimination by permuting the matrix to a form which has narrower bandwidth. The ordering of interest is found by the *Reverse Cuthill – McKee* (*RCM*) algorithm given in [10, p. 153]. When the matrix is symmetrically permuted according to this ordering, it is transformed to a form which generally has smaller bandwidth. For large bounding matrices, we employ the RCM algorithm and permute the matrices according to the resulting orderings before solving them with direct solvers. The execution times of the RCM algorithm and the permutation are negligible in CSR format, and the gain with this ordering is worth its run time.

In conclusion, the space complexity of Algorithm 1 other than the storage set aside for P is $\max\{O(nz), \max_i\{O(n_i^2), O(N^2)\}\}$ reals and integers from Steps 0, 1.b, 1.c, 2.b, and 2.c. Other steps contribute as lower order terms. As for the time complexity of the algorithm, we should account for floating-point comparisons and floating-point arithmetic operations separately. From Steps 0, 1.a-d, 2.a, and 2.d, we have $\max\{O(nk^2), \sum_{i=1}^{N} \max\{O(n_i^2), O(n_i k_i^2)\}, O(Nn), O(N^3)\}$ floating-point comparisons. From Steps 1.d, 2.a, and 2.d, we have $\max\{\sum_{i=1}^{N} O(n_i^3), O(nz), O(Nn), O(N^3)\}$ floating-point arithmetic operations. Other steps contribute as lower order terms. Now it is evident why one should opt for balanced NCD partitionings (cf. Step 0).

Chapter 6

An Application

To assess the validity of our work, we performed various experiments on a current application. The application that we consider arises in wireless asynchronous transfer mode (ATM) networks and possesses NCD structure.

6.1 Wireless ATM model

In [25], a multiservices resource allocation policy (MRAP) is developed to integrate two types of service over time division multiple access (TDMA) frames in a mobile communication environment established on a wireless ATM network. 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. A single cell and a single carrier frequency is modeled.

The TDMA frame is assumed to have C slots. Each mobile user that has a call in progress generates a handover request as s/he moves from one cell to another. To allow the call in progress to continue in the newly entered cell, the handover requests should be served immediately. Hence, handover requests have priority over new call arrivals, and they respectively arrive with probabilities p_h

and p_n. Moreover, each voice call takes up a single slot of a single TDMA frame but may span multiple TDMA frames whereas each data packet is served in a single slot. When all the slots are full, incoming voice calls are rejected. The number of voice calls that may terminate in a given TDMA frame depends on the number of active calls and is modeled as a binomial process with parameter p_s. In this way, it is possible to have multiple departures of voice calls during a TDMA frame. On the other hand, data is 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 $L = \beta/(\alpha+\beta)$. 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 L and S_C, $\beta = 2L(1-L)/(S_C+1-L)$ and $\alpha = \beta(1-L)/L$. 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}. The mean arrival rate of data packets in the on state is defined as $R = \sum_{i=1}^{3} i \times p_{di}$. Hence, the global mean arrival rate of data packets is given by $G = L \times R$. When the buffer is full, any excess packet is dropped. In this model, we do not consider the arrival of multiple handovers or new calls during a TDMA frame since the associated probabilities with these events are small. The arrival process of data and the service process of calls we consider is quite general and subsumes the model in [25].

The parameters of the model are $p_h = C \times 10^{-5}$, $p_n = C \times 5 \times 10^{-6}$, and $p_s = C \times 5 \times 10^{-6}$. The performance measures of interest are the blocking probability of voice calls and the dropping probability of data packets. We obtain these performance measures by generating the underlying MC by a three-component state descriptor (a, b, c), where a denotes the state of the data arrival process, b denotes the number of data packets in the buffer and c denotes the number of active voice calls. State changes happen at frame boundaries and transition probabilities are computed using the priority rules among handover requests, new call arrivals and data packet arrivals. Using the steady state probabilities, the

blocking probability of voice calls is calculated as

$$\begin{aligned} p_{block} &= [(p_n(1-p_h)+(1-p_n)p_h+2p_np_h)(1-p_s)^C \sum_{i=0}^{1}\sum_{j=0}^{B}\pi_{i,j,C} \\ &\quad +p_np_hC(1-p_s)^{C-1}p_s\sum_{i=0}^{1}\sum_{j=0}^{B}\pi_{i,j,C} \\ &\quad +p_np_h(1-p_s)^{C-1}\sum_{i=0}^{1}\sum_{j=0}^{B}\pi_{i,j,C-1}]/[p_n(1-p_h)+(1-p_n)p_h+2p_np_h] \end{aligned}$$

and the dropping probability of data packets can be obtained from

$$\begin{aligned} p_{drop} &= [(p_{d1}+2p_{d2}+3p_{d3})\sum_{i=0}^{C}\pi_{1,B,i}+(p_{d2}+2p_{d3})\sum_{i=0}^{C}\pi_{1,B-1,i} \\ &\quad +p_{d3}\sum_{i=0}^{C}\pi_{1,B-2,i}]/[p_{d1}+2p_{d2}+3p_{d3}]. \end{aligned}$$

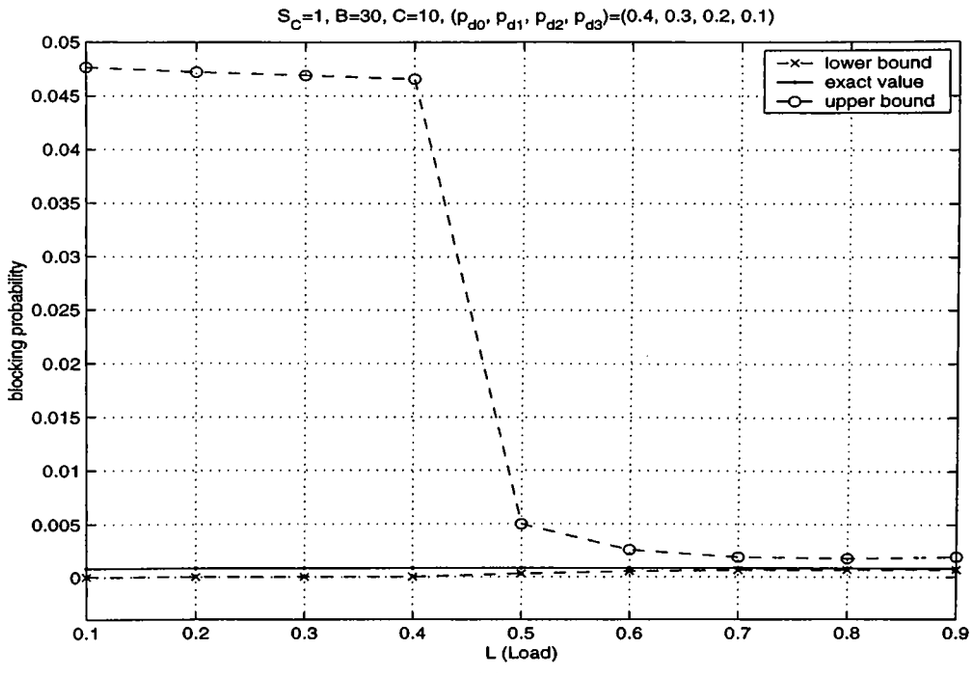
We remark that the above formulae is defined on the product state space having $2(B+1)(C+1)$ states of which some are unreachable. In other words, the states which do not exist in the model are never generated.

6.2 Numerical Results

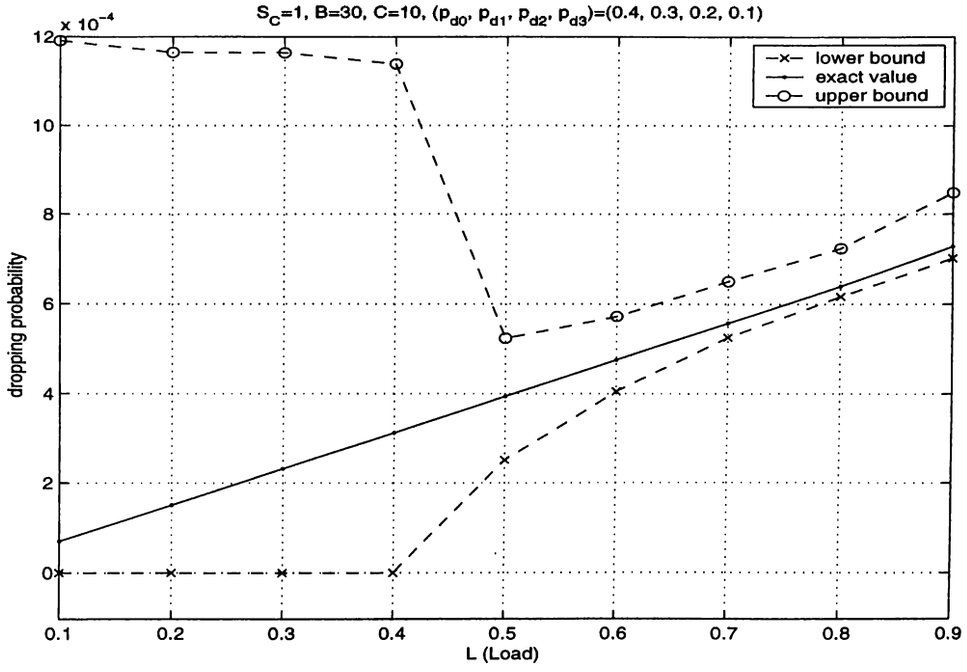
We executed Algorithm 1 on the wireless ATM model. Since the resulting NCD MCs are of moderate order (i.e., thousands of states) and sparsity (i.e., tens of nonzeros per row), we consider the direct solution method of GTH at each level of Algorithm 1. We do not consider the Cuthill-Mckee ordering approach because its effect is negligible for such kind of small systems. All code is written in Fortran/C and compiled in double precision with *g77/gcc* on a SUN UltraSparcstation 10 with 128 MBytes of RAM running Solaris 2.6. The numerical experiments are timed using a C function that reports CPU time. We compare the run-time of Algorithm 1 with that of GTH and iterative aggregation-disaggregation (IAD) [21] which are both geared towards NCD MCs. In order to make a fair comparison, with IAD we use the same partitionings as in Algorithm 1. For all combinations of the integer parameters we considered, there is sufficient space to factorize in sparse format (that is, to apply sparse GE to) the diagonal blocks in IAD.

Furthermore, we use block Gauss-Seidel (BGS) in the disaggregation step and employ a stopping tolerance of 10^{-15} on the infinity norm of the residual vector at each iteration. We remark that for each problem solved, the relative backward error in IAD turns out to be less than 10^{-16}. See [8] for recent results on the computation of the stationary vector of Markov chains. The bounds obtained on performance measures are given in Figures 6.1, 6.2, 6.3, 6.4, 6.5, and 6.6.

In Figures 6.1, 6.2, and 6.3 using Algorithm 1 we present bounds on the blocking probability of voice calls and the dropping probability of data packets in the system with $B = 30$, $C = 10$. We set $(p_{d0}, p_{d1}, p_{d2}, p_{d3}) = (0.4, 0.3, 0.2, 0.1)$ implying $R = 1.0$, take $L \in \{0.1, 0.2, \ldots, 0.9\}$ and $S_C \in \{1, 10, 100\}$. Observe that there is orders of magnitude between the average interarrival time of voice calls and the average interarrival time of data packets, which makes this problem NCD. In fact, the smallest degree of coupling values we computed for this problem and the larger version next are on the order of 10^{-4}. The NCD partitionings considered for $S_C = 1$ in Figures 6.1.(a)-(b) all have 11 blocks with orders between 42 and 62, and a degree of coupling 6×10^{-4}. The NCD partitionings considered for $S_C = 10$ in Figures 6.2.(a)-(b) all have 22 blocks with orders between 21 and 31, and degree of coupling values between 1×10^{-1} (for $L = 0.1$) and 2×10^{-2} (for $L = 0.9$). The NCD partitionings considered for $S_C = 100$ in Figures 6.3.(a)-(b) all have 22 blocks with orders between 21 and 31, and degree of coupling values between 2×10^{-2} (for $L = 0.1$) and 2×10^{-3} (for $L = 0.9$). The underlying MC that has 572 states and 20,198 nonzero elements takes 0.3 seconds to solve when $S_C = 1$ and 0.2 seconds to solve when $S_C \in \{10, 100\}$ using Algorithm 1. Steps 0 and 1.a take a total of about 0 seconds. It takes 2.6 seconds to solve the same MC by GTH for each L. It takes at least 1.5 seconds (5 iterations) to solve when $S_C = 1$ and at least 1.8 seconds (9 iterations) to solve when $S_C \in \{10, 100\}$ using IAD.



(a)



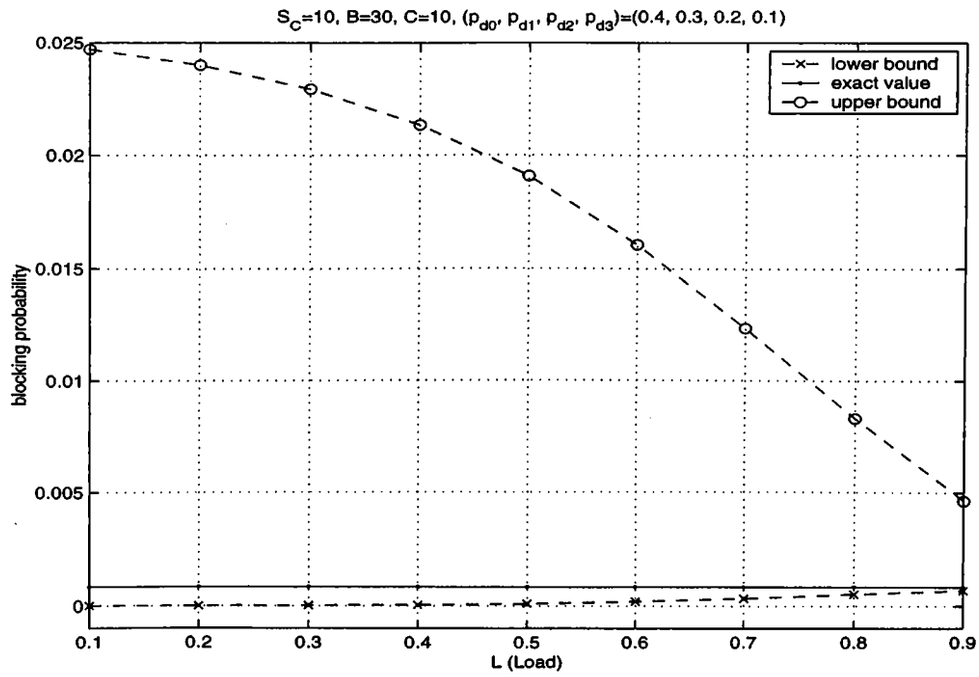
(b)

Figure 6.1: Blocking and dropping probabilities for $S_C = 1$ when $B = 30$ and $C = 10$.

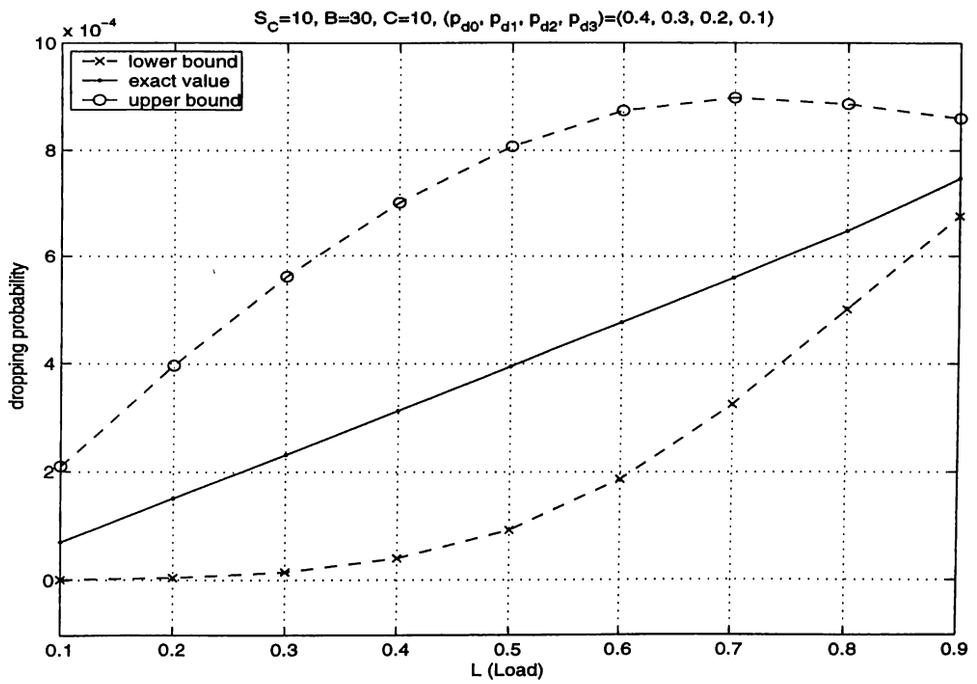
In Figures 6.4, 6.5, and 6.6 using Algorithm 1 we present bounds on the blocking probability of voice calls and the dropping probability of data packets in the system with $B = 60$, $C = 30$. We set $(p_{d0}, p_{d1}, p_{d2}, p_{d3}) = (0.4, 0.3, 0.2, 0.1)$ implying $R = 1.0$, take $L \in \{0.1, 0.2, \ldots, 0.9\}$ and $S_C \in \{1, 10, 100\}$. The NCD partitionings considered for $S_C = 1$ in Figures 6.4.(a)-(b) all have 31 blocks with orders between 62 and 122, and a degree of coupling 5×10^{-3}. The NCD partitionings considered for $S_C = 10$ in Figures 6.5.(a)-(b) all have 62 blocks with orders between 31 and 61, and degree of coupling values between 2×10^{-1} (for $L = 0.1$) and 2×10^{-2} (for $L = 0.9$). The NCD partitioning considered for $S_C = 100$ in Figures 6.6.(a)-(b) all have 62 blocks with orders between 31 and 61, and degree of coupling values between 2×10^{-2} (for $L = 0.1$) and 7×10^{-3} (for $L = 0.9$). The underlying MC that has 2,852 states and 217,778 nonzero elements takes 3.3 (Step 0: 0.3 seconds; Step 1.a: 0.3 seconds) to solve when $S_C = 1$ and 2.5 seconds (Step 0: 0.3 seconds; Step 1.a: 0.2 seconds) to solve when $S_C \in \{10, 100\}$ using Algorithm 1. It takes 260.0 seconds to solve the same MC by GTH for each L. It takes at least 64.2 seconds (3 iterations) to solve when $S_C = 1$ and at least 75.4 seconds (4 iterations) to solve when $S_C \in \{10, 100\}$ using IAD.

Since voice calls have priority in service, their blocking probability is not affected by L and S_C (see Figures 6.1-6.6 part (a)) whereas the dropping probability of data packets increases with L and S_C though the increase with S_C happens very slowly (see Figures 6.1-6.6 part (b)). Both probabilities decrease when we move from Figures 6.1, 6.2, and 6.3 to Figure 6.4, 6.5, and 6.6. A bigger C implies a smaller blocking probability for voice calls, bigger B and C imply a smaller dropping probability for data packets.

The time spent to compute bounds using Algorithm 1 is very promising compared to solving the NCD MCs using GTH or IAD. This is understandable since Algorithm 1 solves multiple smaller systems (i.e., two systems corresponding to each NCD block i with order at most n_i) and two aggregated systems of order at most N whereas GTH solves the global system of order n and IAD performs a number of aggregation-disaggregation iterations. In addition, the memory required for running our bounding algorithm on these problems is moderate and

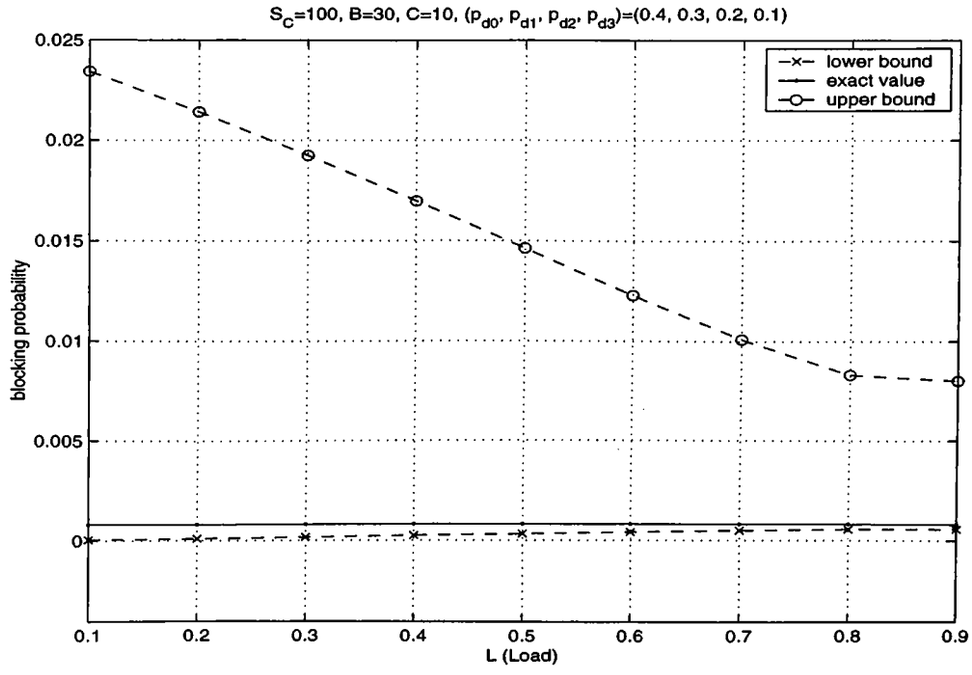


(a)

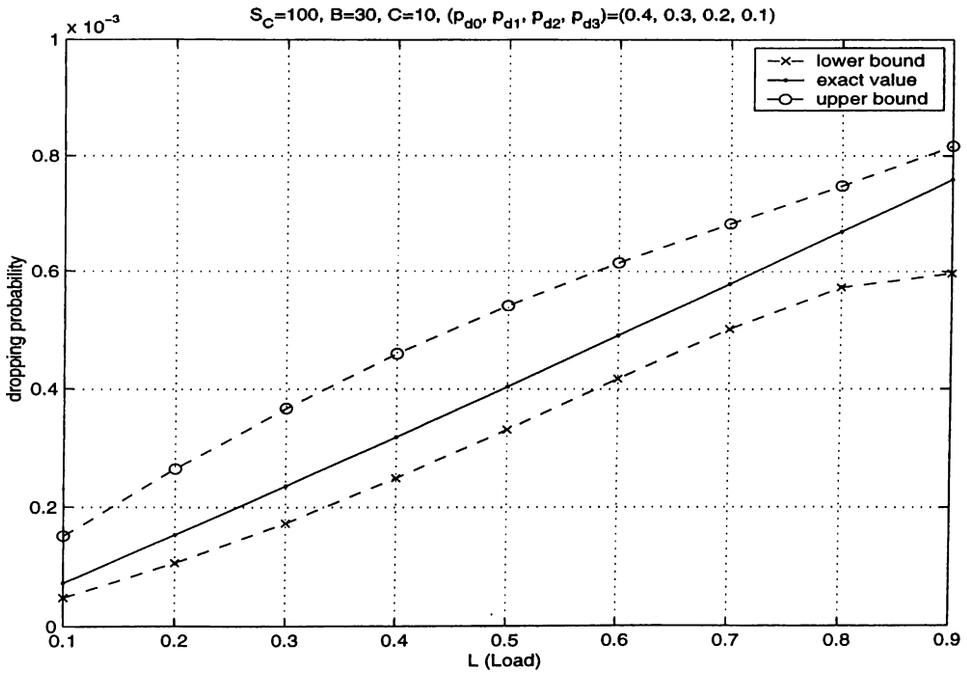


(b)

Figure 6.2: Blocking and dropping probabilities for $S_C = 10$ when $B = 30$ and $C = 10$.

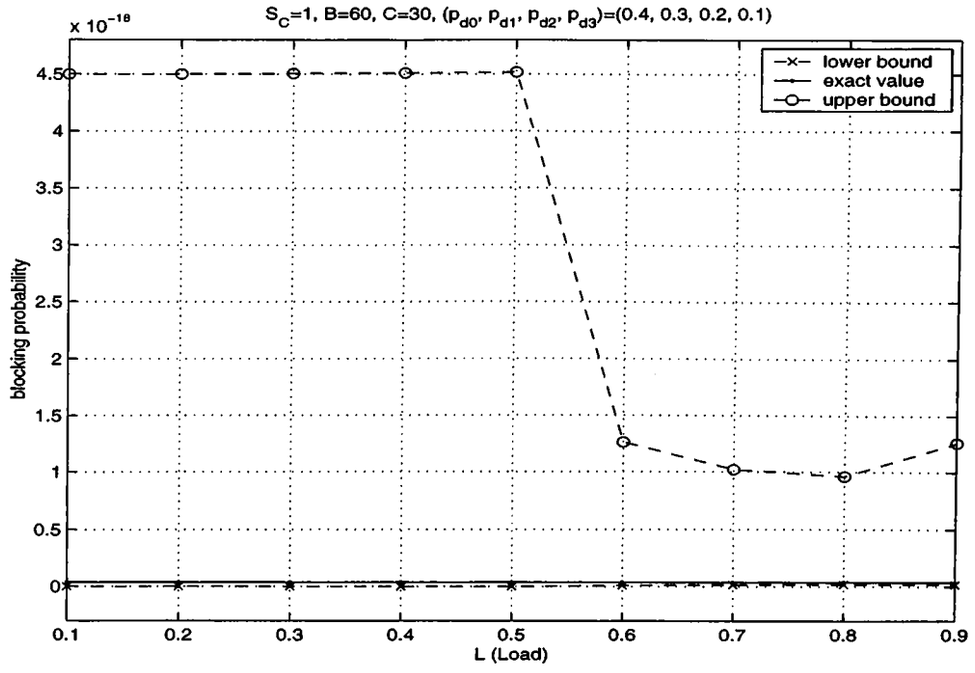


(a)

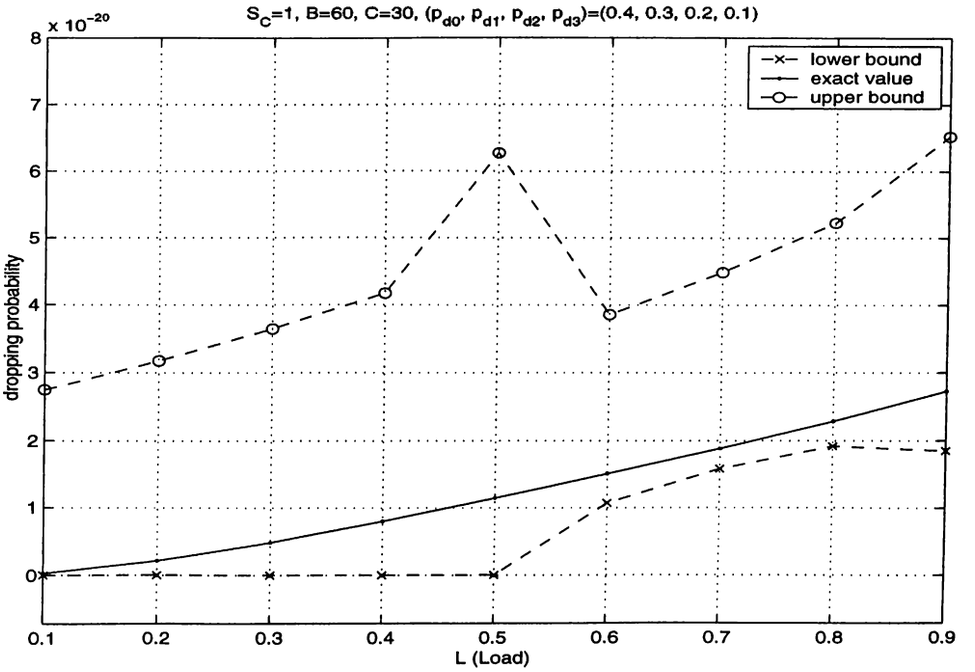


(b)

Figure 6.3: Blocking and dropping probabilities for $S_C = 100$ when $B = 30$ and $C = 10$.

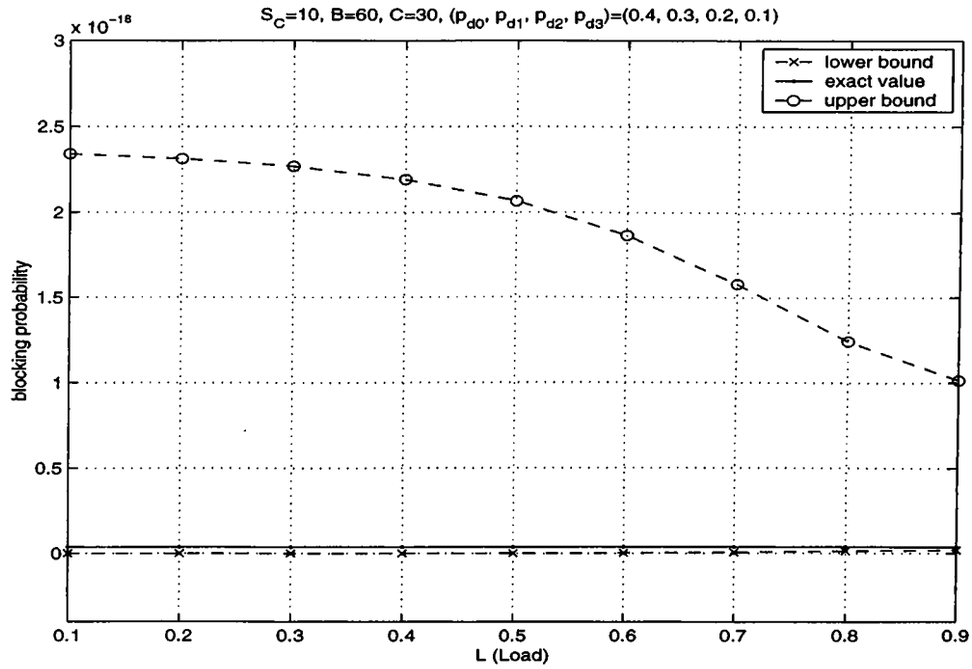


(a)

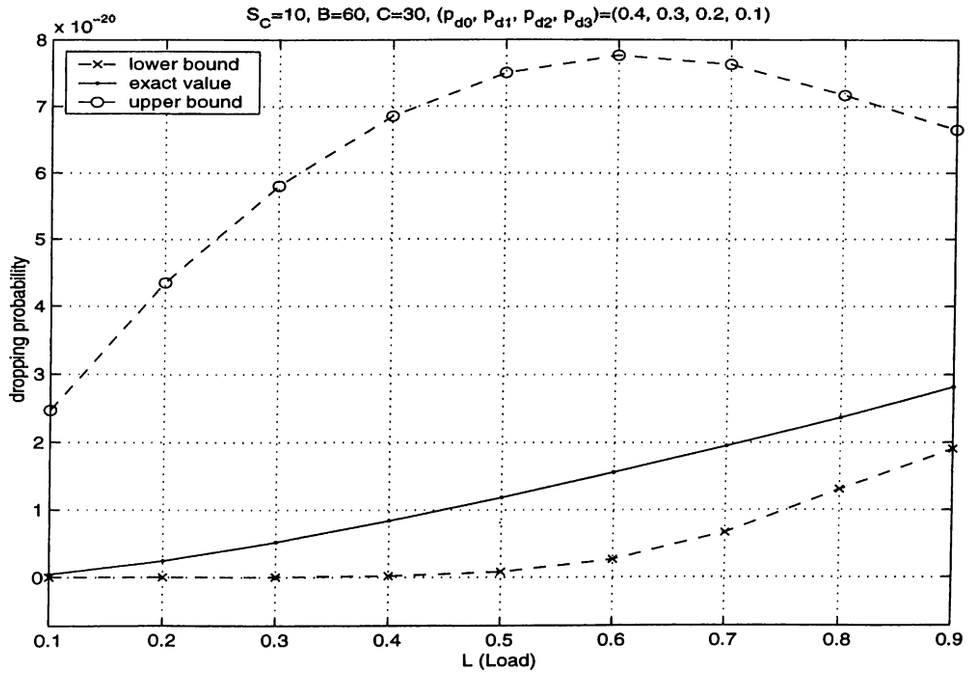


(b)

Figure 6.4: Blocking and dropping probabilities for $S_C = 1$ when $B = 60$ and $C = 30$.

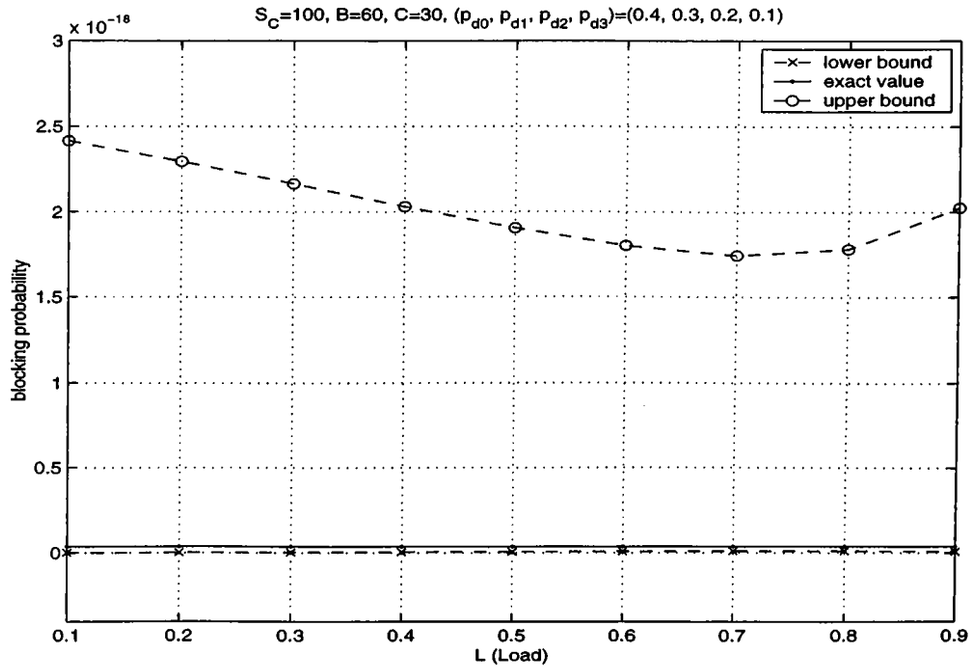


(a)

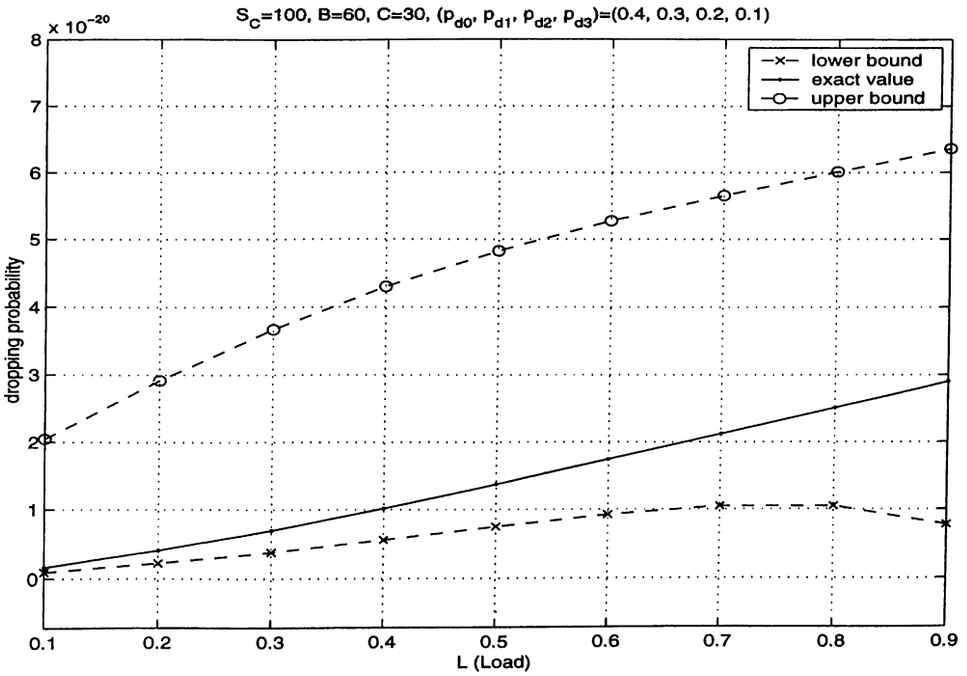


(b)

Figure 6.5: Blocking and dropping probabilities for $S_C = 10$ when $B = 60$ and $C = 30$.



(a)



(b)

Figure 6.6: Blocking and dropping probabilities for $S_C = 100$ when $B = 60$ and $C = 30$.

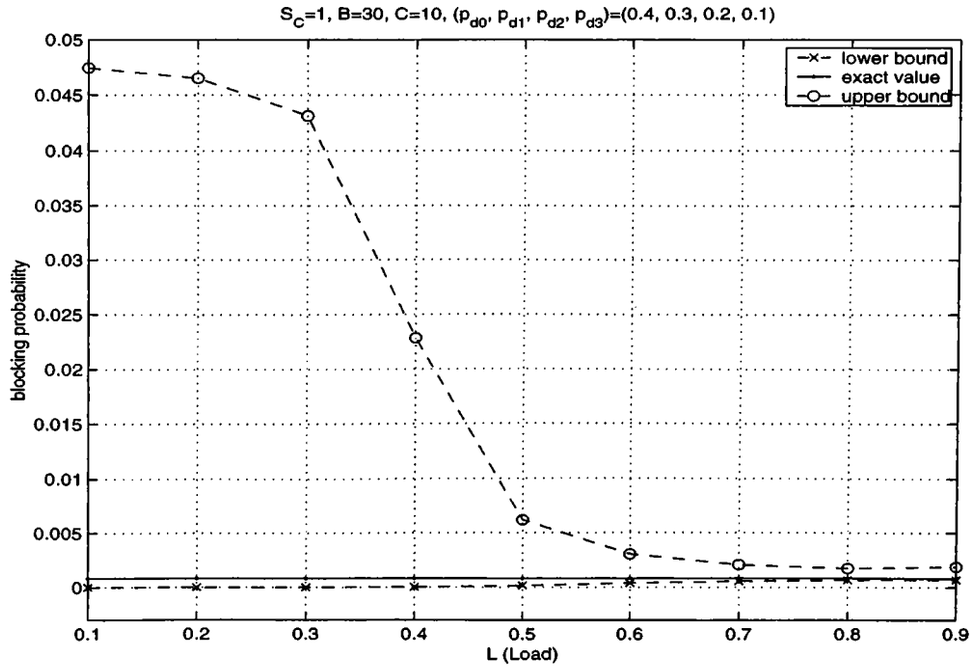
can be easily accommodated by a workstation (see section 5.2.5 for the space complexity of Algorithm 1).

The bounds computed on p_{block} and p_{drop} using Algorithm 1 are highly acceptable; the bounds on p_{drop} are especially tight. Note that the $4(B+1)$ steady state probabilities used in computing p_{block} comprise those $3(C+1)$ used in computing p_{drop}. If we remove the unreachable states from the two formulae, there happens to be exactly $[4(B+1)-2]$ steady state probabilities that contribute to p_{block} and 6 that contribute to p_{drop}. This can be an an intuitive explanation for having tighter bounds for p_{drop} compared to those for p_{block}. There are other factors that influence the quality of the computed bounds such as the NCD partitioning employed, the ordering chosen by our heuristic within each NCD block, and the irreducibility structure of the computed st-monotone matrices.

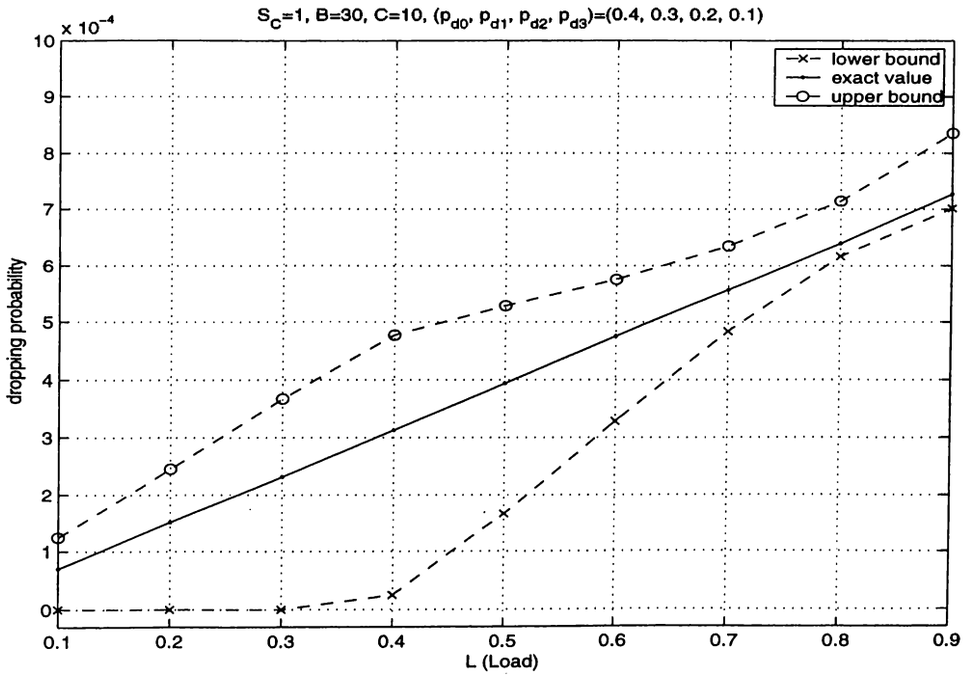
In our experiments, we observed that the location of each state inside an NCD block has considerable influence on its componentwise bound. Recall that the ordering of the states inside each NCD block is determined by the heuristic in Algorithm 10 in Chapter 3. It is generally the case that the closer the state to the end of its corresponding NCD block, the better the componentwise bounds obtained on it. Therefore, in order to see the effect of this conjecture, we place the 6 states of interest that contribute to p_{drop} at the end of their corresponding NCD blocks and order the remaining states due to the heuristic given in Algorithm 10. For example, in Figures 6.1.(a)-(b), 3 of these 6 states are in the 11th block, 2 are in the 10th block, and 1 is in the 9th block. If there exists more than one state that contribute to p_{drop} in the same block (for example, block 11 in the system given in Figures 6.1.(a)-(b) has 3 of them), we select the state which has the largest self transition probability among the states of interest, make it the last state in the block, and place the others in the preceding locations at the end of the block. After positioning the states of interest, Algorithm 10 is applied to find the ordering of remaining states. This improvement has one more advantage. As we mentioned in earlier chapters, bounding matrices may be reducible. Moreover, the identity and number of states in the essential classes of these bounding matrices are unpredictable. The bounding matrix algorithms generate more nonzeros towards the bottom of the computed matrix. Because of

this reason, if a state is placed towards the end of its NCD block, it will most likely be placed in the essential class of its corresponding bounding model. It is expected that if the states of interest reside in the essential class, one will have better componentwise bounds.

Considering these, we made the above change and reran all of the experiments for the systems given in Figures 6.1, 6.2, 6.3, 6.4, 6.5, and 6.6. The improvement affected the results for the systems given in Figures 6.1.(a)-(b) and 6.4.(a)-(b). The new bounds computed for these systems are presented in Figures 6.7 and 6.8. The bounds for the systems in Figures 6.2.(a)-(b), 6.3.(a)-(b), 6.5.(a)-(b), and 6.6.(a)-(b) did not change. Especially for low loads, we obtained considerable improvement on the bounds for the 6 states of interest that contribute to p_{drop} for the systems $S_C = 1$, $B = 30$, $C = 10$ and $S_C = 1$, $B = 60$, $C = 30$. For the systems with $S_C = 10$ and $S_C = 100$ these 6 states of interest have larger steady state probabilities than the systems with $S_C = 1$. Moreover, the systems with $S_C = 10$ and $S_C = 100$ have more unbalanced steady state probabilities than the systems with $S_C = 1$. In addition to these, some of these 6 states are already placed towards the end of their NCD blocks when $S_C = 10$ and $S_C = 100$. This is an intuitive explanation for not having any improvement in the bounds for the systems with $S_C = 10$ and $S_C = 100$ when we place the 6 states of interest towards the end of their NCD blocks. In our experimental runs with different types of applications we observed that Algorithm 1 is more useful for finding bounds on states with larger steady state probability mass.

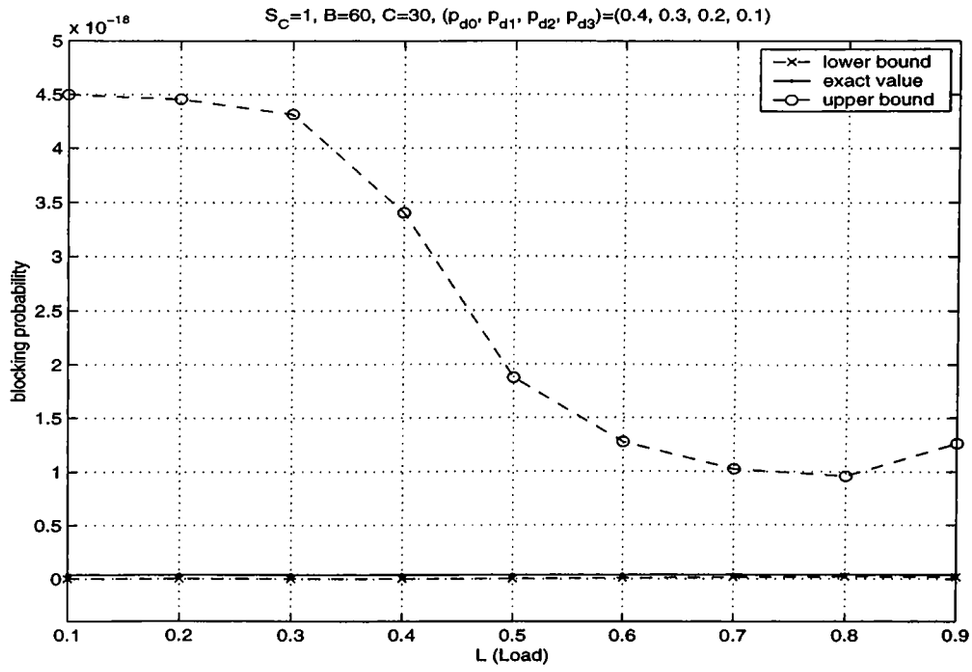


(a)

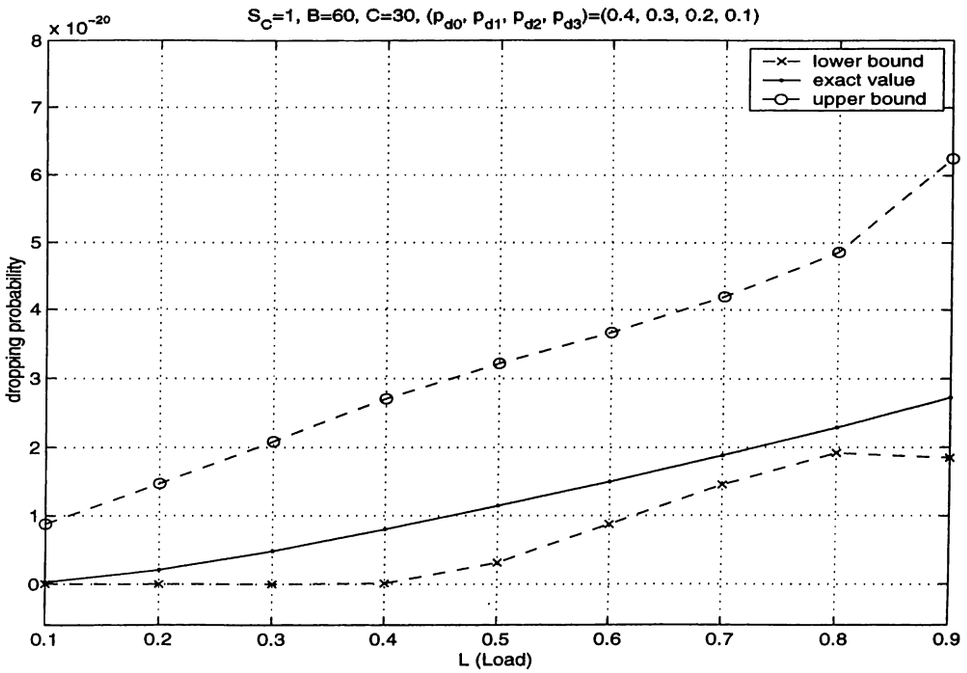


(b)

Figure 6.7: Blocking and dropping probabilities for $S_C = 1$ when $B = 30$ and $C = 10$ after the improvement.



(a)



(b)

Figure 6.8: Blocking and dropping probabilities for $S_C = 1$ when $B = 60$ and $C = 30$ after the improvement.

Chapter 7

Conclusion

In this thesis, we gave an algorithm to find bounds on performance measures of real-life systems which possess the NCD structure when they are modeled by a MC. Bounds on performance measures are obtained by an improved version of a componentwise bounding algorithm for the steady state vector of NCD MCs. The given two-level algorithm uses aggregation, stochastic comparison with the strong stochastic (st) order. In order to improve accuracy, it employs reordering of the states and a better componentwise probability bounding algorithm given st upper- and lower-bounding probability vectors. A thorough analysis of the algorithm from the point of view of having one essential class in a bounding matrix is provided. The essential class in bounding matrices is obtained using an efficient version of ordinary strongly connected component search algorithm on graphs. Moreover, a linked list implementation of the direct solver GTH is considered to compute steady state vectors of bounding matrices. Most of the implementation of the proposed algorithm is done in sparse storage to make this work applicable to large sparse systems and to benefit from the advantage of not dealing with zero entries during computation.

We applied our bounding algorithm to a wireless asynchronous transfer mode (ATM) network and gave bounds on the performance measures for this model. Some of the bounds are improved by placing the states which contribute to the performance measures at the end of their corresponding blocks. The run-time

of the algorithm is much better than that of GTH and iterative aggregation-disaggregation in sparse storage and the quality of the computed bounds on steady state probabilities are highly acceptable for the chosen application.

From the wireless ATM model we experienced that our componentwise bounding algorithm is very effective in NCD MCs with highly unbalanced steady state probabilities and a small number of states accumulating a large probability mass. In addition to this, the states of a system that contribute to the performance measures of interest should be the ones that are placed towards the end of their NCD blocks. Finally, a small degree of coupling in the partitioning is very important in obtaining acceptable componentwise bounds.

Future work may focus on implementing Algorithm 1 for transient analysis. Moreover, Algorithm1 must be applied to other problems to make stronger generalizations on the conditions that makes the algorithm useful.

Bibliography

[1] Abu-Amsha O. and Vincent J.-M., An algorithm to bound functionals of Markov chains with large state space, *Rapport de recherche MAI n 25, IMAG*, (1996).

[2] Baase S., *Computer Algorithms*, Addison-Wesley, Reading, MA (1988)

[3] Berman A. and Plemmons R.J., *Nonnegative Matrices in the Mathematical Sciences*, SIAM Press, Philadelphia (1994).

[4] Courtois P.-J., *Decomposability: Queueing and Computer System Applications*, Academic Press, New York (1977).

[5] Courtois P.-J. and Semal P., Bounds for the positive eigenvectors of nonnegative matrices and for their approximations by decomposition, *Journal of the Assocation for Computer Machinery* **31**(4) 804–825 (1984).

[6] Dayar T., Permuting Markov chains to nearly completely decomposable form. *Tech. Report BU-CEIS-9808, Department of Computer Engineering, Bilkent University, Ankara, Turkey*, (1998); available via ftp from ftp://ftp.cs.bilkent.edu.tr/pub/tech-reports/1998/BU-CEIS-9808.ps.z.

[7] Dayar T. and Pekergin N., Stochastic comparison, reorderings, and nearly completely decomposable Markov chains, In *Proceedings of the International Conference on the Numerical Solution of Markov Chains (NSMC'99)*, (Edited by B. Plateau *et al.*), pp. 228–246, Prensas Universitarias de Zaragoza, Spain (1999).

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

[9] Dayar T. and Stewart W.J., On the effects of using the Grassmann-Taksar-Heyman method in iterative aggregation-disaggregation, *SIAM Journal on Scientific Computing* **17**(1) 287–303 (1996).

[10] Duff I.S., Erisman A.M., and Reid J.K., *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford (1986).

[11] Grassmann W.K., Taksar M.I., and Heyman D.P., Regenerative analysis and steady state distributions for Markov chains, *Operations Research* **33**(5) 1107–1116 (1985).

[12] Kleinrock L., *Queueing Systems Volume 1: Theory*, John Wiley & Sons, New York (1975).

[13] Keilson J. and Kester A., Monotone matrices and monotone Markov processes, *Stochastic Processes and their Applications* **5** 231–241 (1977).

[14] Massey W.A., Stochastic orderings for Markov Processes on partially ordered spaces, *Mathematics of Operations Research* **12**(2) 350–367 (1987).

[15] Meyer C.D., Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems, *SIAM Review* **31**(2) 240–272 (1989).

[16] Pekergin N., Stochastic performance bounds by state reduction, *Performance Evaluation* **36–37** 1–17 (1999).

[17] Pekergin N., Stochastic delay bounds on fair queueing algorithms, In *Proceedings of INFOCOM'99*, pp. 1212–1220, New York, (1999).

[18] Pekergin N., Dayar T., and Alparslan D.N., Componentwise bounds for nearly completely decomposable Markov chains using stochastic comparison and reordering, submitted for publication (2000).

[19] Semal P., Analysis of large Markov models, bounding techniques and applications, *Doctoral Thesis, Université Catholique de Louvain, Belgium*, (1992).

[20] Shaked M. and Shantikumar J.G., *Stochastic Orders and Their Applications*, Academic Press, California (1994).

[21] Stewart W.J., *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, New Jersey (1994).

[22] Stoyan D., *Comparison Methods for Queues and Other Stochastic Models*, John Wiley & Sons, Berlin, Germany (1983).

[23] Tremolieres M., Vincent J.-M., and Plateau B., Determination of the optimal upper bound of a Markovian generator, *Technical Report 106, LGI-IMAG*, (1992).

[24] Truffet L., Near complete decompasibility: bounding the error by stochastic comparison method, *Advances in Applied Probability* **29** 830–855 (1997).

[25] Vèque V. and Ben-Othman J., MRAP: A multiservices resource allocation policy for wireless ATM network, *Computer Networks and ISDN systems*, **29** 2187–2200 (1998).